Ensemble Methods

Zuhayr Ali, Simon Kim

2022-10-22

Classification on 10 thousand diamonds from ggplot2's "diamonds"

This notebook will be exploring the "diamonds" dataset from ggplot2 with the goal of predicting the cut quality based on other statistics.

Cleaning data

We start by loading the dataset and displaying the columns. There is readily available documentation explaining each column. Like in the SVM classification notebook, we will determine cut quality from carat, depth, table, and the x-y-z dimensions.

```
library(mltools)
library(ggplot2)

data("diamonds")
df=diamonds
colnames(df)
```

```
##  [1] "carat"   "cut"     "color"   "clarity" "depth"   "table"   "price"
##  [8] "x"       "y"       "z"
```

```
str(df)
```

```
## tibble [53,940 x 10] (S3: tbl_df/tbl/data.frame)
##  $ carat  : num [1:53940] 0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
##  $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
##  $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
##  $ depth  : num [1:53940] 61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
##  $ table  : num [1:53940] 55 61 65 58 58 57 57 55 61 61 ...
##  $ price  : int [1:53940] 326 326 327 334 335 336 336 337 337 338 ...
##  $ x      : num [1:53940] 3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ y      : num [1:53940] 3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ z      : num [1:53940] 2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

The amount of rows is too much for the algorithms to run in time at over 50 thousand rows, so the data has been pruned to the first 10 thousand.

```
df <- df[1:10000,]
str(df)
```

```
## tibble [10,000 x 10] (S3: tbl_df/tbl/data.frame)
##  $ carat  : num [1:10000] 0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
##  $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
```

```
##  $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
##  $ depth  : num [1:10000] 61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
##  $ table  : num [1:10000] 55 61 65 58 58 57 57 55 61 61 ...
##  $ price  : int [1:10000] 326 326 327 334 335 336 336 337 337 338 ...
##  $ x      : num [1:10000] 3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ y      : num [1:10000] 3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ z      : num [1:10000] 2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

Again like in the SVM classification notebook, we will group the 5 classes of cut quality into 2 superclasses to make this dataset work for binary classification.

```
df$cut_status<-NA
df$cut_status[df$cut=='Fair'|df$cut=='Good'|df$cut=='Very Good']=0
df$cut_status[df$cut=='Premium'|df$cut=='Ideal']=1
df$cut_status=as.factor(df$cut_status)
```

We split the dataset into 80/20 train and test subsets.

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

Data exploration

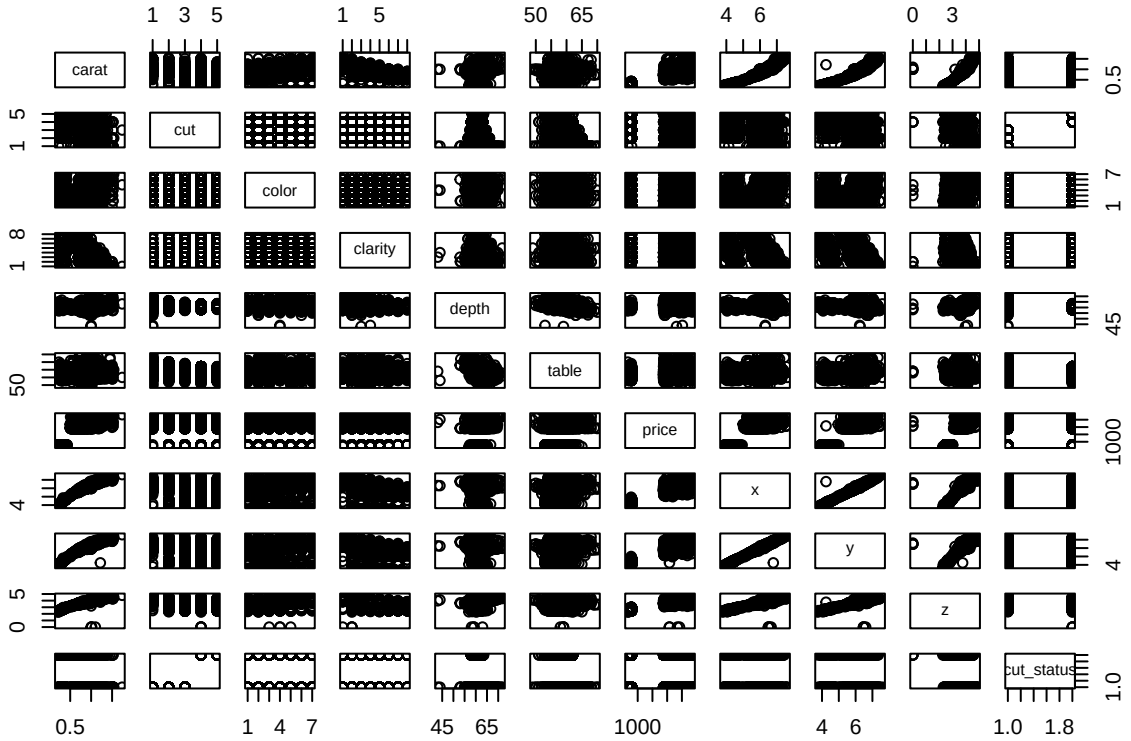
A statistical breakdown of each column...

```
summary(train)
```

```
##      carat               cut       color       clarity         depth
##  Min.   :0.2000   Fair     : 415   D:1066   SI2    :2522   Min.   :43.00
##  1st Qu.:0.7200   Good     :1063   E:1434   SI1    :2158   1st Qu.:61.00
##  Median :0.9000   Very Good:2029   F:1476   VS2    :1322   Median :61.90
##  Mean   :0.8474   Premium  :1983   G:1380   VS1    : 958   Mean   :61.87
##  3rd Qu.:1.0100   Ideal    :2510   H:1308   VVS2   : 421   3rd Qu.:62.80
##  Max.   :1.7400                    I: 855   VVS1   : 307   Max.   :71.80
##                                    J: 481   (Other): 312
##      table           price            x               y               z
##  Min.   :49.00   Min.   : 326   Min.   :3.790   Min.   :3.750   Min.   :0.000
##  1st Qu.:56.00   1st Qu.:3032   1st Qu.:5.780   1st Qu.:5.790   1st Qu.:3.560
##  Median :58.00   Median :3629   Median :6.130   Median :6.140   Median :3.820
##  Mean   :57.83   Mean   :3415   Mean   :5.991   Mean   :5.991   Mean   :3.705
##  3rd Qu.:59.00   3rd Qu.:4206   3rd Qu.:6.400   3rd Qu.:6.390   3rd Qu.:3.970
##  Max.   :70.00   Max.   :4704   Max.   :7.620   Max.   :7.590   Max.   :4.870
##
##  cut_status
##  0:3507
##  1:4493
##
##
##
##
##
```

... followed by a correlation matrix of all columns.

```
pairs(train)
```



We will now use 3 ensemble methods for classification and compare them to decision tree classification as a baseline.

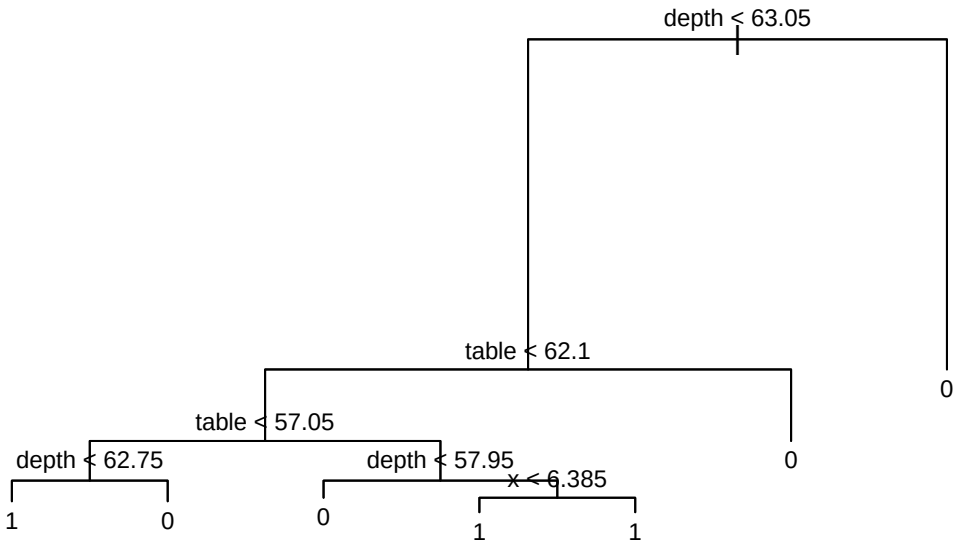
Decision tree

Training and tree plot

```
library(tree)

dtree_start <- Sys.time()
dtree <- tree(cut_status~carat+depth+table+x+y+z, data=train)
dtree_end <- Sys.time()

plot(dtree)
text(dtree, cex=0.75, pretty=0)
```




```
summary(dtree)
```

```
##
## Classification tree:
## tree(formula = cut_status ~ carat + depth + table + x + y + z,
##     data = train)
## Variables actually used in tree construction:
## [1] "depth" "table" "x"
## Number of terminal nodes:  7
## Residual mean deviance:  0.8345 = 6670 / 7993
## Misclassification error rate: 0.2065 = 1652 / 8000
```

```
print(paste('time:', dtree_end - dtree_start, 'seconds'))
```

```
## [1] "time: 0.030066967010498 seconds"
```

Testing and evaluation

```
dtree_pred <- predict(dtree, newdata=test, type="class")

table(dtree_pred, test$cut_status)
```

```
##
## dtree_pred    0    1
##          0  449   39
##          1  383 1129
```

```
acc_dtree <- mean(dtree_pred==test$cut_status)
mcc_dtree <- mcc(factor(dtree_pred), test$cut_status)

print(paste('accuracy:', acc_dtree))
```

```
## [1] "accuracy: 0.789"
```

```
print(paste("mcc=", mcc_dtree))
```

```
## [1] "mcc= 0.581007967510787"
```

Random forest

Training

```
library(randomForest)
```

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

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

```
##
## Attaching package: 'randomForest'
```

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

```
rf_start <- Sys.time()
rf <- randomForest(cut_status~carat+depth+table+x+y+z, data=train, importance=TRUE)
rf_end <- Sys.time()

rf
```

```
##
## Call:
##  randomForest(formula = cut_status ~ carat + depth + table + x +      y + z, data = train, importanc
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 13.43%
## Confusion matrix:
##      0    1 class.error
## 0 2796  711  0.20273738
## 1  363 4130  0.08079234
```

```
print(paste('time:', rf_end - rf_start, 'seconds'))
```

```
## [1] "time: 3.99489188194275 seconds"
```

Testing and evaluation

```
rf_pred <- predict(rf, newdata=test, type="response")

acc_rf <- mean(rf_pred==test$cut_status)
mcc_rf <- mcc(factor(rf_pred), test$cut_status)
```

```
print(paste("accuracy:", acc_rf))
```

```
## [1] "accuracy: 0.8695"
```

```
print(paste("mcc=", mcc_rf))
```

```
## [1] "mcc= 0.730050419790422"
```

XGBoost

Testing

```
library(xgboost)

train_label <- ifelse(train$cut_status==1, 1, 0)
train_matrix <- data.matrix(train[, -11])

test_label <- ifelse(test$cut_status==1, 1, 0)
test_matrix <- data.matrix(test[, -11])

xg_start <- Sys.time()
xg <- xgboost(data=train_matrix, label=train_label, nrounds=5, objective='binary:logistic')
```

```
## [1]  train-logloss:0.437700
## [2]  train-logloss:0.296542
## [3]  train-logloss:0.207577
## [4]  train-logloss:0.148050
## [5]  train-logloss:0.106852
```

```
xg_end <- Sys.time()

print(paste('time:', xg_end - xg_start, 'seconds'))
```

```
## [1] "time: 0.0141658782958984 seconds"
```

Testing and evaluation

```
xg_probs <- predict(xg, test_matrix)
xg_pred <- ifelse(xg_probs>0.5, 1, 0)

acc_xg <- mean(xg_pred==test_label)
mcc_xg <- mcc(factor(xg_pred), test$cut_status)

print(paste("accuracy:", acc_xg))
```

```
## [1] "accuracy: 1"
```

```
print(paste("mcc=", mcc_xg))
```

```
## [1] "mcc= 1"
```

Light GBM

Data prep

```
library(caret)
```

```
## Loading required package: lattice
```

```
library(lightgbm)
```

```
## Loading required package: R6
```

```
##
## Attaching package: 'lightgbm'
```

```
## The following objects are masked from 'package:xgboost':
##
##     getinfo, setinfo, slice
```

```
df$cut_status <- as.numeric(df$cut_status)-1

train = as.matrix(df[i, ])
test = as.matrix(df[-i, ])

train_x = train[, -11]
train_y = train[, 11]

test_x = test[, -11]
test_y = test[, 11]

dtrain = lgb.Dataset(train_x, label = train_y)
```

```
## Warning in storage.mode(data) <- "double": NAs introduced by coercion
```

```
dtest = lgb.Dataset.create.valid(dtrain, data = test_x, label = test_y)
```

```
## Warning in storage.mode(data) <- "double": NAs introduced by coercion
```

```
lgbm_params = list(objective = "binary", num_leaves = 4L, learning_rate = 1.0)
```

Training

```
lgbm_start <- Sys.time()
lgbm <- lgb.train(lgbm_params, dtrain, nrounds = 10L, verbose = -1L)
lgbm_end <- Sys.time()

print(paste('time:', lgbm_end - lgbm_start, 'seconds'))
```

```
## [1] "time: 0.0469250679016113 seconds"
```

Testing and evaluation

```
lgbm_probs = predict(lgbm, test_x, reshape=T)
```

```
## Warning in storage.mode(data) <- "double": NAs introduced by coercion
```

```
lgbm_pred = ifelse(lgbm_probs>0.5, 1, 0)

confusionMatrix(as.factor(test_y), as.factor(lgbm_pred))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0  585  247
##          1   95 1073
##
##                Accuracy : 0.829
##                  95% CI : (0.8118, 0.8453)
##     No Information Rate : 0.66
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6386
##
##  Mcnemar's Test P-Value : 3.21e-16
##
##             Sensitivity : 0.8603
##             Specificity : 0.8129
##          Pos Pred Value : 0.7031
##          Neg Pred Value : 0.9187
##              Prevalence : 0.3400
##          Detection Rate : 0.2925
##    Detection Prevalence : 0.4160
##       Balanced Accuracy : 0.8366
##
##        'Positive' Class : 0
##
```

```
mcc_lgbm <- mcc(factor(lgbm_pred), factor(test_y))
print(paste("mcc=", mcc_lgbm))
```

```
## [1] "mcc= 0.646971224984015"
```

Comparison

While the decision tree does a capable job of predicting cut quality, it produces the worst results of all methods and is the 2nd slowest to train at over 34 milliseconds. The accuracy is only 78.9% and the correlation is less than 0.6. The first ensemble method, random forests, is a marked improvement in results but at the cost of speed. The accuracy is 86.95% and the correlation is slightly above 0.73, but it takes nearly 3 and a half seconds to train. the second ensemble method, XGBoost performs the best of the methods in this notebook without any serious competition. For a slight improvement in training speed over the decision tree at over 25 milliseconds, both the accuracy and correlation are perfect at 100% and 1 respectively. The final ensemble method, light GBM, produces the worst results of the ensemble methods but is still better than decision trees, and ties with XGBoost for the best speed. The accuracy is 82.9% and the correlation is less than 0.65.