MSDS 5213: HW2

Olivia Samples

6/26/2020

Homework

For the homework, use glass.csv data.

Dataset description

The dataset description is available here http://archive.ics.uci.edu/ml/datasets/Glass+Identification

Your tasks

Download the dataset Transform the data if needed, it is preferable if you convert the class number into the corresponding class name. Make sure the ID column is deleted. Split the dataset into a training set and a testing set; use an 80-20 split (10 points) Use the train function to build a basic regression tree Find the best values for cp for the training dataset (10 points) Test the produced model on the test dataset and produce a confusion matrix (10 points) Use the cp from part 1 in a rpart function and create a fancy plot of the decision tree (10 point) Use the bagging function to build a bagged tree experiment with nbagg from 1 to 70 to see if you can get a better model (20 points) Test the best produced model on the test dataset and produce a confusion matrix (10 points) Use the train function to build a random forest tree Find the best values for the number of tree and the best values for the number of variable randomly picked for the split (10 points) Test the produced model on the test dataset and produce a confusion matrix (10 points) This assignment needs to be reported using RMarkDown (10 points)

```
rm(list=ls())
setwd("/Users/osamples/Documents/Lipscomb/MSDS 5213/Assignments/HW2")
library(rpart)
library(rattle)
```

```
## Loading required package: tibble

## Loading required package: bitops

## Rattle: A free graphical interface for data science with R.
## Version 5.4.0 Copyright (c) 2006-2020 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

```
library(caTools)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
glass = read.csv(file="glass.csv", head=FALSE, sep=",")
summary(glass)
```

```
##        V1               V2              V3              V4
##  Min.   :  1.00   Min.   :1.511   Min.   :10.73   Min.   :0.000
##  1st Qu.: 54.25   1st Qu.:1.517   1st Qu.:12.91   1st Qu.:2.115
##  Median :107.50   Median :1.518   Median :13.30   Median :3.480
##  Mean   :107.50   Mean   :1.518   Mean   :13.41   Mean   :2.685
##  3rd Qu.:160.75   3rd Qu.:1.519   3rd Qu.:13.82   3rd Qu.:3.600
##  Max.   :214.00   Max.   :1.534   Max.   :17.38   Max.   :4.490
##        V5              V6              V7               V8
##  Min.   :0.290   Min.   :69.81   Min.   :0.0000   Min.   : 5.430
##  1st Qu.:1.190   1st Qu.:72.28   1st Qu.:0.1225   1st Qu.: 8.240
##  Median :1.360   Median :72.79   Median :0.5550   Median : 8.600
##  Mean   :1.445   Mean   :72.65   Mean   :0.4971   Mean   : 8.957
##  3rd Qu.:1.630   3rd Qu.:73.09   3rd Qu.:0.6100   3rd Qu.: 9.172
##  Max.   :3.500   Max.   :75.41   Max.   :6.2100   Max.   :16.190
##        V9             V10                V11
##  Min.   :0.000   Min.   :0.00000   Min.   :1.00
##  1st Qu.:0.000   1st Qu.:0.00000   1st Qu.:1.00
##  Median :0.000   Median :0.00000   Median :2.00
##  Mean   :0.175   Mean   :0.05701   Mean   :2.78
##  3rd Qu.:0.000   3rd Qu.:0.10000   3rd Qu.:3.00
##  Max.   :3.150   Max.   :0.51000   Max.   :7.00
```

Create a function to update the column names.

```r
column_names = function(names, df){
i = 1
for (name in names){
        colnames(df)[i] = name
        i = i + 1
}
return(df)
}
```

Create a function that can update the class names. (Turning it off for now because the class_names are really long.)

```r
convert_class = function(cn, col){
        #col <- ifelse(col == 1, cn[1], ifelse(col == 2, cn[2], ifelse(col == 3, cn[3], ifelse(col ==4,
        return(factor(col))
}
```

Transform the Data

```r
headers = c( "ID", "RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba", "Fe", "Type")

class_names = c("building_windows_float_processed", "building_windows_non_float_processed",
```

```
 "vehicle_windows_float_processed", "vehicle_windows_non_float_processed (none in this database)", "con

mydata = column_names(headers, glass)[-c(1)]
mydata$Type = convert_class(class_names, mydata$Type)
```

Split the data into training and testing data using an 80-20 split

```
set.seed(101)
split <- sample.split(mydata$Type, SplitRatio = 0.8)
train <- subset(mydata, split==TRUE)
test <- subset(mydata, split==FALSE)
testny <- subset(mydata, select =-Type, split==FALSE)
```

Regression Tree

We can use rpart with the train method from the caret library to find the best values for cp

10-folds cross validation

```
fitControl <- trainControl(method = 'cv', number=10)
```

We will attempt to find the best complexity parameter

```
Grid <- expand.grid(cp=seq(0, 0.05, 0.005))
```

Now we will run the training to determine the optimimal cp value.

```
trained_tree <- train(Type ~ . , data = train, method = 'rpart',
trControl=fitControl, metric = 'Accuracy', maximize=TRUE, tuneGrid=Grid)
trained_tree
```

```
## CART
##
## 171 samples
##   9 predictor
##   6 classes: '1', '2', '3', '5', '6', '7'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 154, 154, 152, 153, 153, 155, ...
## Resampling results across tuning parameters:
##
##   cp     Accuracy   Kappa
##   0.000  0.6746053  0.5567838
##   0.005  0.6746053  0.5567838
##   0.010  0.6801608  0.5633654
##   0.015  0.6854240  0.5701755
##   0.020  0.6909795  0.5777704
##   0.025  0.6909795  0.5777704
##   0.030  0.6909795  0.5773664
##   0.035  0.6965695  0.5768496
##   0.040  0.6778195  0.5511293
##   0.045  0.6778195  0.5453454
##   0.050  0.6586528  0.5137771
##
## Accuracy was used to select the optimal model using the largest value.
```

```
## The final value used for the model was cp = 0.035.
```

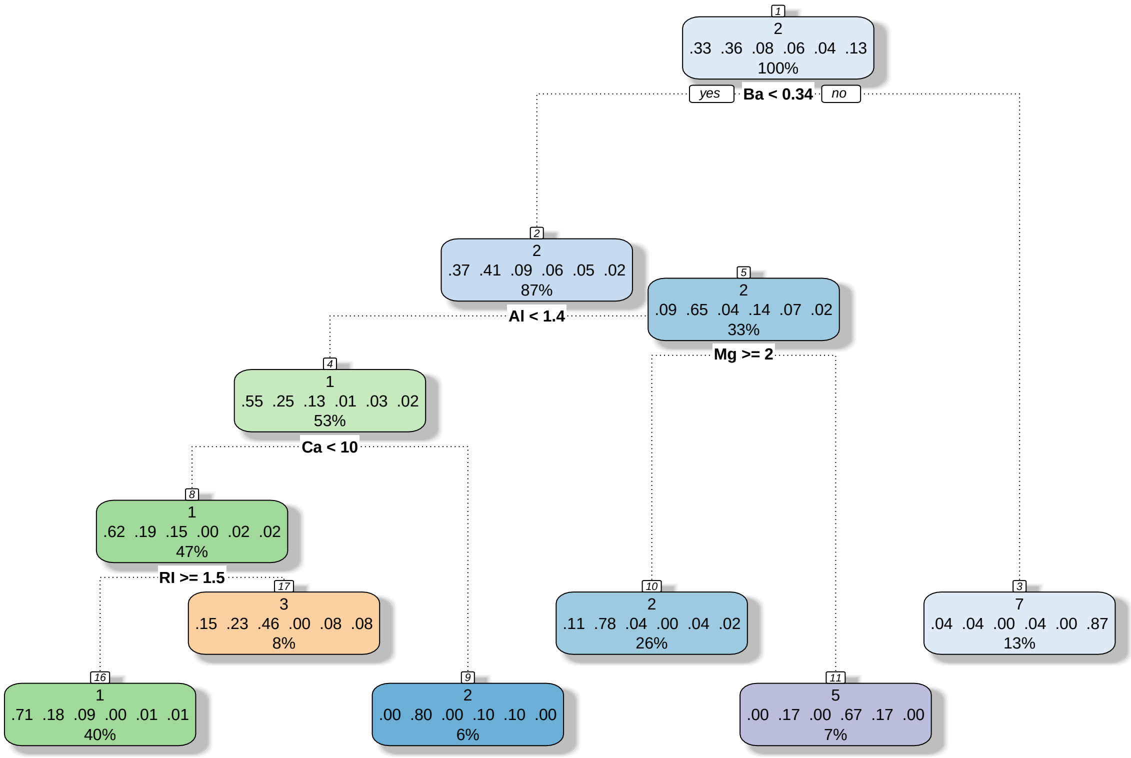
Now we will use this model with the optimal cp value to perform a prediction and produce a confusion matrix.

```
out = predict(trained_tree,testny, predictorstype="class")
confusionMatrix (out, test$Type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  5  6  7
##          1 12  5  0  0  0  0
##          2  1  9  2  0  1  0
##          3  1  1  1  0  0  0
##          5  0  0  0  3  1  0
##          6  0  0  0  0  0  0
##          7  0  0  0  0  0  6
##
## Overall Statistics
##
##                Accuracy : 0.7209
##                  95% CI : (0.5633, 0.8467)
##     No Information Rate : 0.3488
##     P-Value [Acc > NIR] : 7.344e-07
##
##                   Kappa : 0.6203
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 5 Class: 6 Class: 7
## Sensitivity            0.8571   0.6000  0.33333  1.00000  0.00000   1.0000
## Specificity            0.8276   0.8571  0.95000  0.97500  1.00000   1.0000
## Pos Pred Value         0.7059   0.6923  0.33333  0.75000      NaN   1.0000
## Neg Pred Value         0.9231   0.8000  0.95000  1.00000  0.95349   1.0000
## Prevalence             0.3256   0.3488  0.06977  0.06977  0.04651   0.1395
## Detection Rate         0.2791   0.2093  0.02326  0.06977  0.00000   0.1395
## Detection Prevalence   0.3953   0.3023  0.06977  0.09302  0.00000   0.1395
## Balanced Accuracy      0.8424   0.7286  0.64167  0.98750  0.50000   1.0000
```

Now we will use the model with the optimized cp to create a fancy plot of the decision tree.

```
temp <- rpart.control(xval=10, minbucket = 2,minsplit = 4,cp = trained_tree$bestTune)
dfit <- rpart(Type ~ . , data = train, control=temp)
fancyRpartPlot(dfit)
```



Rattle 2020−Jun−30 08:15:35 osamples

Bagged Tree

The ipred package contains functions for bagged tree.

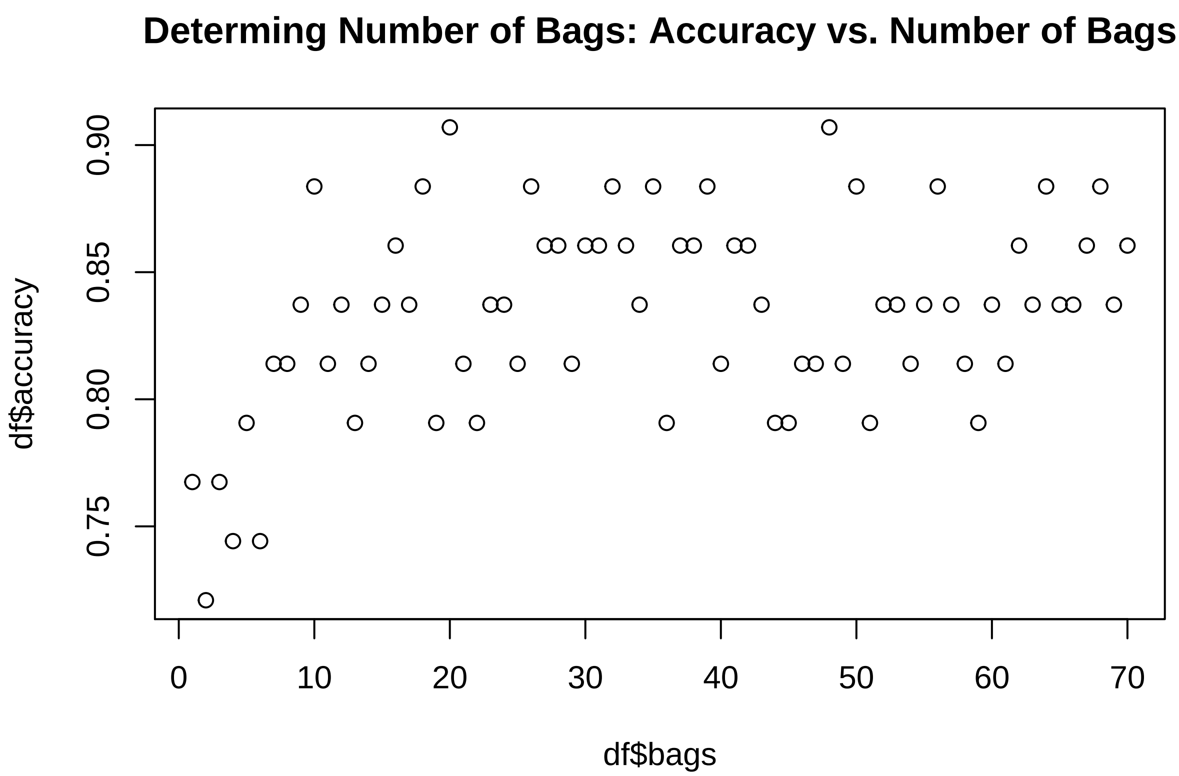
```
library(ipred)
```

Create a number of bag optimization function.

```
opt_bag = function(train, test, maxbag = 70){
        set.seed(1)
        df = data.frame("accuracy" = 1:maxbag, "bags" = 1:maxbag)
        list = list()
        for(i in (1:maxbag)){
                baggedTree = bagging(Type ~ .,data = train, nbagg = i)
                out2 = predict(baggedTree,subset(test, select=-Type))
                df[,1][i] = postResample(test$Type, out2)[1]
        }
        return(df)
        #return(order(df[,1],decreasing=T)[1])
}
```

Run the model on the optimized number of bags between 1 and 70. Plot the accuracy to determine when the number of bags begins to stop affecting the accuracy positively.

```
df = opt_bag(train, test)
plot(df$bags, df$accuracy, main = "Determing Number of Bags: Accuracy vs. Number of Bags")
```



Based on this plot, the Number of bags stops increasing accuracy around $n = 38$. For that reason, we will choose $nbagg = 38$. Test the best produced model on the test dataset and produce a confusion matrix.

```
baggedTree <- bagging(Type ~ .,data = train, nbagg = 38)

out2 = predict(baggedTree, subset(test, select=-Type), predictorstype='class')
confusionMatrix(out2, test$Type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  5  6  7
##          1  9  1  0  0  0  0
##          2  1 13  2  0  0  0
##          3  4  0  1  0  0  0
##          5  0  0  0  3  0  0
##          6  0  1  0  0  2  0
##          7  0  0  0  0  0  6
##
## Overall Statistics
##
##                Accuracy : 0.7907
##                  95% CI : (0.6396, 0.8996)
##     No Information Rate : 0.3488
##     P-Value [Acc > NIR] : 3.86e-09
##
##                   Kappa : 0.7242
##
##  Mcnemar's Test P-Value : NA
##
```

```
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 5 Class: 6 Class: 7
## Sensitivity            0.6429   0.8667  0.33333  1.00000  1.00000   1.0000
## Specificity            0.9655   0.8929  0.90000  1.00000  0.97561   1.0000
## Pos Pred Value         0.9000   0.8125  0.20000  1.00000  0.66667   1.0000
## Neg Pred Value         0.8485   0.9259  0.94737  1.00000  1.00000   1.0000
## Prevalence             0.3256   0.3488  0.06977  0.06977  0.04651   0.1395
## Detection Rate         0.2093   0.3023  0.02326  0.06977  0.04651   0.1395
## Detection Prevalence   0.2326   0.3721  0.11628  0.06977  0.06977   0.1395
## Balanced Accuracy      0.8042   0.8798  0.61667  1.00000  0.98780   1.0000
```

Random Forest Tree

To use random Forest we use the package of the same name that can be found here https://cran.r-project.org/web/packages/randomForest/randomForest.pdf

```
library(randomForest)
```

```
## randomForest 4.6-14

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

## The following object is masked from 'package:rattle':
##
##     importance
```

We will use the caret package to determine the optimal number of trees and the best values for the number of variables randomly picked for the split.

```
control <- trainControl(method="repeatedcv", number=10, repeats=3)

metric <- "Accuracy"
n <- round(sqrt(ncol(train)))
tunegrid <- expand.grid(.mtry=seq(1,n,1))
rf_default <- train(Type ~., data=train, method="rf",
metric=metric, tuneGrid=tunegrid, trControl=control)
print(rf_default)
```

```
## Random Forest
##
## 171 samples
##   9 predictor
##   6 classes: '1', '2', '3', '5', '6', '7'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 153, 154, 155, 155, 155, 153, ...
## Resampling results across tuning parameters:
##
```

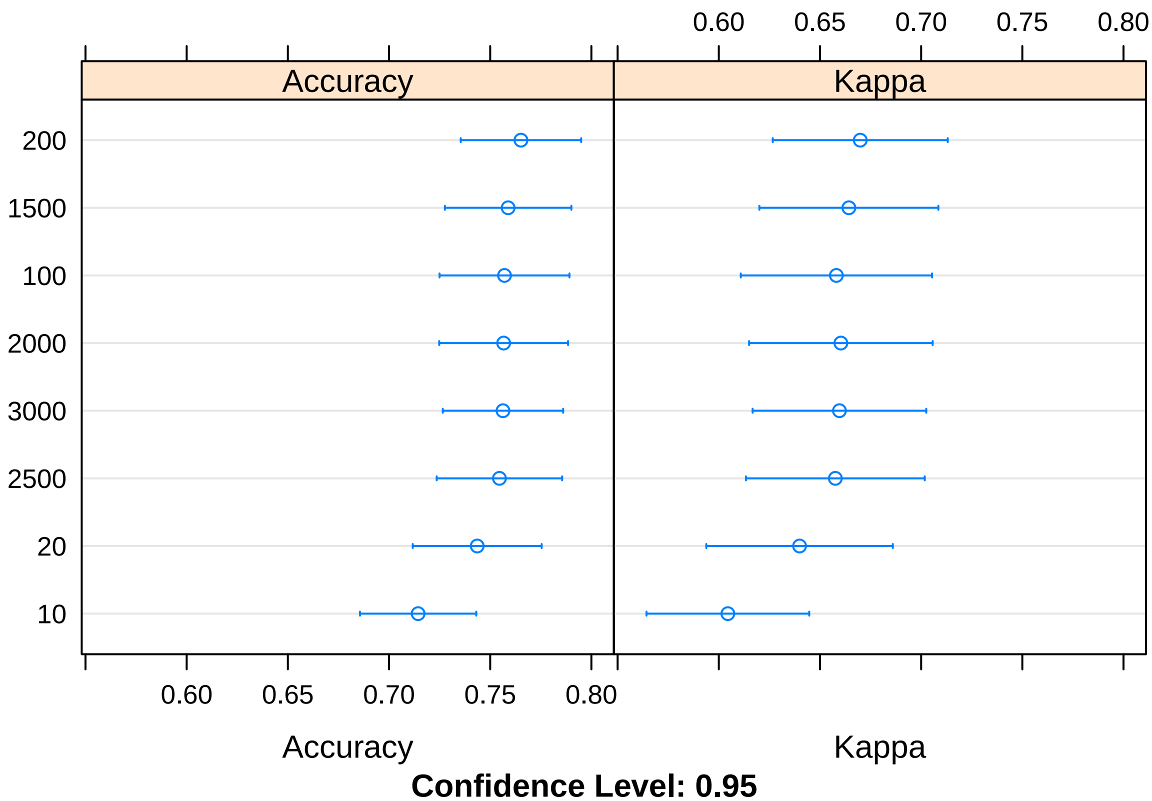
```
##   mtry  Accuracy   Kappa
##   1     0.7525993  0.6476280
##   2     0.7580732  0.6621537
##   3     0.7596076  0.6664879
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 3.
```

It appears that the optimal number of variables randomly picked for the split is 2.

```r
control <- trainControl(method="repeatedcv", number=10, repeats=3, search="grid")
tunegrid <- expand.grid(.mtry=seq(1,n,1))
modellist <- list()
for (ntree in c(10, 20, 100, 200, 1500, 2000, 2500, 3000)) {
    set.seed(1)
    fit <- train(Type~., data=train, method="rf", metric=metric, tuneGrid=tunegrid, trControl=control, n
    key <- toString(ntree)
    modellist[[key]] <- fit
}
# compare results
results <- resamples(modellist)
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: 10, 20, 100, 200, 1500, 2000, 2500, 3000
## Number of resamples: 30
##
## Accuracy
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## 10   0.5882353 0.6519608 0.7140523 0.7143831 0.7647059 0.8421053    0
## 20   0.6000000 0.6875000 0.7361111 0.7435819 0.7911765 0.9444444    0
## 100  0.5882353 0.7058824 0.7573529 0.7570656 0.8093750 0.9444444    0
## 200  0.5882353 0.7222222 0.7573529 0.7651952 0.8125000 0.9411765    0
## 1500 0.5789474 0.7099673 0.7647059 0.7588672 0.8125000 0.9411765    0
## 2000 0.5789474 0.6920956 0.7647059 0.7566450 0.8125000 0.9411765    0
## 2500 0.5789474 0.6920956 0.7647059 0.7545617 0.8125000 0.9411765    0
## 3000 0.5882353 0.6920956 0.7647059 0.7563161 0.8125000 0.9411765    0
##
## Kappa
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## 10   0.4079602 0.5348051 0.5999027 0.6044852 0.6725778 0.7896679    0
## 20   0.4039735 0.5447833 0.6382461 0.6399153 0.7088701 0.9234043    0
## 100  0.3897436 0.5845018 0.6532044 0.6581123 0.7361660 0.9234043    0
## 200  0.4166667 0.6021739 0.6706539 0.6698996 0.7373087 0.9174757    0
## 1500 0.4166667 0.6002347 0.6754158 0.6642844 0.7435897 0.9174757    0
## 2000 0.4166667 0.5739130 0.6754158 0.6603275 0.7417272 0.9174757    0
## 2500 0.4166667 0.5739130 0.6754158 0.6575929 0.7417272 0.9174757    0
## 3000 0.4166667 0.5739130 0.6754158 0.6596225 0.7417272 0.9174757    0
```

```r
dotplot(results)
```



It appears that the optimal number of trees is 2000, so we will use this number. Hence, we will build a random forest model with mtry = 2 and ntree = 2000.

```
rfModel <- randomForest(Type ~ . , data = train,ntree=20,mtry=2)
```

Now we will do the actual prediction and produce a confusion matrix.

```
out3 = predict(rfModel,testny, predictorstype = 'class')
confusionMatrix(out3, test$Type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  5  6  7
##          1 10  1  0  0  0  0
##          2  1 13  1  1  0  0
##          3  3  0  2  0  0  0
##          5  0  0  0  2  0  0
##          6  0  1  0  0  2  0
##          7  0  0  0  0  0  6
##
## Overall Statistics
##
##                Accuracy : 0.814
##                  95% CI : (0.666, 0.9161)
##     No Information Rate : 0.3488
##     P-Value [Acc > NIR] : 5.208e-10
##
##                   Kappa : 0.7529
##
```

```
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 5 Class: 6 Class: 7
## Sensitivity            0.7143   0.8667  0.66667  0.66667  1.00000   1.0000
## Specificity            0.9655   0.8929  0.92500  1.00000  0.97561   1.0000
## Pos Pred Value         0.9091   0.8125  0.40000  1.00000  0.66667   1.0000
## Neg Pred Value         0.8750   0.9259  0.97368  0.97561  1.00000   1.0000
## Prevalence             0.3256   0.3488  0.06977  0.06977  0.04651   0.1395
## Detection Rate         0.2326   0.3023  0.04651  0.04651  0.04651   0.1395
## Detection Prevalence   0.2558   0.3721  0.11628  0.04651  0.06977   0.1395
## Balanced Accuracy      0.8399   0.8798  0.79583  0.83333  0.98780   1.0000
```