Lab10

2022-12-04

Question 1:

```
# First make sure you install and load rattle and caret!
# install.packages('rattle')
# install.packages('caret', dependencies = TRUE)
# install.packages('Rcpp')
# install.packages('e1071')
require(rpart)
require(rpart.plot)
require(rattle)
require(caret)
require(dplyr)
require(Rcpp)
require(e1071)

kickstarter <- read.csv('~/Downloads/kickstarter_lab10.csv', header = TRUE,
                        row.names = 1)
```

We have to do some data cleaning to make sure we have the right variable types and remove all NA values. First, let's make some variables into factors so that R treats them as categorical.

```
kickstarter$country <- factor(kickstarter$country)
kickstarter$state <- factor(kickstarter$state)
kickstarter$main_category <- factor(kickstarter$main_category)
```

Let's finish up cleaning the dataset by removing all NA rows.

```
kickstarter_clean <- na.omit(kickstarter)
```

We want to use validation to make sure our model doesn't overfit. To do this, we'll want to split up our dataset into what we call **train** and **test** sets.Then we'll run our model with the training sets. The most basic way to do this is to get a random sample of indices,then just select the appropriate rows. Split the data into 30% test set and 70% train set

```
# Find what 70% is (we use floor to have a round number)
num_test <- floor(nrow(kickstarter_clean) * 0.7)

# Sample the indices
test_rows <- sample(1:nrow(kickstarter_clean),num_test)

# Create test set with indices
test_kickstarter <- kickstarter_clean[test_rows,]
```

```
# Create train set with the rest
train_kickstarter <- kickstarter_clean[-test_rows,]
```

We're doing this in steps, first finding how many rows to take a random sample of, then using the sample function to draw that many numbers. Then, we use those indices to create the test set, subsetting the data frame so that it only contains those rows, and the rest get relegated to the train set, which we subset by putting a "-" in front of the indices we don't want.

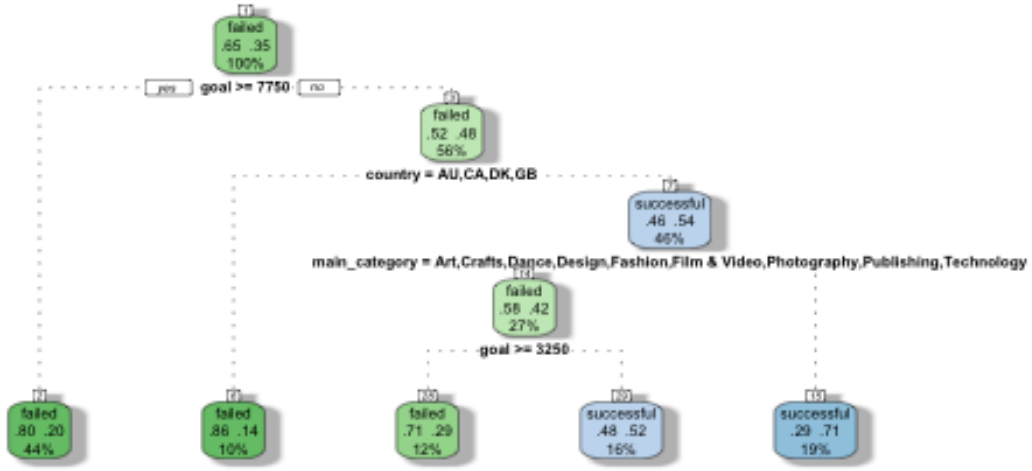
Question 2: predict state (failed vs. successful) using goal, country, and main category as the features

```
rows_test <- sample(1:nrow(kickstarter_clean),
                    floor(0.1*nrow(kickstarter_clean)))
kickstarter_test <- kickstarter_clean[rows_test,]
kickstarter_train <- kickstarter_clean[-rows_test,]

treemod <- rpart(state ~ goal + country + main_category,
                 data = train_kickstarter,
                 method = 'class',
                 control = rpart.control(minsplit = 25))
```

Let's break down each of the arguments in this function. First, we specify the model, putting the label that we want to predict on the left side of the "~" and all the features we want to include on the right. We include arguments for the dataframe from which we're taking the data, and the tree method we want to use (in this case, since we are doing a classification tree, we use 'class'). Then, we can use the control argument to set decision tree parameters. In this case, we are setting the minimum number of observations needed in a node to add a split.

We have stored the model in the treemod object. Let's look at what the model gave us. We can use summary to look at the summary of the model, but it might be easier to look a visualization instead.

```
# You can try running the summary, but it will give a LOT of output
# summary(treemod)

# The fancy tree visualization
fancyRpartPlot(treemod, sub = "")
```



Question 3: Evaluating the Model Now that we have a model, we need to test it.We can get predictions using the predict function.

```
pred <- predict(treemod,test_kickstarter)
```

Gives us the prediction scores for both failed and successful. They are basically the same thing, since you can get one from taking one minus the other. We'll focus on the successful one, since we want to identify the successful kickstarters.Let's attach it to the test set data frame.

```
test_kickstarter$state_score <- pred[,2]
```

Now, we're only really interested in the columns that represent the prediction and the actual value of whether successful or not.So, let's select just the columns of our dataframe that we need.

```
test_pred <- test_kickstarter %>% select(state, state_score)
```

Finally, we need a way to convert from the scores to an actual prediction.Typically, we use some sort of arbitrary value as a cutoff,or take a certain percentage of the most likely observations.

We'll go with the latter for this example. Let's go with a 40% threshold.

```
test_pred <- test_pred %>% arrange(desc(state_score))
test_pred$pred <- 0

top_scores <- floor(nrow(test_pred)*0.4)
test_pred$pred[1:top_scores] <- 1
```

We can get the values of precision and recall using confusionMatrix function from the caret package. First, we create a table with the confusion matrix, then run the function with the table as the argument. Note that we specify what the positive value is – since we are trying to predict which kickstarters are successful, we have 1 as our positive value. In addition, make sure your predicted values are first in the table, or else you'll get the opposite results as you want!

```
pred_tab <- table(test_pred$pred,test_pred$state)
dimnames(pred_tab)[[2]] = c("0","1")
confusionMatrix(pred_tab, positive = "1")

## Confusion Matrix and Statistics
##
##
##      0   1
##   0 138  68
##   1  60  76
##
##                Accuracy : 0.6257
##                  95% CI : (0.5721, 0.6772)
##     No Information Rate : 0.5789
##     P-Value [Acc > NIR] : 0.0442
##
##                   Kappa : 0.2265
##
##  Mcnemar's Test P-Value : 0.5361
##
##             Sensitivity : 0.5278
##             Specificity : 0.6970
##          Pos Pred Value : 0.5588
##          Neg Pred Value : 0.6699
##              Prevalence : 0.4211
##          Detection Rate : 0.2222
##    Detection Prevalence : 0.3977
##       Balanced Accuracy : 0.6124
##
##       'Positive' Class : 1
##
```

#sometimes the this command won't run if original values #weren't labeled with '0' and '1'. You can fix this with the following code.

#Question4 Note that we don't actually see the words "precision" or "recall" here – instead, we can find them by their alternate names:sensitivity (for recall) and positive predictive value (for precision).We can also use the precision and recallfunctions (also in the caret

package). Note that we use relevant to specify which outcome we're trying to predict (similar to the positive argument above).

```
precision(pred_tab, relevant = '1')

## [1] 0.5588235

recall(pred_tab, relevant = '1')

## [1] 0.5277778
```