Hepatitis Disease Prediction

Classification Modeling with K-Fold Cross Validation

Sean O'Malley

Objective : Predict the presence of hepatitis in patients

Methodology 1 : Support Vector Machine with K-Fold Cross Validation using e1071 package

Methodology 2 : Artificial Neural Net using neuralnet package with K-Fold Cross Validation using for loop

Data Manipulation Methods : Over 40% of the observations within the dataset contain a null values in way that is not consistent to any specific feature of the data, so to get a full view of the data we have to work with I have chosen two different data sets to run through our two models.

- **hep_clean** omits all observations containing null values
- **hep_replace** replaces the missing values with the mean of the respective feature that observation is associated with

```
hep <- read.delim("http://archive.ics.uci.edu/ml/machine-learning-databases/hepatitis/hepatitis.da
ta",
                  sep = ",",
                  na.strings = '?',
                  col.names = c("class", "age", "sex", "steroid", "antivirals", "fatigue", "mala
ise", "anorexia", "liver_big", "liver_firm", "spleen_palpable", "spiders", "ascites", "varices",
"bilirubin", "alk_phosphate", "sgot", "albumin", "protime", "histology"))

# Clean Data Set Manipulation
hep_clean <- na.omit(hep)
#hep_clean$class <- as.factor(hep_clean$class)

summary(hep_clean)
```

```
##      class            age              sex           steroid
##  Min.   :1.000   Min.   :20.00   Min.   :1.000   Min.   :1.000
##  1st Qu.:2.000   1st Qu.:32.00   1st Qu.:1.000   1st Qu.:1.000
##  Median :2.000   Median :38.50   Median :1.000   Median :2.000
##  Mean   :1.837   Mean   :40.66   Mean   :1.137   Mean   :1.525
##  3rd Qu.:2.000   3rd Qu.:49.25   3rd Qu.:1.000   3rd Qu.:2.000
##  Max.   :2.000   Max.   :72.00   Max.   :2.000   Max.   :2.000
##    antivirals       fatigue        malaise         anorexia
##  Min.   :1.000   Min.   :1.00   Min.   :1.000   Min.   :1.00
##  1st Qu.:1.000   1st Qu.:1.00   1st Qu.:1.000   1st Qu.:2.00
##  Median :2.000   Median :1.00   Median :2.000   Median :2.00
##  Mean   :1.738   Mean   :1.35   Mean   :1.613   Mean   :1.85
##  3rd Qu.:2.000   3rd Qu.:2.00   3rd Qu.:2.000   3rd Qu.:2.00
##  Max.   :2.000   Max.   :2.00   Max.   :2.000   Max.   :2.00
##    liver_big       liver_firm    spleen_palpable    spiders
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:2.000   1st Qu.:1.000   1st Qu.:2.000   1st Qu.:1.000
##  Median :2.000   Median :2.000   Median :2.000   Median :2.000
##  Mean   :1.837   Mean   :1.525   Mean   :1.812   Mean   :1.688
##  3rd Qu.:2.000   3rd Qu.:2.000   3rd Qu.:2.000   3rd Qu.:2.000
##  Max.   :2.000   Max.   :2.000   Max.   :2.000   Max.   :2.000
##     ascites        varices        bilirubin     alk_phosphate
##  Min.   :1.00   Min.   :1.000   Min.   :0.300   Min.   : 26.00
##  1st Qu.:2.00   1st Qu.:2.000   1st Qu.:0.700   1st Qu.: 68.25
##  Median :2.00   Median :2.000   Median :1.000   Median : 85.00
##  Mean   :1.85   Mean   :1.875   Mean   :1.221   Mean   :102.91
##  3rd Qu.:2.00   3rd Qu.:2.000   3rd Qu.:1.300   3rd Qu.:133.50
##  Max.   :2.00   Max.   :2.000   Max.   :4.800   Max.   :280.00
##       sgot           albumin         protime         histology
##  Min.   : 14.00   Min.   :2.100   Min.   :  0.00   Min.   :1.000
##  1st Qu.: 30.75   1st Qu.:3.500   1st Qu.: 46.00   1st Qu.:1.000
##  Median : 56.50   Median :4.000   Median : 62.00   Median :1.000
##  Mean   : 82.03   Mean   :3.844   Mean   : 62.51   Mean   :1.413
##  3rd Qu.:102.75   3rd Qu.:4.200   3rd Qu.: 77.25   3rd Qu.:2.000
##  Max.   :420.00   Max.   :5.000   Max.   :100.00   Max.   :2.000
```

```
# Replace Data Set Manipulation
replace_fun <- function(x) {
                           for(i in 1:ncol(x)){x[is.na(x[,i]), i] <- mean(x[,i], na.rm = TRUE)}
                           return(as.data.frame(x))
}

hep_replace <- replace_fun(hep)

hep_replace$class <- round(hep_replace$class, 0)

summary(hep_replace)
```

```
##      class            age              sex           steroid
##  Min.   :1.000   Min.   : 7.00   Min.   :1.000   Min.   :1.00
##  1st Qu.:2.000   1st Qu.:32.00   1st Qu.:1.000   1st Qu.:1.00
##  Median :2.000   Median :39.00   Median :1.000   Median :2.00
##  Mean   :1.792   Mean   :41.27   Mean   :1.097   Mean   :1.51
##  3rd Qu.:2.000   3rd Qu.:50.00   3rd Qu.:1.000   3rd Qu.:2.00
##  Max.   :2.000   Max.   :78.00   Max.   :2.000   Max.   :2.00
##    antivirals       fatigue         malaise         anorexia
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:2.000   1st Qu.:1.000   1st Qu.:1.000   1st Qu.:2.000
##  Median :2.000   Median :1.000   Median :2.000   Median :2.000
##  Mean   :1.844   Mean   :1.346   Mean   :1.601   Mean   :1.791
##  3rd Qu.:2.000   3rd Qu.:2.000   3rd Qu.:2.000   3rd Qu.:2.000
##  Max.   :2.000   Max.   :2.000   Max.   :2.000   Max.   :2.000
##    liver_big       liver_firm   spleen_palpable    spiders
##  Min.   :1.000   Min.   :1.00   Min.   :1.000   Min.   :1.000
##  1st Qu.:2.000   1st Qu.:1.00   1st Qu.:2.000   1st Qu.:1.000
##  Median :2.000   Median :2.00   Median :2.000   Median :2.000
##  Mean   :1.833   Mean   :1.58   Mean   :1.799   Mean   :1.658
##  3rd Qu.:2.000   3rd Qu.:2.00   3rd Qu.:2.000   3rd Qu.:2.000
##  Max.   :2.000   Max.   :2.00   Max.   :2.000   Max.   :2.000
##     ascites         varices        bilirubin    alk_phosphate
##  Min.   :1.000   Min.   :1.000   Min.   :0.30   Min.   : 26.0
##  1st Qu.:2.000   1st Qu.:2.000   1st Qu.:0.80   1st Qu.: 78.0
##  Median :2.000   Median :2.000   Median :1.00   Median :102.0
##  Mean   :1.866   Mean   :1.879   Mean   :1.43   Mean   :105.5
##  3rd Qu.:2.000   3rd Qu.:2.000   3rd Qu.:1.50   3rd Qu.:119.8
##  Max.   :2.000   Max.   :2.000   Max.   :8.00   Max.   :295.0
##       sgot           albumin         protime         histology
##  Min.   : 14.00   Min.   :2.100   Min.   :  0.00   Min.   :1.000
##  1st Qu.: 33.00   1st Qu.:3.500   1st Qu.: 57.00   1st Qu.:1.000
##  Median : 59.50   Median :3.900   Median : 61.85   Median :1.000
##  Mean   : 86.35   Mean   :3.816   Mean   : 61.85   Mean   :1.455
##  3rd Qu.: 99.50   3rd Qu.:4.200   3rd Qu.: 65.50   3rd Qu.:2.000
##  Max.   :648.00   Max.   :6.400   Max.   :100.00   Max.   :2.000
```

Overview and Intent for Tuning Models

Tuning parameters usually regulate the model complexity and hence are a key ingredient for any predictive task. We care about the choice of tuning parameter because they are closely linked with the accuracy of predictions returned by the model. A predictive model is considered good when it is capable of predicting previously unseen samples with high accuracy. The accuracy of a model's predictions is usually gauged using a loss function, such as the mean-squared error for continuous outputs, or the 1- 0 loss for a categorical outcome.

Problems :

We can identify two areas where prediction error occurs, in the training error or in the test error. The training error gets smaller as long as the predicted responses are close to the observed responses, and will get larger if for some of the observations, the predicted and observed responses differ substantially.

training error – is calculated using the training sample used to fit the model and training errors decrease monotonically as the model gets more complicated.

test error – is calculated using the training sample used to fit the model and testing errors initially decreases, then, from a certain flexibility level, starts increasing again.

overfitting – tendency of a model to adapt too well to the training data

bias-variance tradeoff – the problem of simultaneously minimizing two sources of error that prevent supervised learning algorithms from generalizing beyond their training set.

- The bias is error from erroneous assumptions in the learning algorithm to miss the relevant relations between features and target outputs, or underfitting.
- The variance is error from sensitivity to small fluctuations in the training set. High variance can cause overfitting: modeling the random noise in the training data, rather than the intended outputs.

bias-variance decomposition – a way of analyzing a learning algorithm's expected generalization error (how accurately an algorithm is able to predict outcome values for previously unseen data) with respect to a particular problem as a sum of three terms, the bias, variance and the irreducible error, resulting from noise in the problem itself.

Solution: **Cross Validation**

In totality, the problems we face suggest that it is not advisable to compare the predictive accuracy using the same observations used for estimating models. However in many cases it is not possible to draw a large independent set of observations for testing the model's performance, because collecting data is typically an expensive activity. A classic 70/30 test training split is a solid suggestion to this problem, but the output is often quite noisy.

A possible solution is to use **cross validation**. In its basic version, the k-fold cross validation, the samples are randomly partitioned into k“>kk sets, called folds, of roughly equal size.

- A model is fit using all the samples except the first subset.
- The prediction error of the model is calculated using the first held out samples.
- This operation is repeated for each fold and the model's performance is calculated by averaging the errors across the different test sets.
- k“>kk is usually fixed at 5 to 10.

Cross validation provides an estimate of the test error for each model. Cross validation is one of the most widely used method for model selection, and for choosing tuning parameter values.

Initial Test/Train Data Split Function

```
split_data <- function(x, p = 0.7, s = 777) {
  set.seed(s)
  index = sample(1:dim(x)[1])
  train = x[index[1:floor(dim(x)[1]*p)],]
  test = x[index[((ceiling(dim(x)[1]*p))+1):dim(x)[1]],]
  return(list(train = train, test = test))
}


# Apply to Clean Data Set
hep_clean_split <- split_data(hep_clean)

hep_clean_test <- hep_clean_split$test

hep_clean_train <- hep_clean_split$train


# Apply to Replace Data Set
hep_replace_split <- split_data(hep_replace)

hep_replace_test <- hep_replace_split$test

hep_replace_train <- hep_replace_split$train
```

Radial SVM without K-Fold Cross Validation Using Clean Data Set

```
svm1 <- svm(class~., data = hep_clean_train, kernel = "radial", cost = 1, gamma = 1/ncol(hep_clean
_train))
```

Use to Predict Results

```
predict1 <- predict(svm1, hep_clean_test, type = "response")

predict1 <- round(predict1, 0)
```

View Predicted Results

```
cor(predict1, hep_clean_test$class)
```

```
## [1] 0.4662524
```

```
confusionMatrix(predict1, data = hep_clean_test$class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1  1  3
##          2  0 20
##
##                Accuracy : 0.875
##                  95% CI : (0.6764, 0.9734)
##     No Information Rate : 0.9583
##     P-Value [Acc > NIR] : 0.9836
##
##                   Kappa : 0.3571
##  Mcnemar's Test P-Value : 0.2482
##
##             Sensitivity : 1.00000
##             Specificity : 0.86957
##          Pos Pred Value : 0.25000
##          Neg Pred Value : 1.00000
##              Prevalence : 0.04167
##          Detection Rate : 0.04167
##    Detection Prevalence : 0.16667
##       Balanced Accuracy : 0.93478
##
##        'Positive' Class : 1
##
```

Interpret Predicted Results

Given this is our most bare bones model, we are doing pretty well with an accuracy of 0.875. However looking at a kappa of only 0.3571, very poor p value, and a specificity of 0.86 does show room for improvement. Especially considering that we are attempting to predict the presense of a disease, which has much larger reciprocations for false negatives.

Radial SVM with K-Fold Cross Validation Using Clean Data Set

To obtain the optimum classification model via obtaining the minimum value error, we will perform a 10 fold cross validation using the tune.svm function in the e1071 package.

```
svm1_tuned <- tune.svm(class~., data = hep_clean_train, cost = 10^2, kernel = "radial",
                       gamma = 1/ncol(hep_clean_train),
                       tunecontrol=tune.control(cross = 10))
```

Summary of Tuned Model

```
summary(svm1_tuned)
```

```
##
## Error estimation of 'svm' using 10-fold cross validation: 0.07558753
```

```
svm1_tuned$performances
```

```
##   gamma cost      error dispersion
## 1  0.05  100 0.07558753 0.06192425
```

```
svm1_tuned_fit <- svm1_tuned$best.model
```

Use to Predict Results

```
predict1.2 <- predict(svm1_tuned_fit, hep_clean_test)

predict1.2 <- round(predict1.2, 0)
```

View Predicted Results

```
cor(predict1.2, hep_clean_test$class)
```

```
## [1] 0.1
```

```
confusionMatrix(predict1.2, data = hep_clean_test$class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1  1  3
##          2  3 17
##
##                Accuracy : 0.75
##                  95% CI : (0.5329, 0.9023)
##     No Information Rate : 0.8333
##     P-Value [Acc > NIR] : 0.9088
##
##                   Kappa : 0.1
##  Mcnemar's Test P-Value : 1.0000
##
##             Sensitivity : 0.25000
##             Specificity : 0.85000
##          Pos Pred Value : 0.25000
##          Neg Pred Value : 0.85000
##              Prevalence : 0.16667
##          Detection Rate : 0.04167
##    Detection Prevalence : 0.16667
##       Balanced Accuracy : 0.55000
##
##        'Positive' Class : 1
##
```

Interpret Predicted Results

Well, that didn't turn out as planned. By tuning our previous model with a kfold of 10, our model accuracy and overall metrics actually decreased to a very poor set of indicator outputs.

Radial SVM 3 with K-Fold Cross Validation Using Clean Data Set

Lets try another one of e1071's cool features, which is the best.svm feature. This does much of the work for us in terms of outputting the best fit model given a fairly bare bones amount of imputs. We are again establishing a kfold of 10, however with minimal imputs to see if we can beat our initial SVM model using tuning and the clean dataset.

```
svm1_best <- best.svm(class~., data = hep_clean_train,
                      tunecontrol=tune.control(cross = 10))

predict1.3 <- predict(svm1_best, hep_clean_test)

predict1.3 <- round(predict1.3, 0)

confusionMatrix(predict1.3, data = hep_clean_test$class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1  1  3
##          2  0 20
##
##                Accuracy : 0.875
##                  95% CI : (0.6764, 0.9734)
##     No Information Rate : 0.9583
##     P-Value [Acc > NIR] : 0.9836
##
##                   Kappa : 0.3571
##  Mcnemar's Test P-Value : 0.2482
##
##             Sensitivity : 1.00000
##             Specificity : 0.86957
##          Pos Pred Value : 0.25000
##          Neg Pred Value : 1.00000
##              Prevalence : 0.04167
##          Detection Rate : 0.04167
##    Detection Prevalence : 0.16667
##       Balanced Accuracy : 0.93478
##
##        'Positive' Class : 1
##
```

Interesting, we were actually able to attain the exact accuracy of our first model, suggesting that our first model actually was about as accurate as it was going to get, maybe? Nevertheless, lets move on to SVM's using the full replaced dataset to see if that improves our predictive power.

Radial SVM Using Replace Data Set

```
svm2 <- svm(class~., data = hep_replace_train, kernel = "radial", cost = 1, gamma = 1/ncol(hep_rep
lace_train))
```

Use to Predict Results

```
predict2 <- predict(svm2, hep_replace_test, type = "response")

predict2 <- round(predict2, 0)
```

View Predicted Results

```
cor(predict2, hep_replace_test$class)
```

```
## [1] 0.314373
```

```
confusionMatrix(predict2, data = hep_replace_test$class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1  4  4
##          2  6 32
##
##                Accuracy : 0.7826
##                  95% CI : (0.6364, 0.8905)
##     No Information Rate : 0.7826
##     P-Value [Acc > NIR] : 0.5837
##
##                   Kappa : 0.3114
##  Mcnemar's Test P-Value : 0.7518
##
##             Sensitivity : 0.40000
##             Specificity : 0.88889
##          Pos Pred Value : 0.50000
##          Neg Pred Value : 0.84211
##              Prevalence : 0.21739
##          Detection Rate : 0.08696
##    Detection Prevalence : 0.17391
##       Balanced Accuracy : 0.64444
##
##        'Positive' Class : 1
##
```

Interpret Predicted Results

Given our replaced dataset our accuracy in predicting the test dataset has actually decreased from the smaller, clean dataset. Note that we are predicting the class of disease based upon a test and train group that have both had missing values replaced my the mean of the respected feature. I believe this decrease in perfect accuracy does expose its negative repercussions in the original SVM, but in theory should be largely improved by the k-fold process given the larger dataset to maximize the advantage of the kfold process.

Radial SVM with K-Fold Cross Validation Using Replace Data Set

```
svm2.2 <- svm(class~., data = hep_clean_train, cost = 10^2, kernel = "radial",
              gamma = 1/ncol(hep_clean_train), tunecontrol=tune.control(cross = 10))
```

Use to Predict Results

```
predict2.2 <- predict(svm2.2, hep_replace_test, type = "response")

predict2.2 <- round(predict2.2, 0)
```

View Predicted Results

```
cor(predict2.2, hep_replace_test$class)
```

```
## [1] 0.3925657
```

```
confusionMatrix(predict2.2, data = hep_replace_test$class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1  3  5
##          2  2 36
##
##                Accuracy : 0.8478
##                  95% CI : (0.7113, 0.9366)
##     No Information Rate : 0.8913
##     P-Value [Acc > NIR] : 0.8789
##
##                   Kappa : 0.3784
##  Mcnemar's Test P-Value : 0.4497
##
##             Sensitivity : 0.60000
##             Specificity : 0.87805
##          Pos Pred Value : 0.37500
##          Neg Pred Value : 0.94737
##              Prevalence : 0.10870
##          Detection Rate : 0.06522
##    Detection Prevalence : 0.17391
##       Balanced Accuracy : 0.73902
##
##        'Positive' Class : 1
##
```

Interpret Predicted Results

Great, I was actually able to increase the accuracy of the replace dataset prediction by and accuracy and kappa of nearly 6%! When considering the confusion matrix, minimizing the False Negative rate is by far the highest priority, so in the subsequent models, we'd like to reduce that FN number of 5. We do not want to send someone sick home thinking that they are okay.

The model is great, that in theory, this provides some proof that k-folds work better when they have sufficient data to work with, and that if I were to put this algorithm into production, as it accumulated futher data, that the k-fold has shown that it can continue to improve the accuracy of the model.

Artificial Neural Net

Linear ANN using Clean Data Set

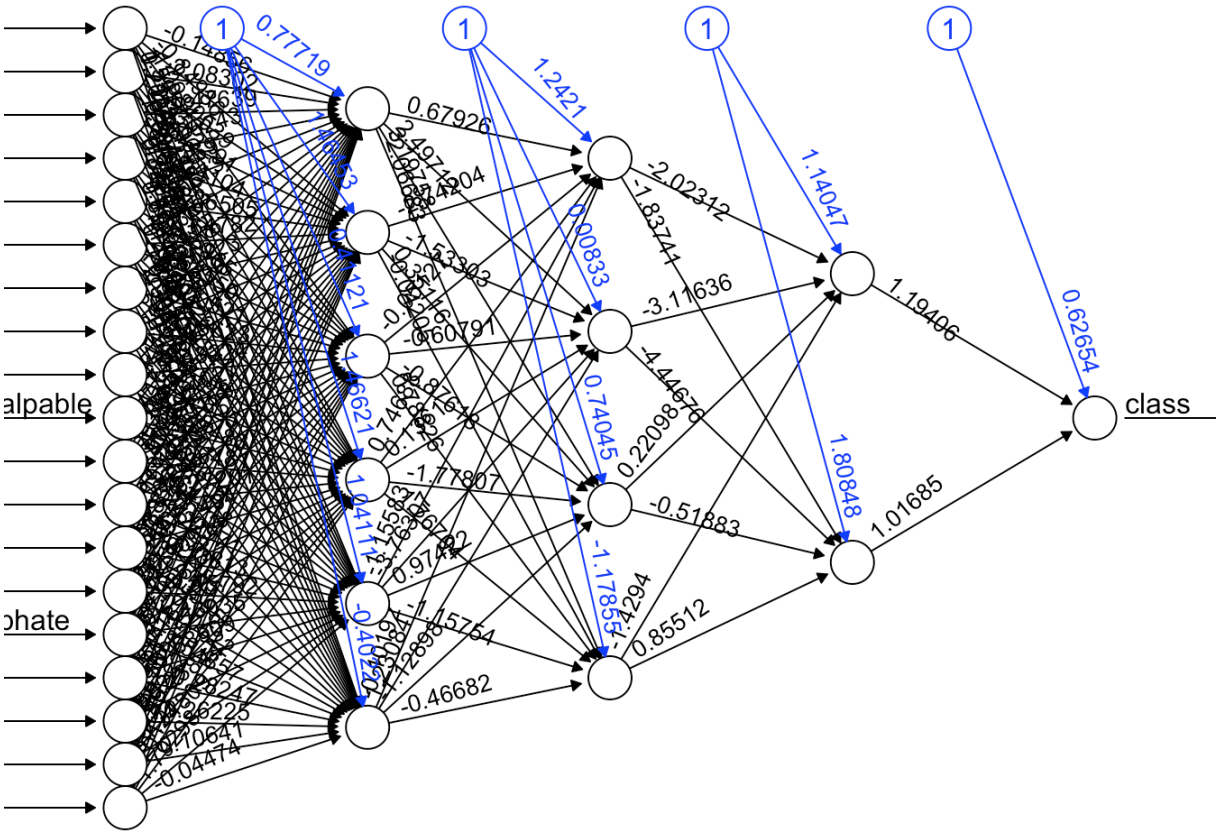
Build Model

We set the classification formula, source of data, number of hidden units,

```
hep_names <- colnames(hep)
hep_predictors <- hep_names[!hep_names%in%"class"]
hep_predictors <- paste(hep_predictors,collapse = "+")
hep_formula <- as.formula(paste("class~", hep_predictors, collapse = "+"))

nnet1 <- neuralnet(formula = hep_formula, hidden = c(6,4,2), linear.output = T, data = hep_clean_t
rain)

plot(nnet1, rep = "best")
```



Predict with Model

```
nnet1_predict <- compute(nnet1, hep_clean_test[2:20])

nnet1_predict$net.result <- round(nnet1_predict$net.result, 0)
```

View Predicted Results

```
results1 <- data.frame(actual = hep_clean_test$class, prediction = nnet1_predict$net.result)

# Now test prediction strength with a correllation between the actual and predicted values

cor(results1$prediction, results1$actual)
```

```
## [1] 0.269679945
```

```
confusionMatrix(results1$prediction, results1$actual)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1  1  1
##          2  3 19
##
##                Accuracy : 0.8333333
##                  95% CI : (0.6261583, 0.9526464)
##     No Information Rate : 0.8333333
##     P-Value [Acc > NIR] : 0.6293784
##
##                   Kappa : 0.25
##  Mcnemar's Test P-Value : 0.6170751
##
##             Sensitivity : 0.25000000
##             Specificity : 0.95000000
##          Pos Pred Value : 0.50000000
##          Neg Pred Value : 0.86363636
##              Prevalence : 0.16666667
##          Detection Rate : 0.04166667
##    Detection Prevalence : 0.08333333
##       Balanced Accuracy : 0.60000000
##
##        'Positive' Class : 1
##
```

Interpret Predicted Results

I wanted to create a simple yet robust neural net as a great starting piont before we embark on the cross validation process. We ended up with great results. We have an accuracy of 87%, of which only one false negative was recorded. Reminder that we want to avoid these the most because the implications can be deadly if we falsely predict that someone is not sick. Looking to our other metrics, our kappa of 0.5 is acceptable, but not great; our p-value needs to improve and our sensitivity is pretty poor. This could be simply a result of the small sample size, but we need to see if we can make this model better using the clean data via cross validation next.

Linear ANN using Clean Data Set with Cross Validation

Build and Predict with Model

```
cv.error <- NULL

set.seed(450)

k <- 10

require(plyr)
```

```
## Loading required package: plyr
```

```
## ------------------------------------------------------------------------------
```

```
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)
```

```
## ------------------------------------------------------------------------------
```

```
##
## Attaching package: 'plyr'
```

```
## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
pbar <- create_progress_bar('text')
pbar$init(k)
```

```
##
  |
  |                                                                      |   0%
```

```
for(i in 1:k){
    index <- sample(1:nrow(hep_clean_train),round(0.9*nrow(hep_clean_train)))

    train.cv <- hep_clean_train[index,]

    test.cv <- hep_clean_train[-index,]

    nn <- neuralnet(formula = hep_formula, hidden = c(6,4,2), linear.output = T, data = train.cv)

    pr.nn <- compute(nnet1, test.cv[2:20])

    pr.nn <- pr.nn$net.result*(max(hep_clean_train$class)-min(hep_clean_train$class))+min(hep_clea
n_train$class)

    pr.nn <- round(pr.nn, 0)

    test.cv.r <- (test.cv$class)*(max(hep_clean_train$class)-min(hep_clean_train$class))+min(hep_c
lean_train$class)

    cv.error[i] <- sum((test.cv.r - pr.nn)^2)/nrow(test.cv)

    pbar$step()
}
```

```
##
  |
  |======                                                           |  10%
  |
  |=============                                                    |  20%
  |
  |====================                                             |  30%
  |
  |==========================                                       |  40%
  |
  |================================                                 |  50%
  |
  |=======================================                          |  60%
  |
  |==============================================                   |  70%
  |
  |=====================================================            |  80%
  |
  |============================================================     |  90%
  |
  |=================================================================| 100%
```

```
cv.error
```

```
##  [1] 0.0000000000 0.0000000000 0.0000000000 0.1666666667 0.0000000000
##  [6] 0.0000000000 0.1666666667 0.3333333333 0.1666666667 0.1666666667
```

```
mean(cv.error)
```

```
## [1] 0.1
```

Interpret Results

The interpretation of a cross validation using a neural net is much tougher, because the process often has to be done manually, accomplish this cross validation, I created a for loop that repartitioned the test and train group into a 90/10 split and then had the loop itereate through a k fold of 10, outputting a mean squared error of each interpretation. The results showed that our error was incredibly low, with only a mean error of 0.1, which is not too bad.

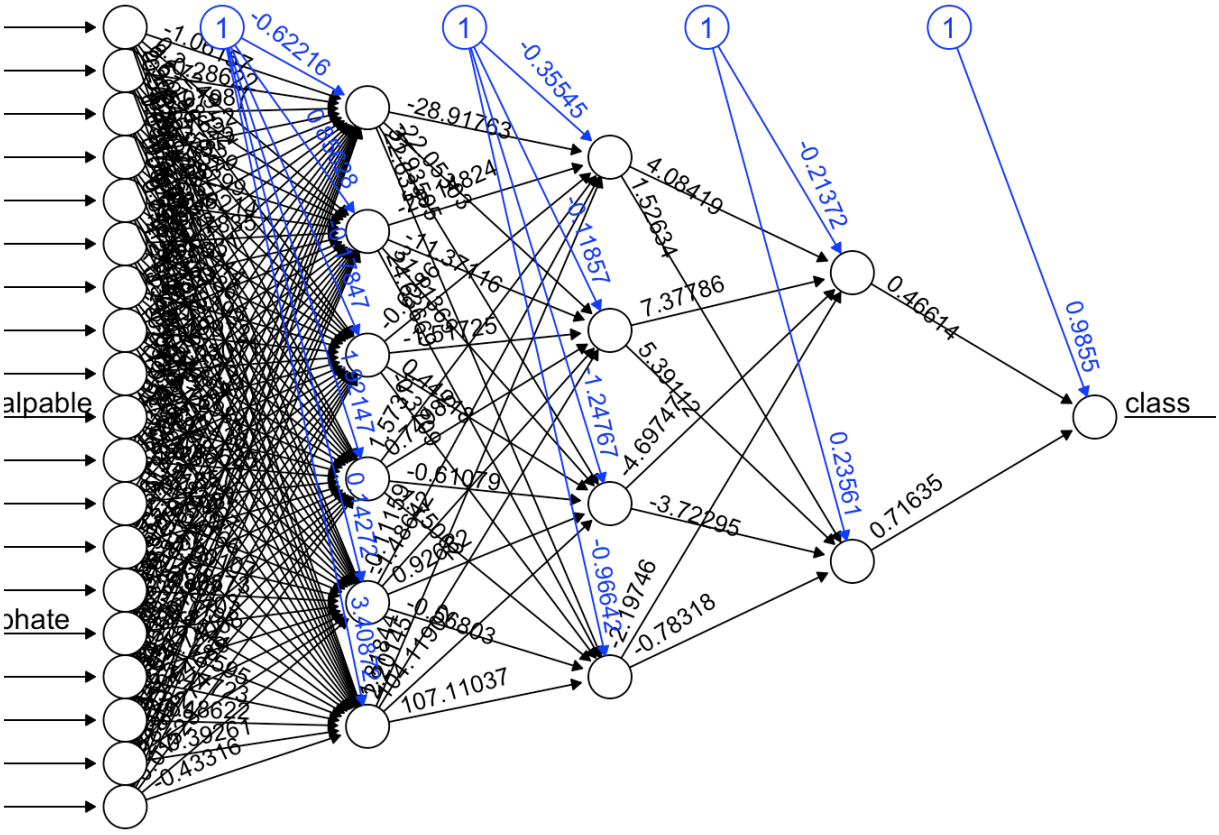
The visibility of a confusion matrix is however a bit tougher with a estimating the mean squared error within this function. It allows us to see that we have a great model, but does not allow us to specify the type of errors made due to how the for loop function is built.

Linear ANN using Replace Data Set

Because we are late in the analysis, I'll combine my code chunks, as you have probably gathered how the process works by this point. I will build the model, plot it, predict with it, then view the results; this time, all using the replaced dataset.

```
nnet2 <- neuralnet(formula = hep_formula, hidden = c(6,4,2), linear.output = T, data = hep_replace
_train)

plot(nnet2, rep = "best")
```




```
nnet2_predict <- neuralnet::compute(nnet2, hep_replace_test[2:20])
nnet2_predict$net.result <- round(nnet2_predict$net.result, 0)

results2 <- data.frame(actual = hep_replace_test$class, prediction = nnet2_predict$net.result)

cor(results2$prediction, results2$actual)
```

```
## [1] 0.1249638467
```

```
confusionMatrix(results2$prediction, results2$actual)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1  2  5
##          2  6 33
##
##                Accuracy : 0.7608696
##                  95% CI : (0.6123311, 0.8741386)
##     No Information Rate : 0.826087
##     P-Value [Acc > NIR] : 0.9089695
##
##                   Kappa : 0.1245675
##  Mcnemar's Test P-Value : 1.0000000
##
##             Sensitivity : 0.25000000
##             Specificity : 0.86842105
##          Pos Pred Value : 0.28571429
##          Neg Pred Value : 0.84615385
##              Prevalence : 0.17391304
##          Detection Rate : 0.04347826
##    Detection Prevalence : 0.15217391
##       Balanced Accuracy : 0.55921053
##
##        'Positive' Class : 1
##
```

We see a fairly solid initial model, but one that has not been an improvement upon what we have already built. The poor kappa suggests that some of our predictions might have happend by chance alone. Also our p-value is pretty rough considering what we have built thus far. Most of the error is happening in the true positive rate, or our ability to predict that someone has the disease, lets do a cross validation and see if we can help reduce the error occurring in this model.

Linear ANN with Cross Validation using Replace Data Set with Cross Validation

Build and Predict with Model

```
cv.error <- NULL

set.seed(450)

k <- 10

require(plyr)

pbar <- create_progress_bar('text')
pbar$init(k)
```

```
##
  |
  |                                                                      |   0%
```

```
for(i in 1:k){
    index <- sample(1:nrow(hep_replace_train),round(0.9*nrow(hep_replace_train)))

    train.cv <- hep_replace_train[index,]

    test.cv <- hep_replace_train[-index,]

    nn <- neuralnet(formula = hep_formula, hidden = c(6,4,2), linear.output = T, data = train.cv)

    pr.nn <- compute(nnet1, test.cv[2:20])

    pr.nn <- pr.nn$net.result*(max(hep_replace_train$class)-min(hep_replace_train$class))+min(hep_
replace_train$class)

    pr.nn <- round(pr.nn, 0)

    test.cv.r <- (test.cv$class)*(max(hep_replace_train$class)-min(hep_replace_train$class))+min(h
ep_replace_train$class)

    cv.error[i] <- sum((test.cv.r - pr.nn)^2)/nrow(test.cv)

    pbar$step()
}
```

```
##
  |
  |======                                                                |  10%
  |
  |=============                                                         |  20%
```

```
## Warning: algorithm did not converge in 1 of 1 repetition(s) within the
## stepmax
```

```
##
  |
  |===================                                                   |  30%
  |
  |==========================                                            |  40%
  |
  |================================                                      |  50%
  |
  |=======================================                               |  60%
  |
  |==============================================                        |  70%
  |
  |=====================================================                 |  80%
  |
  |============================================================          |  90%
  |
  |======================================================================| 100%
```

```
cv.error
```

```
##  [1] 0.36363636364 0.45454545455 0.36363636364 0.27272727273 0.18181818182
##  [6] 0.00000000000 0.27272727273 0.18181818182 0.09090909091 0.18181818182
```

```
mean(cv.error)
```

```
## [1] 0.2363636364
```

Interpret Results

Once again, the error is higher than expected with this last neural net model. If taken at a slanted comparison to the previous replace data neural net, the model has improved a little when considering the mean error, however the model still does not predict the test data as well as many of our other models. Its interesting to see the replacement of variables to take a tole on the accuracy, and I think this is especially so because we are attempting to predict a factor with many other binary factors, so the replacing of the mean is not entirely as helpful as if we were working primarily with continuous numeric data.

Conclusion

At the end of the day, I have been happy with our ability to predict disease considering that we had only 80 clean observations to work with. The artificial neural nets are an incredibly complex output that proved, on the whole to be more accurate, but the cross validation did in this case appear to have a smaller magnitude of affect on the neural nets vs. the support vector machines.

The cross validations will continue to add delta to a model as the model gets bigger, so to implement cross validation tuning within a functionalized and automated algorithm will more than likely better the model, especially as you move forward with time and more data. The implementation of the cross validation within a neural net proved to be a massive task for me, and I look forward to continuing to better wrap my head around the integration of these two complicated, yet powerful concepts.