Ensemble Techniques: Rice Classification - 10/23

Code

Rice Seed Classification with Ensemble Techniques

On this R markdown file, we'll be investigating a Rice seed classification dataset found on Kaggle. (https://www.kaggle.com/datasets/mssmartypants/rice-type-classification)

(Link for pdf viewers: https://www.kaggle.com/datasets/mssmartypants/rice-type-classification (https://www.kaggle.com/datasets/mssmartypants/rice-type-classification))

The dataset has two classes to categorize. A "0" indicates a Gonen rice seed (commonly found and produced in Northwest Turkey) and "1" for jasmine rice. This is tied to the class attribute of the dataset.

This dataset was explored and elaborated on in the kernelClassfication.rmd section of the project. So for the sake of brevity we will forgo this step here and continue with classification via ensemble techniques.

Hide

```
df0 <- read.csv("riceClassification.csv", header = TRUE)
df <- df0[,-1] # remove id section of rice grain
df$Class <- as.factor(df$Class) #factorize class so r doesn't treat it as a numeric v
alue
head(df)
```

A… <int>	MajorAxisLength <dbl>	MinorAxisLength <dbl>	Eccentricity <dbl>	ConvexArea <int>	EquivDiameter <dbl>
1 4537	92.22932	64.01277	0.7199162	4677	76.00452
2 2872	74.69188	51.40045	0.7255527	3015	60.47102
3 3048	76.29316	52.04349	0.7312109	3132	62.29634
4 3073	77.03363	51.92849	0.7386387	3157	62.55130
5 3693	85.12478	56.37402	0.7492816	3802	68.57167
6 2990	77.41707	50.95434	0.7528609	3080	61.70078

6 rows | 1-7 of 11 columns

Hide

```
#test/train split
set.seed(1234)
i <- sample(1 : nrow(df), round(nrow(df) * 0.8), replace = FALSE)
train <- df[i,]
test <- df[-i,]
```

Decision Tree Baseline

Hide

```
head(train)
```

	A...	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexAr...	EquivDiameter
	<int>	<dbl>	<dbl>	<dbl>	<int>	<dbl>
7452	4785	138.3683	45.00288	0.9456315	4932	78.05416
8016	6238	158.8883	51.05029	0.9469786	6406	89.12053
7162	6102	155.7152	50.98259	0.9448826	6277	88.14368
8086	6661	164.1947	52.67739	0.9471392	6850	92.09261
9196	6414	163.3800	50.84921	0.9503337	6656	90.36901
623	3956	113.6621	45.04988	0.9180999	4019	70.97137

6 rows | 1-7 of 11 columns

Hide

```
tail(train)
```

	A...	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexAr...	EquivDiameter
	<int>	<dbl>	<dbl>	<dbl>	<int>	<dbl>
18034	7922	159.8574	64.27246	0.9156128	8232	100.43208
7357	7193	168.6588	54.96993	0.9453960	7399	95.69959
14711	8088	152.1012	68.40084	0.8931765	8247	101.47887
2820	5771	144.8662	51.87860	0.9336779	5928	85.71969
17476	8741	163.6827	69.34900	0.9058121	9090	105.49591
14045	8748	157.7907	71.67771	0.8908702	9043	105.53814

6 rows | 1-7 of 11 columns

Hide

```
NA
```

Hide

```
library(tree)
library(rpart)


tree_rice <- rpart(train$Class~., data=train, method="class")
tree_rice
```

```
n= 14548

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 14548 6571 1 (0.45167721 0.54832279)
  2) MinorAxisLength>=58.57892 6557   97 0 (0.98520665 0.01479335) *
  3) MinorAxisLength< 58.57892 7991  111 1 (0.01389063 0.98610937) *
```

Hide

```
summary(tree_rice)
```

```
Call:
rpart(formula = train$Class ~ ., data = train, method = "class")
  n= 14548

         CP nsplit  rel error     xerror        xstd
1 0.9683458      0 1.00000000 1.00000000 0.009134871
2 0.0100000      1 0.03165424 0.03195861 0.002189378

Variable importance
MinorAxisLength    AspectRation    Eccentricity       Roundness            Area   Equ
ivDiameter
             18              17              17              16              16
16

Node number 1: 14548 observations,    complexity param=0.9683458
  predicted class=1  expected loss=0.4516772  P(node) =1
    class counts:  6571  7977
   probabilities: 0.452 0.548
  left son=2 (6557 obs) right son=3 (7991 obs)
  Primary splits:
      MinorAxisLength < 58.57892  to the right, improve=6796.012, (0 missing)
      Eccentricity    < 0.9151628 to the left,  improve=6168.608, (0 missing)
      AspectRation    < 2.48087   to the left,  improve=6168.608, (0 missing)
      Roundness       < 0.7171025 to the right, improve=5878.564, (0 missing)
      Area            < 7136.5    to the right, improve=5247.883, (0 missing)
  Surrogate splits:
      Eccentricity  < 0.9140821 to the left,  agree=0.962, adj=0.916, (0 split)
      AspectRation  < 2.465914  to the left,  agree=0.962, adj=0.916, (0 split)
      Roundness     < 0.7171025 to the right, agree=0.949, adj=0.886, (0 split)
      Area          < 7075.5    to the right, agree=0.931, adj=0.846, (0 split)
      EquivDiameter < 94.91473  to the right, agree=0.931, adj=0.846, (0 split)

Node number 2: 6557 observations
  predicted class=0  expected loss=0.01479335  P(node) =0.4507149
    class counts:  6460    97
   probabilities: 0.985 0.015

Node number 3: 7991 observations
  predicted class=1  expected loss=0.01389063  P(node) =0.5492851
    class counts:   111  7880
   probabilities: 0.014 0.986
```

Hide

```
plot(tree_rice, uniform = TRUE)
text(tree_rice, use.n = TRUE, all = TRUE)
```

I'm not sure why the decision tree isn't plotting, especially when we see a very different tree_rice summary that looks to be pretty sophisticated. Strange. Let's try a prediction next.

Hide

```
tree_pred <- predict(tree_rice, newdata=test, type="class")
summary(tree_pred)
```

```
   0    1
1634 2003
```

Hide

```
table(tree_pred, test$Class)
```

```
tree_pred    0    1
        0 1601   33
        1   28 1975
```

Hide

```
mean(tree_pred==test$Class)
```

```
[1] 0.9832279
```

Regardless of our wonky printing decision tree plot, we have some great accuracy values for a standard decision tree.

Random Forest

Hide

```
library(randomForest)
```

```
randomForest 4.7-1.1
Type rfNews() to see new features/changes/bug fixes.
```

Hide

```
start.time <- Sys.time()

set.seed(1234)
rf <- randomForest(train$Class ~ ., data=train, importance=TRUE, proximity = TRUE)

end.time <- Sys.time()
time.taken <- end.time - start.time
print(time.taken)
```

```
Time difference of 4.667877 mins
```

Boosting via adabag

Hide

```
library(adabag)
adbag1 <- boosting(Class~. , data = train, boos = FALSE, mfinal = 100, coeflearn = 'B
reiman')
summary(adbag1)
```

```
           Length Class   Mode
formula         3  formula call
trees         100  -none-  list
weights       100  -none-  numeric
votes       29096  -none-  numeric
prob        29096  -none-  numeric
class       14548  -none-  character
importance     10  -none-  numeric
terms           3  terms   call
call            6  -none-  call
```

Predict on adabag

Hide

```
pred <- predict(adbag1, newdata=test, type = "response")
acc_adabad <- mean(pred$Class==test$Class)
#mcc_adabag <- mcc(factor(pred$Class), test$Class)
print(paste("accuracy=", acc_adabad))
```

```
[1] "accuracy= NaN"
```

Hide

```
#print(paste("mcc=", mcc_adabag))
```

XGBoost

Hide

```
require(xgboost)
start.time3 <- Sys.time()

data(agaricus.train, package='xgboost')
data(agaricus.test, package='xgboost')
trainxg <- agaricus.train
testxg <- agaricus.test

model <- xgboost(data=trainxg$data, label=trainxg$label, nrounds=8, objective='binar
y:logistic')
```

```
[1] train-logloss:0.439409
[2] train-logloss:0.299260
[3] train-logloss:0.209937
[4] train-logloss:0.150151
[5] train-logloss:0.108673
[6] train-logloss:0.079348
[7] train-logloss:0.058385
[8] train-logloss:0.043147
```

Hide

```
pred <- predict(model, testxg$data)
pred <- ifelse(pred> 0.5,1,0)
table(pred, testxg$label)
```

```
pred   0   1
   0 835   0
   1   0 776
```

Hide

```
cv.res <- xgb.cv(data=trainxg$data, label=trainxg$label, nfold=5, nrounds=2, objectiv
e='binary:logistic')
```

```
[1] train-logloss:0.439974+0.000337 test-logloss:0.440172+0.000249
[2] train-logloss:0.299852+0.000304 test-logloss:0.300093+0.000387
```

Hide

```
end.time3 <- Sys.time()
time.taken3 <- end.time3 - start.time3
print(time.taken3)
```

```
Time difference of 0.7746556 secs
```

Final Remarks

With all of our times in, we can see that randomforest and adaboost take AGES to run, with a questionable tradeoff. They don't seem to have much better accuracy than that of a standard decision tree, at least not with the accuracy-time tradeoff I'm willing to consider. One surprise was how lightning quick xgboost was! It took less than a second to run, which was stunning. The book did mention that we needed to alter xgboosts input to get lightning fast outputs and we did so and got something really special.