Stats 101C Final Project - Random Forest (Jamie)

James Griffith - UID: 805458693

2023-12-06

Loading Data and Normalizing

```
### Stats 101C Final Project - Jamie's Portion ###

# Set seed
set.seed(123)
# Load Libraries
library(randomForest) # the randomForest package was used to perform the random forest analysis.
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
# Read in the pre-processed data.
x_pca <- read.csv(header = FALSE,"C:\\Users\\jamgr\\OneDrive\\Documents\\Stats 101C\\Datasets\\X_pca.csv")
y <- read.csv( "C:\\Users\\jamgr\\OneDrive\\Documents\\Stats 101C\\Datasets\\y.csv")
y[,5] <- as.factor(y[,5]) # Switch to encoding y as a factor so it will be processed as a classification problem by the randomForest function. This is required to have randomForest perform classification rather than regression (as far as I can tell after going through the documentation a few times)
# Normalize
x_pca_n <- as.data.frame(scale(x_pca)) # Normalize using scale() and then converting the Matrix output to a data frame.

## Split into training and testing data.
sampled_values <- sample(nrow(x_pca_n), nrow(x_pca_n)*.8) # Sample 40000 entries to put in training set (80/20 split)
train <- x_pca_n[sampled_values, ] # Create training data
test <- x_pca_n[-sampled_values, ] # Create testing data
train[,51] <- y[sampled_values, 5] # Append y to the training data (note I prefer this method for clarity since training[,51] is more understandable than y[sampled_values,5] in later code. You can also just use y[sampled_values,5] or y[-sampled_values,5], directly instead if you want to keep it separate.)
test[,51] <- y[-sampled_values, 5] # Create training data
```

Fit Random Forest to the training dataset

```
## Fitting Random Forest to the train dataset
classifier_RF = randomForest(x = train[,-51],
                             y = train[,51],
                             ntree = 50) # Number of trees, this was the highest value my device
could compute in a reasonable amount of time.
classifier_RF # I'm pretty sure this tells us our training error/Performance on the training se
t. (OOB error rate and confusion matrix for the training set)
```

```
##
## Call:
##  randomForest(x = train[, -51], y = train[, 51], ntree = 50)
##                Type of random forest: classification
##                      Number of trees: 50
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 20.68%
## Confusion matrix:
##       0     1 class.error
## 0 15771  4173   0.2092359
## 1  4100 15956   0.2044276
```

Use the model on the testing dataset

We obtain an error value of ~18%, the most important variables for prediction were columns 3,4,2,10, and 5. Increasing the number of trees slightly increases accuracy in exchange for a lot more computation. (ntree = 50 yields ~19% error, ntree = 500 yields ~18% error)

```
## Testing Data
# Predicting the Test set results
y_pred_rf = predict(classifier_RF, type="class", newdata = test)
# Confusion Matrix for test data
confusion_mtx_rf = table(test[,51], y_pred_rf)
confusion_mtx_rf
```

```
##    y_pred_rf
##        0    1
##   0 4083  973
##   1  889 4055
```

```
# Compute error rate for test data
error_rf <- mean(y_pred_rf != test[,51])
error_rf # Current output is 0.1794
```

```
## [1] 0.1862
```

```
## Additional Code to look at which variables were the most helpful/important
# Variable importance
importance(classifier_RF) # Get measures of each variable's importance.
```

```
##     MeanDecreaseGini
## V1          403.6942
## V2          886.2415
## V3         3564.3920
## V4         1393.5857
## V5          715.2859
## V6          530.7262
## V7          415.3102
## V8          523.6960
## V9          355.9740
## V10         720.7823
## V11         278.4715
## V12         293.4588
## V13         239.9739
## V14         251.3732
## V15         294.7662
## V16         311.0549
## V17         271.1208
## V18         243.1955
## V19         257.6573
## V20         236.5622
## V21         329.8190
## V22         254.0239
## V23         258.8989
## V24         236.7090
## V25         231.9803
## V26         325.7512
## V27         297.0049
## V28         298.8970
## V29         229.7995
## V30         225.9438
## V31         244.9739
## V32         413.4023
## V33         248.4806
## V34         315.8911
## V35         250.0078
## V36         241.2936
## V37         254.3892
## V38         231.5113
## V39         226.4784
## V40         230.9749
## V41         321.2520
## V42         256.6369
## V43         244.5929
## V44         237.6112
## V45         234.5448
## V46         247.9440
## V47         227.6257
## V48         221.1097
## V49         239.6148
## V50         234.7587
```

```
# Variable importance plot
varImpPlot(classifier_RF) # Plot those values
```