CodeNotebook

Scott McCoy - U80152879

2/27/2021

Mobile Price Classification

Models

- Linear Models
    - Nnet Multiclass Linear Model (Baseline)
    - Binary Logistic Regression
    - Multiple Binary Logistic Regressions
    - Layered Binary Logistic Regressions
- Tree-based Models
    - Decision Tree
    - Random Forest
        - Multiple Binary Random Forests
        - Multiclass Random Forest

Linear Models

Neural Net Multiclass Linear Model

There are a number of methods that can be applied to a multiclass classification problem.

Of the various models we applied to our problem, the best performing was a log-linear neural network model.

We used this as a baseline to inform our strategy and compare results to the methods discusses in the course.

```
defaultW <- getOption("warn")
options(warn = -1)
options(warn = defaultW)

set.seed(430)
library(data.table)
library(nnet)
mo <- fread('~/R/mobile/train.csv')
mo_obs <- nrow(mo)
mo_idx <- sample(mo_obs, size = trunc(0.70 * mo_obs))
mo_trn <- mo[mo_idx, ]
mo_test <- mo[-mo_idx, ]

Y_train <- mo_trn[,price_range]
Y_test <- mo_test[,price_range]

model_multi <- multinom(price_range ~ ., data = mo_trn) # instantiating model
```

```
## # weights:  88 (63 variable)
## initial  value 1940.812106
## iter  10 value 1539.449569
## iter  20 value 1372.517934
## iter  30 value 1328.431365
## iter  40 value 1203.483614
## iter  50 value 826.577171
## iter  60 value 446.763231
## iter  70 value 59.572210
## iter  80 value 31.622093
## iter  90 value 21.277743
## iter 100 value 15.526852
## final  value 15.526852
## stopped after 100 iterations
```

```
# train predictions
Y_train_hat_df <- predict(model_multi, newdata = mo_trn, type = "prob")

Y_train_hat <- data.table(colnames(Y_train_hat_df)[max.col(Y_train_hat_df,ties.method="first")])
Y_train_hat <- transform(Y_train_hat, V1 = as.numeric(V1))

train_accuracy <- mean(Y_train == Y_train_hat)

#test predictions
Y_test_hat_df <- predict(model_multi, newdata = mo_test, type = "prob")

Y_test_hat <- data.table(colnames(Y_test_hat_df)[max.col(Y_test_hat_df,ties.method="first")])
Y_test_hat <- transform(Y_test_hat, V1 = as.numeric(V1))[,V1]

test_accuracy <- mean(Y_test == Y_test_hat)

# Evaluation:
cm <- table(observed=Y_test, predicted=Y_test_hat) # confusion matrix

diag = diag(cm) # number of correctly classified instances per class
rowsums = apply(cm, 1, sum) # number of instances per class
colsums = apply(cm, 2, sum) # number of predictions per class


precision = diag / colsums
recall = diag / rowsums
f1 = 2 * precision * recall / (precision + recall)

scores <- data.frame(precision, recall, f1)

train_accuracy
```

```
## [1] 0.9971429
```

```
test_accuracy
```

```
## [1] 0.9716667
```

```
cm
```

```
##         predicted
## observed   0   1   2   3
##        0 147   1   0   0
##        1   2 147   2   0
##        2   0   5 147   6
##        3   0   0   1 142
```

```
scores
```

```
##   precision    recall        f1
## 0 0.9865772 0.9932432 0.9898990
## 1 0.9607843 0.9735099 0.9671053
## 2 0.9800000 0.9303797 0.9545455
## 3 0.9594595 0.9930070 0.9759450
```

We have a test accuracy of 97% for this model. This high accuracy of this linear model indicates that there is likely a strong linear relationship between the predictors and price_range.

Binary Logistic Regression

Because we have more than 2 classes of predictors, we cannot simply predict between all 4 with a simple logistic regression.

Our first approach was to predict between the the lowest price range (0,1) and the highest price (2,3) with a binomial model.

```
defaultW <- getOption("warn")
options(warn = -1)


library(caTools)
library(caret)
mo_p <- fread('~/R/mobile/processed_train.csv')
m_binary <- mo_p[, !c("price_range", "p0", "p1", "p2", "p3")]
sampleSplit <- sample.split(Y=m_binary$price_binary, SplitRatio=0.7)
trainSet <- subset(x=m_binary, sampleSplit==TRUE)
testSet <- subset(x=m_binary, sampleSplit==FALSE)

log_model_binary <- glm(price_binary ~ ., family=binomial(link='logit'), data=trainSet)

probabs <- predict(log_model_binary, testSet[,!c("price_binary")],type='response')
preds <- ifelse(probabs > 0.5, 1, 0)

test_accuracy <- mean(testSet$price_binary == preds)

cm <- table(observed=testSet$price_binary, predicted=preds)



test_accuracy
```

```
## [1] 0.995
```

```
cm
```

```
##         predicted
## observed   0   1
##        0 300   0
##        1   3 297
```

This model does an excellent job differentiating between low prices and high prices.

Multiple Binary Logistic Regression

Our next step was to run multiple binomial models, one for each class of price range.

We can select between all 4 classes by ultimately predicting the class with the highest probability between all 4 models.

```
mo_obs <- nrow(mo_p)
mo_idx <- sample(mo_obs, size = trunc(0.70 * mo_obs))
mo_trn <- mo_p[mo_idx, ]
mo_test <- mo_p[-mo_idx, ]

# 4 separate train and test sets
X_train <- mo_trn[,1:20]
Y_train <- mo_trn[,price_range]
Y_train0 <- mo_trn[,p0]
Y_train1 <- mo_trn[,p1]
Y_train2 <- mo_trn[,p2]
Y_train3 <- mo_trn[,p3]
X_train0 <- data.table(X_train)
X_train0[,p0 := Y_train0]
X_train1 <- data.table(X_train)
X_train1[,p1 := Y_train1]
X_train2 <- data.table(X_train)
X_train2[,p2 := Y_train2]
X_train3 <- data.table(X_train)
X_train3[,p3 := Y_train3]

X_test <- mo_test[,1:20]
Y_test <- mo_test[,price_range]
Y_test0 <- mo_test[,p0]
Y_test1 <- mo_test[,p1]
Y_test2 <- mo_test[,p2]
Y_test3 <- mo_test[,p3]
X_test0 <- data.table(X_test)
X_test0[,p0 := Y_test0]
X_test1 <- data.table(X_test)
X_test1[,p1 := Y_test1]
X_test2 <- data.table(X_test)
X_test2[,p2 := Y_test2]
X_test3 <- data.table(X_test)
X_test3[,p3 := Y_test3]

# fitting models
glm.fit0 <- glm(p0 ~ ., data = X_train0, family = binomial)
glm.fit1 <- glm(p1 ~ ., data = X_train1, family = binomial)
glm.fit2 <- glm(p2 ~ ., data = X_train2, family = binomial)
glm.fit3 <- glm(p3 ~ ., data = X_train3, family = binomial) # binomial for logistic regression

# train predictions
Y_train_hat0 <- predict(glm.fit0, newdata = X_train0, type = "response")
Y_train_hat1 <- predict(glm.fit1, newdata = X_train1, type = "response")
Y_train_hat2 <- predict(glm.fit2, newdata = X_train2, type = "response")
Y_train_hat3 <- predict(glm.fit3, newdata = X_train3, type = "response")

Y_train_hat_df <- data.table("0" = Y_train_hat0, "1" = Y_train_hat1, '2' = Y_train_hat2, "3" = Y
_train_hat3)

Y_train_hat <- data.table(colnames(Y_train_hat_df)[max.col(Y_train_hat_df,ties.method="first")])
Y_train_hat <- lapply(Y_train_hat[,], as.numeric)
```

```
train_accuracy <- mean(Y_train == Y_train_hat$V1)

#test predictions
Y_test_hat0 <- predict(glm.fit0, newdata = X_test0, type = "response")
Y_test_hat1 <- predict(glm.fit1, newdata = X_test1, type = "response")
Y_test_hat2 <- predict(glm.fit2, newdata = X_test2, type = "response")
Y_test_hat3 <- predict(glm.fit3, newdata = X_test3, type = "response")

Y_test_hat_df <- data.table("0" = Y_test_hat0, "1" = Y_test_hat1, '2' = Y_test_hat2, "3" = Y_tes
t_hat3)

Y_test_hat <- data.table(colnames(Y_test_hat_df)[max.col(Y_test_hat_df,ties.method="first")])
Y_test_hat <- lapply(Y_test_hat[,], as.numeric)

test_accuracy <- mean(Y_test == Y_test_hat$V1)

# Evaluation
cm <- table(observed=Y_test, predicted=Y_test_hat$V1)

diag = diag(cm) # number of correctly classified instances per class
rowsums = apply(cm, 1, sum) # number of instances per class
colsums = apply(cm, 2, sum) # number of predictions per class

precision = diag / colsums
recall = diag / rowsums
f1 = 2 * precision * recall / (precision + recall)

scores <- data.frame(precision, recall, f1)
train_accuracy
```

```
## [1] 0.8857143
```

```
test_accuracy
```

```
## [1] 0.8516667
```

```
cm
```

```
##         predicted
## observed   0   1   2   3
##        0 170   2   1   0
##        1   2 100  45   0
##        2   0  31 107   3
##        3   0   0   5 134
```

```
scores # has trouble distinguishing between 1 and 2
```

```
##   precision    recall        f1
## 0 0.9883721 0.9826590 0.9855072
## 1 0.7518797 0.6802721 0.7142857
## 2 0.6772152 0.7588652 0.7157191
## 3 0.9781022 0.9640288 0.9710145
```

This set of models identifies the lowest and highest price ranges well, but has trouble distinguishing between price range 1 and 2.

Layered Binary Regressions

Our binary first model predicts well between low and high prices (0,1) vs (2,3), and our multiple binary models do a good job predicting the lowest (0) and highest (3) price ranges.

We decided to combine the strengths of both sets of models by initially predicting high vs low, then predicting 0 vs 1 for low predictions, or 2 vs 3 for high predictions.

```
Y_train <- mo_trn[,price_binary]
X_train <- mo_trn[, 1:20]
X_train[,price_binary := Y_train]                  # training set - first binary (low-high) mode
l
X_train01 <- mo_trn[price_range < 2, c(1:20,24) ]  # training set - second layer binary model
 (0,1)
X_train23 <- mo_trn[price_range > 1, c(1:20,26) ]  # training set - second layer binary model
 (2,3)


Y_test <- mo_test[,price_binary]
X_test <- mo_test[, 1:20]
X_test[, price_binary := Y_test]

# fitting models
glm.fit <- glm(price_binary ~ ., data = X_train, family = binomial) # binomial for logistic regr
ession

glm.fit01 <- glm(p1 ~ ., data = X_train01, family = binomial)
glm.fit23 <- glm(p3 ~ ., data = X_train23, family = binomial)

######################
# 1st Prediction Layer

#test predictions
Y_test_hat <- predict(glm.fit, newdata = X_test, type = "response")
Y_test_hat <- data.table(Y_test_hat > 0.5)
Y_test_hat <- transform(Y_test_hat, V1 = as.numeric(V1))[,V1]

binary1_accuracy <- mean(Y_test == Y_test_hat)

x2 <- data.table(mo_test)
x2[, hi_lo_prediction := Y_test_hat]

low_table <- x2[hi_lo_prediction == 0, c(1:20, 24, 21)]
high_table <-  x2[hi_lo_prediction == 1, c(1:20, 26, 21)] # split based on prediction from first
layer

######################
# 2nd Prediction Layer

# if first layer predicted low, second layer predicts 0 or 1
# if first layer predicted high, second layer predicts 2 or 3

#test predictions
Y_test_hat_low <- predict(glm.fit01, newdata = low_table[, 1:21], type = "response")
Y_test_hat_low <- data.table(Y_test_hat_low > 0.5)
Y_test_hat_low <- transform(Y_test_hat_low, V1 = as.numeric(V1))[,V1]

test_accuracy_low <- mean(low_table$price_range == Y_test_hat_low) # price_range column - not us
ed as input, just for checking results

Y_test_hat_high <- predict(glm.fit23, newdata = high_table[, 1:21], type = "response")
```

```
Y_test_hat_high <- data.table(Y_test_hat_high > 0.5)
Y_test_hat_high <- Y_test_hat_high + 2
Y_test_hat_high <- transform(Y_test_hat_high, V1 = as.numeric(V1))[,V1]

test_accuracy_high <- mean(high_table$price_range == Y_test_hat_high)  # price_range column - no
t used as input, just for checking results

# Combining Results
matrix1 <- data.table(low_table)
matrix1[, prediction := Y_test_hat_low]
matrix1[, p1 := NULL]
matrix2 <- data.table(high_table)
matrix2[, prediction := Y_test_hat_high]
matrix2[, p3 := NULL]

results <- rbind(matrix1, matrix2)

# Evaluation

test_accuracy <- mean(results$price_range == results$prediction)

cm <- table(observed=results$price_range, predicted=results$prediction)

diag = diag(cm) # number of correctly classified instances per class
rowsums = apply(cm, 1, sum) # number of instances per class
colsums = apply(cm, 2, sum) # number of predictions per class

precision = diag / colsums
recall = diag / rowsums
f1 = 2 * precision * recall / (precision + recall)

scores <- data.frame(precision, recall, f1)

binary1_accuracy
```

```
## [1] 0.9766667
```

```
test_accuracy_low
```

```
## [1] 0.9713376
```

```
test_accuracy_high
```

```
## [1] 0.9370629
```

```
test_accuracy  #overall accuracy for final output through both layers
```

```
## [1] 0.955
```

```
cm
```

```
##         predicted
## observed   0   1   2   3
##        0 170   3   0   0
##        1   2 135  10   0
##        2   0   4 134   3
##        3   0   0   5 134
```

```
scores
```

```
##   precision    recall        f1
## 0 0.9883721 0.9826590 0.9855072
## 1 0.9507042 0.9183673 0.9342561
## 2 0.8993289 0.9503546 0.9241379
## 3 0.9781022 0.9640288 0.9710145
```

This approach yields accuracy almost as high as our baseline neural net model.

Logistic Models Summary: * Having more than 2 predictors complicated problem, and we had to be creative with models we applied to this dataset. * These models overall had high accuracy, indicating a linear relationship between predictors and price range.

Tree-based Models

Tree Models have the advantage of being easily adaptable to a multiclass classification problem, and are easily interpretable.

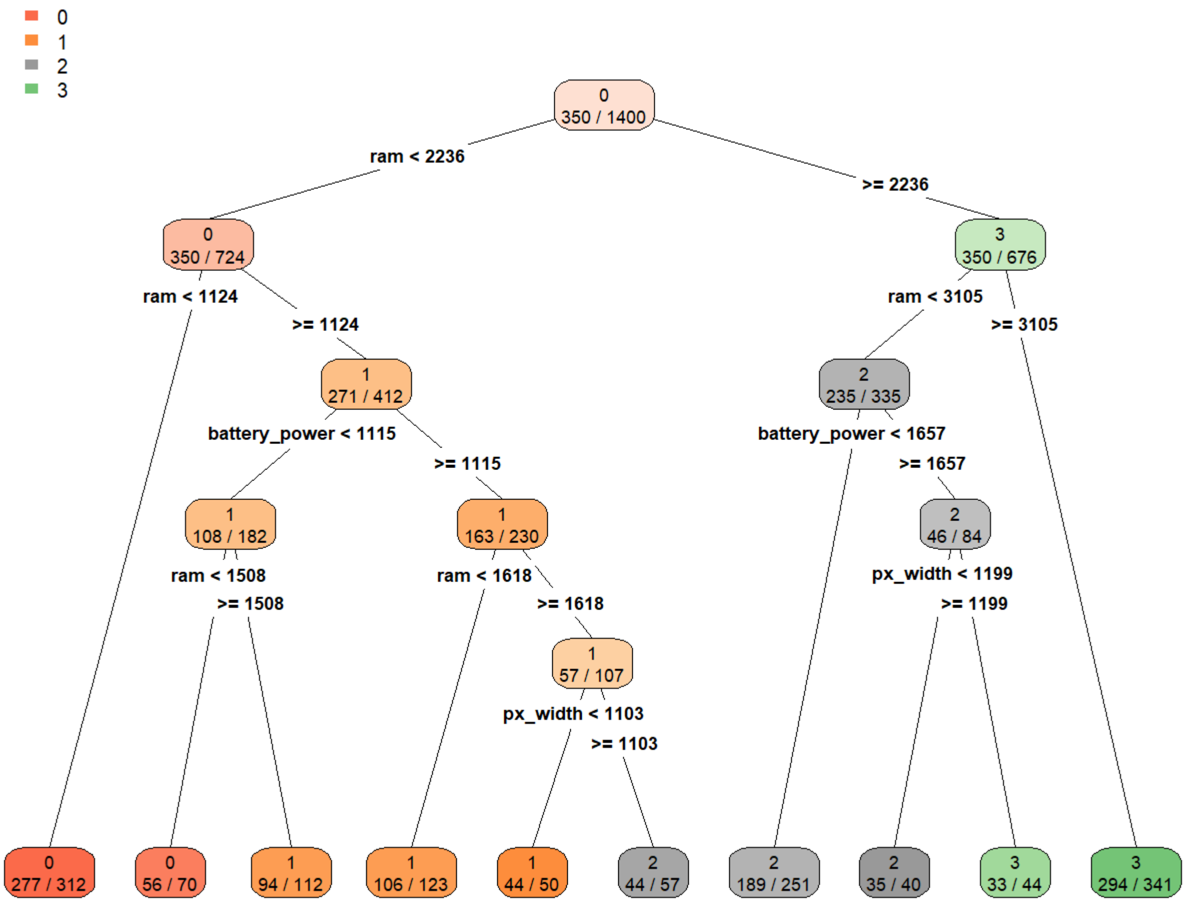
We apply several multiclass and multiple binary tree models below.

Simple Decision Tree

```
library(rpart)
library(rpart.plot)

split = sample.split(mo$price_range, SplitRatio = 0.7)
data_train = subset(mo, split == TRUE)
data_test = subset(mo, split == FALSE)

tree = rpart(price_range ~ .,method = "class", data = data_train)

tree.pred = predict(tree, newdata = data_test, type = 'class')
tree.accuracy = mean(tree.pred == data_test$price_range)
tree.accuracy
```

```
## [1] 0.8
```

```
# Pruning Tree
tree2 <- prune(tree, cp = 0.01000000)
rpart.plot(tree2, type = 4, branch = 0, extra = 2)
```



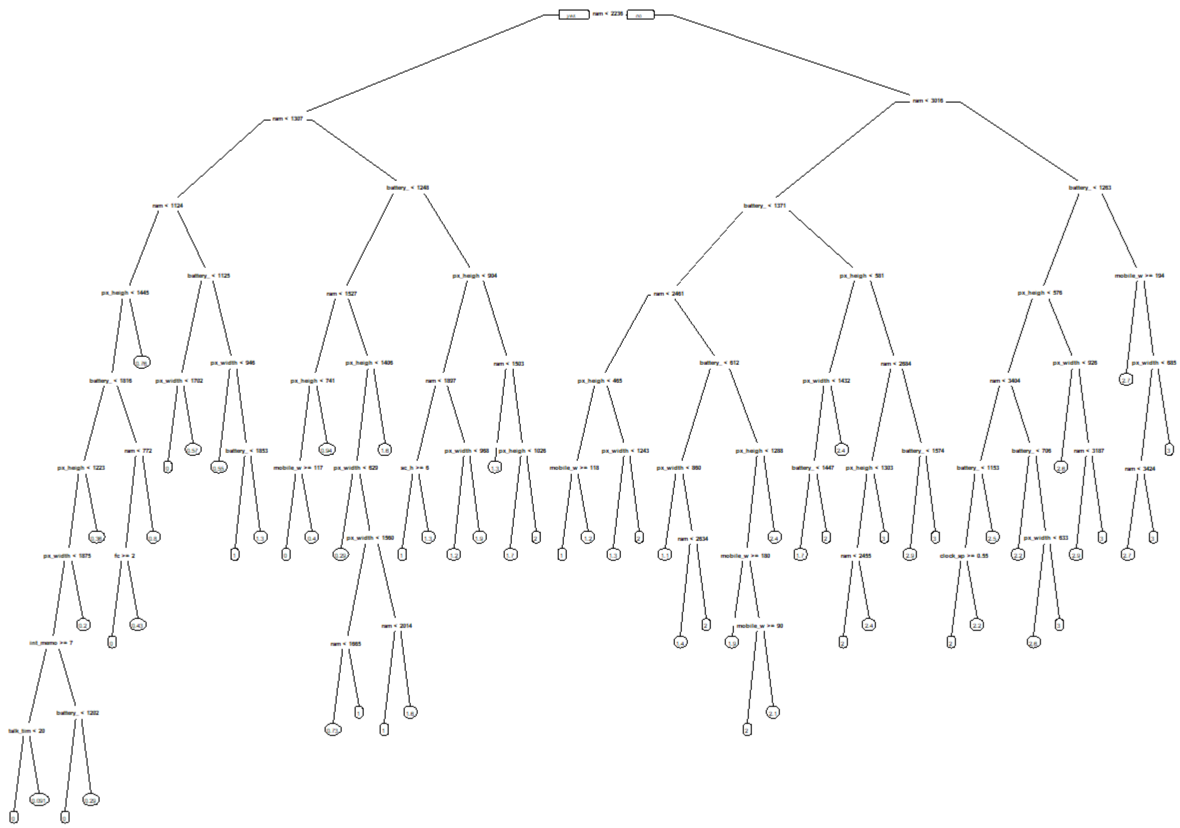

```
CFit1 <- predict(tree2, data_test, type = "class")
#ConfM1 <- table(data_train$price_range, CFit1)
#(E1 <- (sum(ConfM1) - sum(diag(ConfM1)))/sum(ConfM1))
tree.accuracy = mean(CFit1 == data_test$price_range)
tree.accuracy
```

```
## [1] 0.8
```

```
# Cross Validation
tr.control = trainControl(method = "cv", number = 10)
cp.grid = expand.grid(.cp = (0:10)*0.001)
tr = train(price_range ~., data = data_train, method = "rpart", trControl = tr.control, tuneGrid
= cp.grid)
tr
```

```
## CART
##
## 1400 samples
##   20 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1260, 1260, 1260, 1260, 1260, 1260, ...
## Resampling results across tuning parameters:
##
##   cp     RMSE       Rsquared   MAE
##   0.000  0.3555902  0.8992706  0.1818983
##   0.001  0.3752021  0.8874764  0.1981023
##   0.002  0.3806766  0.8836119  0.2117234
##   0.003  0.3851284  0.8810468  0.2218490
##   0.004  0.3973982  0.8739792  0.2412526
##   0.005  0.4220260  0.8576575  0.2890290
##   0.006  0.4292327  0.8526930  0.3074388
##   0.007  0.4325203  0.8501444  0.3164713
##   0.008  0.4356231  0.8479527  0.3232046
##   0.009  0.4382216  0.8460195  0.3305465
##   0.010  0.4428475  0.8425110  0.3235908
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was cp = 0.
```

```
# Predictions with best tree from CV
best.tree = tr$finalModel
prp(best.tree)
```



```
best.tree.pred = predict(best.tree, newdata = data_test)
tree.accuracy = mean(best.tree.pred == data_test$price_range)
tree.accuracy
```

```
## [1] 0.6216667
```

Random Forests

Multiple Binary Random Forest Classifiers

```
library(mltools)
library(randomForest)
library(tidyr)
library(tidyselect)
library(ggplot2)
library(dplyr)

mobile_data <- fread('~/R/mobile/train.csv', stringsAsFactors = T)

#set price_range as factor for one hot encoding
mobile_data$price_range <- as.factor(mobile_data$price_range)

#one hot encode the data for price range
mobile_data_one = one_hot(mobile_data,cols='price_range')

#split the data into training and test
mobile_data_one[, test:=0]
mobile_data_one[sample(nrow(mobile_data_one), 300), test:=1] # take 300 random rows and stick th
em in the test set
# now split
mobile_data_one_test <- mobile_data_one[test==1]
mobile_data_one_train <- mobile_data_one[test==0]

### Train data for each price level randomForest model, setting target variable as a factor
mobile_train_0 <- mobile_data_one_train %>% select(-(price_range_1:test))
mobile_train_0$price_range_0 <- as.factor(mobile_train_0$price_range_0)

mobile_train_1 <- mobile_data_one_train %>% select(-c(price_range_0,(price_range_2:test)))
mobile_train_1$price_range_1 <- as.factor(mobile_train_1$price_range_1)

mobile_train_2 <- mobile_data_one_train %>% select(-c((price_range_0:price_range_1),(price_range
_3:test)))
mobile_train_2$price_range_2 <- as.factor(mobile_train_2$price_range_2)

mobile_train_3 <- mobile_data_one_train %>% select(-c((price_range_0:price_range_2),test))
mobile_train_3$price_range_3 <- as.factor(mobile_train_3$price_range_3)

###Test data
mobile_predictors_test <- mobile_data_one_test %>% select(-(price_range_0:test))
#instantiate test Ys
price_0_test <- mobile_data_one_test %>% select(price_range_0)
price_0_test_f <- as.factor(price_0_test$price_range_0)

price_1_test <- mobile_data_one_test %>% select(price_range_1)
price_1_test_f <- as.factor(price_1_test$price_range_1)

price_2_test <- mobile_data_one_test %>% select(price_range_2)
price_2_test_f <- as.factor(price_2_test$price_range_2)

price_3_test <- mobile_data_one_test %>% select(price_range_3)
price_3_test_f <- as.factor(price_3_test$price_range_3)

########################################################################
```

```
####Code below did not work for the random Forest model but
###could be used for other applications

#cross validation (?)

#separate X (predictors)
mobile_predictors_train <- mobile_data_one_train %>% select(-(price_range_0:test))

#instantiate each individual train Ys and obtain the vector of the values
price_0_train <- mobile_data_one_train %>% select(price_range_0)
y_0_train <- price_0_train$price_range_0

price_1_train <- mobile_data_one_train %>% select(price_range_1)
y_1_train <- price_1_train$price_range_1

price_2_train <- mobile_data_one_train %>% select(price_range_2)
y_2_train <- price_2_train$price_range_2

price_3_train <- mobile_data_one_train %>% select(price_range_3)
y_3_train <- price_3_train$price_range_3


##########################################################################
#fit the models for each price level
#Random Forest Classifier for price range 0
fit.rndfor_0 <- randomForest(price_range_0 ~.,
                           data = mobile_train_0,
                           importance = TRUE,
                           xtest = mobile_predictors_test,
                           ytest = price_0_test_f)
#Random Forest Classifier for price range 1
fit.rndfor_1 <- randomForest(price_range_1 ~.,
                             data = mobile_train_1,
                             importance=TRUE,
                             xtest = mobile_predictors_test,
                             ytest = price_1_test_f)
#Random Forest Classifier for price range 2
fit.rndfor_2 <- randomForest(price_range_2 ~.,
                             data = mobile_train_2,
                             importance=TRUE,
                             xtest = mobile_predictors_test,
                             ytest = price_2_test_f)
#Random Forest Classifier for price range 3
fit.rndfor_3 <- randomForest(price_range_3 ~.,
                             data = mobile_train_3,
                             importance=TRUE,
                             xtest = mobile_predictors_test,
                             ytest = price_3_test_f)

#Analyze the results
# Price Range 0 train
y_hat_0 <- fit.rndfor_0$predicted
price_0_acc <- mean(y_hat_0 == y_0_train)
# Price Range 0 test
```

```
y_test_hat_0 <- fit.rndfor_0$test$predicted
price_0_acc_test <- mean(y_test_hat_0 == price_0_test$price_range_0)

#Price Range 1 train
y_hat_1 <- fit.rndfor_1$predicted
price_1_acc <- mean(y_hat_1 == y_1_train)
# Price Range 1 test
y_test_hat_1 <- fit.rndfor_1$test$predicted
price_1_acc_test <- mean(y_test_hat_1 == price_1_test$price_range_1)

#Price Range 2 train
y_hat_2 <- fit.rndfor_2$predicted
price_2_acc <- mean(y_hat_2 == y_2_train)
# Price Range 2 test
y_test_hat_2 <- fit.rndfor_2$test$predicted
price_2_acc_test <- mean(y_test_hat_2 == price_2_test$price_range_2)

#Price Range 3 train
y_hat_3 <- fit.rndfor_3$predicted
price_3_acc <- mean(y_hat_3 == y_3_train)
# Price Range 3 test
y_test_hat_3 <- fit.rndfor_3$test$predicted
price_3_acc_test <- mean(y_test_hat_3 == price_3_test$price_range_3)

################################################################################################
#Code below has the purpose of combining all 4 previous models

#Building a model for a prediction with all models
#set up y_test as a 4 level factor
price_levels <- mobile_data_one_test %>% select((price_range_0:price_range_3))
colnames(price_levels) <- c("0","1","2","3")
w <- which(price_levels==1,arr.ind = T)
mobile_data_one_test$price_level <- toupper(names(price_levels)[w[order(w[,1]),2]])
#Add these values into a data table
prediction_dt <- data.table("0" = fit.rndfor_0$test$votes[,2],
                            "1" = fit.rndfor_1$test$votes[,2],
                            "2" = fit.rndfor_2$test$votes[,2],
                            "3" = fit.rndfor_3$test$votes[,2])

label_rf <- apply(prediction_dt,1,which.max)-1
decision_dt <- data.table("predicted values"= label_rf)

decision_dt$actualvalues <- mobile_data_one_test$price_level



#Evaluate decisions
y_test <- as.numeric(decision_dt$actualvalues)
predictions <- as.numeric(decision_dt$`predicted values`)
analysis_table <- table(y_test,predictions)

diag = diag(analysis_table) # number of correctly classified instances per class
rowsums = apply(analysis_table, 1, sum) # number of instances per class
colsums = apply(analysis_table, 2, sum) # number of predictions per class
```

```
precision = diag / colsums
recall = diag / rowsums
f1 = 2 * precision * recall / (precision + recall)

scores <- data.frame(precision, recall, f1)
test_accuracy <- mean(y_test == predictions)

scores
```

```
##   precision    recall        f1
## 0 0.8961039 0.9324324 0.9139073
## 1 0.7662338 0.7468354 0.7564103
## 2 0.7625000 0.7530864 0.7577640
## 3 0.8939394 0.8939394 0.8939394
```

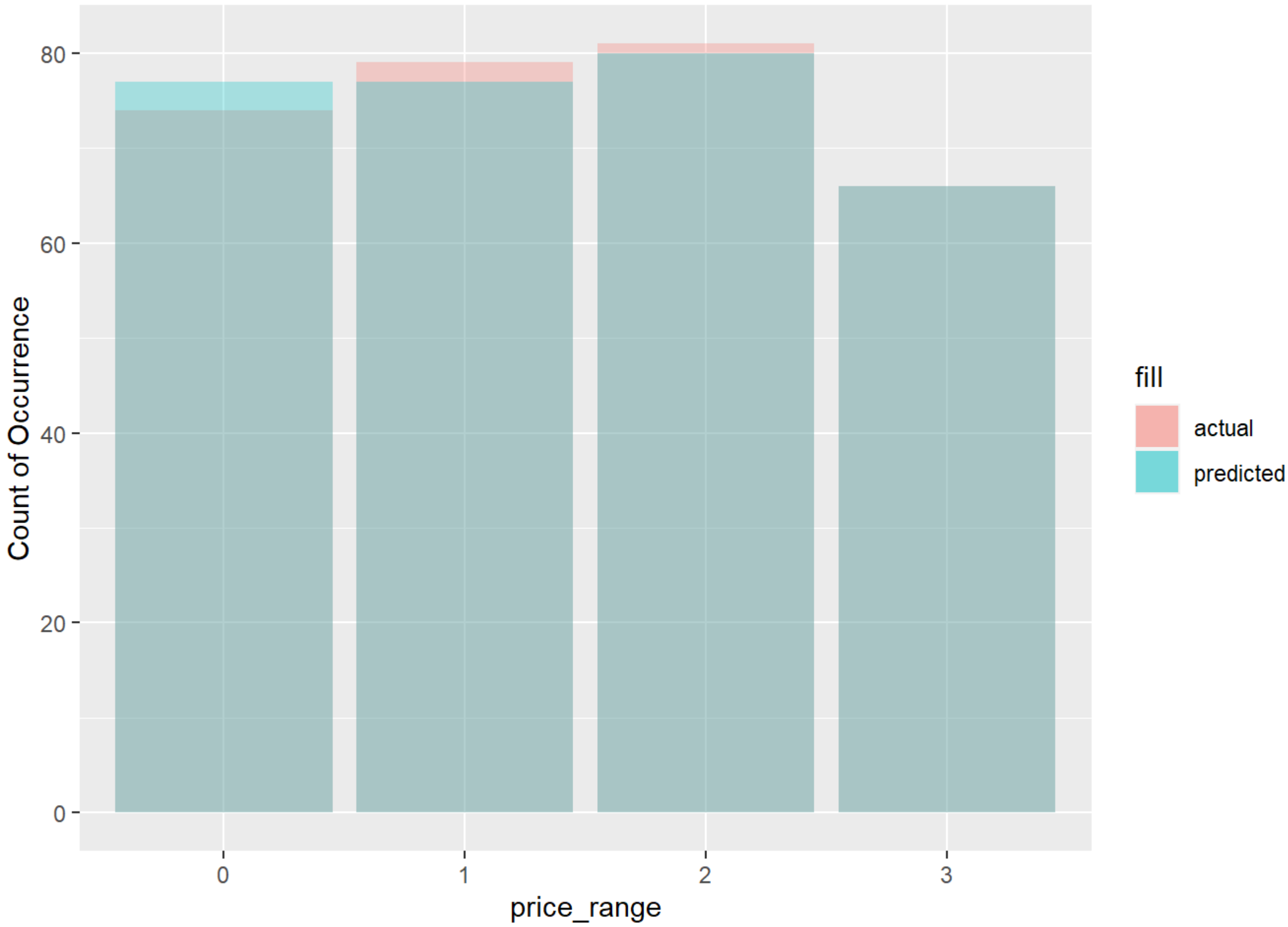
```
analysis_table
```

```
##       predictions
## y_test  0  1  2  3
##      0 69  5  0  0
##      1  8 59 12  0
##      2  0 13 61  7
##      3  0  0  7 59
```

```
test_accuracy
```

```
## [1] 0.8266667
```

```
#graphing the results
df_rf <- data.table(rowsums)
df_rf$pred <- colsums
df_rf$price_range <- c("0",'1','2','3')
colnames(df_rf) <- c('actual','predicted','price_range')
ggplot(NULL,aes(x=price_range,y=actual))+
  geom_bar(aes(fill="actual"), data= df_rf, stat = 'identity',position = "dodge",alpha=0.3)+
  geom_bar(aes(y=predicted,fill="predicted"),data=df_rf,stat = 'identity',position = "dodge",alp
ha = 0.3)+
  ylab("Count of Occurrence")
```



Multiclass Random Forest Classifier

Here we use cross validation to find optimal values of 2 Random Forest hyperparameters * m - the number of predictors randomly selected for each split * number of trees

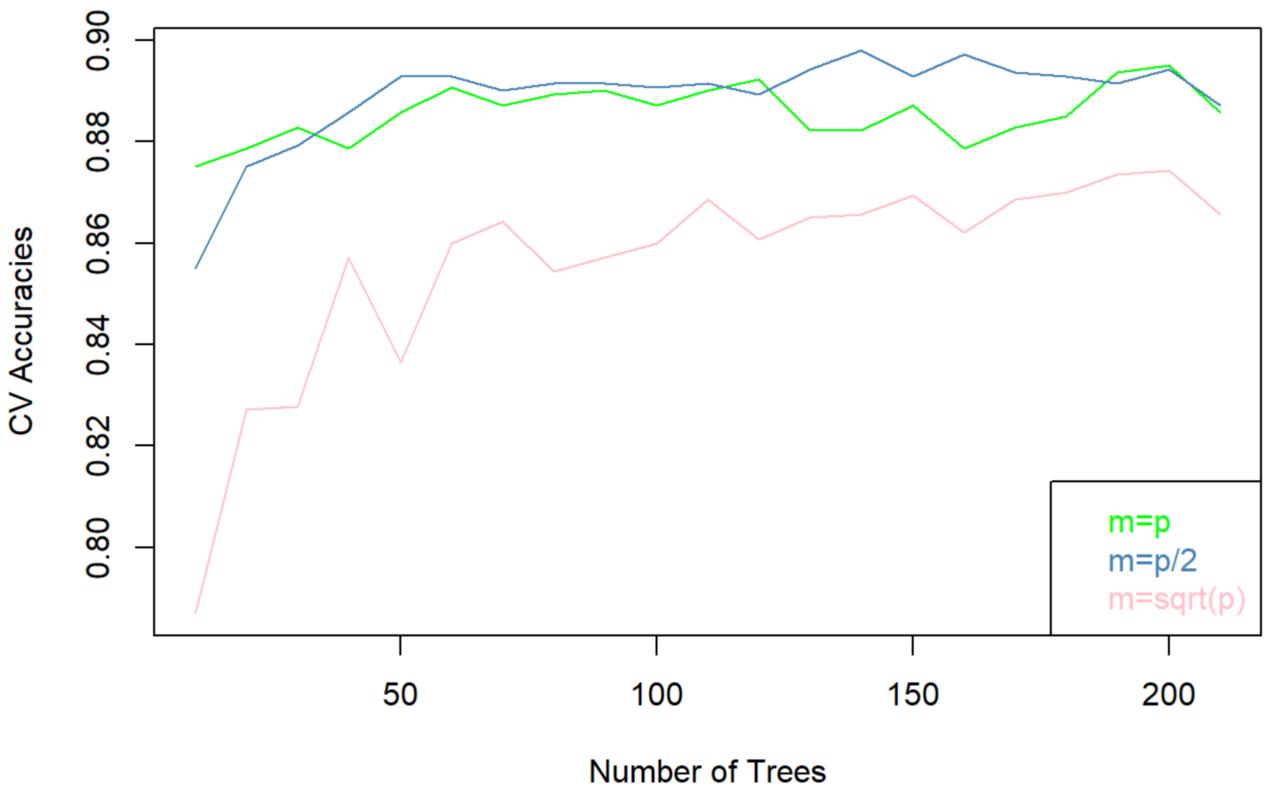
```
mo <- fread('~/R/mobile/train.csv')
mo$price_range <- as.factor(mo$price_range)
mo_obs <- nrow(mo)
mo_idx <- sample(mo_obs, size = trunc(0.70 * mo_obs))
mo_trn <- mo[mo_idx, ]
mo_test <- mo[-mo_idx, ]

Y_test <- mo_test[,price_range]

#https://www.blopig.com/blog/2017/04/a-very-basic-introduction-to-random-forests-using-r/


dd <- data.table()
dd
```

```
## Null data.table (0 rows and 0 cols)
```

```
p <- length(colnames(mo_trn)) -1
p_ov2 <- p / 2
p_sqrt <- sqrt(length(colnames(mo_trn)) -1)

trees <- seq(from = 10, to = 210, by = 10)


for (num_pred in c(p, p_ov2, p_sqrt)) {
  CV_accuracies = c()
  for (num_trees in trees){

    #Perform K-fold cross validation
    k = 5
    #Randomly shuffle the data
    mo_trn_cross <- mo_trn[sample(nrow(mo_trn)),]

    #Create K equally size folds
    folds <- cut(seq(1,nrow(mo_trn_cross)),breaks=k,labels=FALSE)

    accuracies <- c()

    #Perform K-fold cross validation

    for(i in 1:k){
      #Segement  data by fold with which() function
      testIndexes <- which(folds==i,arr.ind=TRUE)
      testData <- mo_trn_cross[testIndexes, ]
      trainData <- mo_trn_cross[-testIndexes, ]
      Y_CV <- testData$price_range


      #num_features <- sqrt(length(colnames(mo_trn)) -1)


      rf_classifier <- randomForest(price_range ~ ., data = trainData, ntree = num_trees, mtry =
num_pred, importance = TRUE )
      Y_test_hat <- predict(rf_classifier, newdata = testData, type = "class")

      accuracy <- mean(Y_test_hat == Y_CV)

      accuracies <- c(accuracies, accuracy)
    }
    CV_accuracy <- mean(accuracies)
    CV_accuracies <- c(CV_accuracies, CV_accuracy)
  }
  dd <- cbind(dd, CV_accuracies)


}
dd[, num_tree := trees]
dd
```

```
##     CV_accuracies CV_accuracies CV_accuracies num_tree
##  1:     0.8750000     0.8550000     0.7871429       10
##  2:     0.8785714     0.8750000     0.8271429       20
##  3:     0.8828571     0.8792857     0.8278571       30
##  4:     0.8785714     0.8857143     0.8571429       40
##  5:     0.8857143     0.8928571     0.8364286       50
##  6:     0.8907143     0.8928571     0.8600000       60
##  7:     0.8871429     0.8900000     0.8642857       70
##  8:     0.8892857     0.8914286     0.8542857       80
##  9:     0.8900000     0.8914286     0.8571429       90
## 10:     0.8871429     0.8907143     0.8600000      100
## 11:     0.8900000     0.8914286     0.8685714      110
## 12:     0.8921429     0.8892857     0.8607143      120
## 13:     0.8821429     0.8942857     0.8650000      130
## 14:     0.8821429     0.8978571     0.8657143      140
## 15:     0.8871429     0.8928571     0.8692857      150
## 16:     0.8785714     0.8971429     0.8621429      160
## 17:     0.8828571     0.8935714     0.8685714      170
## 18:     0.8850000     0.8928571     0.8700000      180
## 19:     0.8935714     0.8914286     0.8735714      190
## 20:     0.8950000     0.8942857     0.8742857      200
## 21:     0.8857143     0.8871429     0.8657143      210
##     CV_accuracies CV_accuracies CV_accuracies num_tree
```

```
colnames(dd) <- c("m=p", "m=p/2", "m=sqrt(p)", "num_tree")

plot(dd$`m=p` ~ dd$num_tree, type = 'l', ylim = c(min(dd[,1:3]), max(dd[,1:3])), col = 'green',
 xlab = 'Number of Trees', ylab = "CV Accuracies")
lines(dd$num_tree, dd$`m=p/2`, col = "steelblue")
lines(dd$num_tree, dd$`m=sqrt(p)`, col = "pink")

legend("bottomright",
       legend = c("m=p", "m=p/2", "m=sqrt(p)"),
       text.col = c("green", "steelblue", "pink")

)
```




```
which.max(dd$`m=p/2`) # 140 trees with m = p/2 is max cv accuracy
```

```
## [1] 14
```

```
rf_classifier <- randomForest(price_range ~ ., data = mo_trn, ntree = 140, mtry = p_ov2, importa
nce = TRUE )

Y_test_hat <- predict(rf_classifier, newdata = mo_test, type = "class")

test_accuracy <- mean(Y_test_hat == Y_test)


cm <- table(observed=Y_test, predicted=Y_test_hat)

diag = diag(cm) # number of correctly classified instances per class
rowsums = apply(cm, 1, sum) # number of instances per class
colsums = apply(cm, 2, sum) # number of predictions per class

precision = diag / colsums
recall = diag / rowsums
f1 = 2 * precision * recall / (precision + recall)

scores <- data.frame(precision, recall, f1)



test_accuracy
```

```
## [1] 0.8866667
```

```
cm
```

```
##         predicted
## observed   0   1   2   3
##        0 149  10   0   0
##        1  12 113   8   0
##        2   0  11 117  13
##        3   0   0  14 153
```

```
scores
```

```
##   precision    recall        f1
## 0 0.9254658 0.9371069 0.9312500
## 1 0.8432836 0.8496241 0.8464419
## 2 0.8417266 0.8297872 0.8357143
## 3 0.9216867 0.9161677 0.9189189
```

```
options(warn = defaultW)
```

Typically the recommended value for m in a random forest is the square root of the number of predictors, but that approach consistently under preformed compared to m = p (same as bagging), and m = p / 2.

The best cv accuracy came from a model with 140 trees and m = p / 2.