# Modeling and Scoring with RevoScaleR

Ali Zaidi

October 11, 2016

# Introduction

# URL for Today

Please refer to the github repository for course materials
github.com/akzaidi/R-cadence

# Agenda

- We will learn in this tutorial how to train and test models with the `RevoScaleR` package.
- Use your knowledge of data manipulation to create **train** and **test** sets.
- Use the modeling functions in `RevoScaleR` to train a model.
- Use the `rxPredict` function to test/score a model.
- We will see how you can score models on a variety of data sources.
- Use a functional methodology, i.e., we will create functions to automate the modeling, validation, and scoring process.

# Prerequisites

- Understanding of `rxDataStep` and `xdfs`
- Familiarity with `RevoScaleR` modeling and datastep functions: `rxLinMod`, `rxGlm`, `rxLogit`, `rxDTree`, `rxDForest`, `rxSplit`, and `rxPredict`
- Understand how to write functions in R
- Access to at least one interesting dataset

# Typical Lifecycle

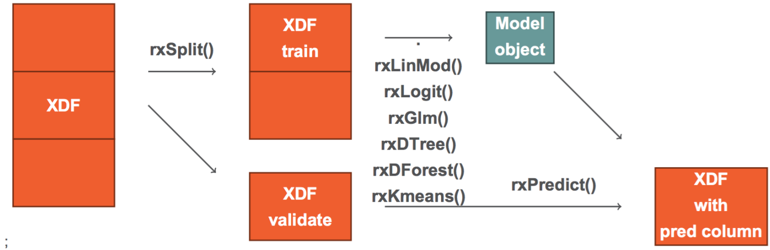



Figure 1:

Typical Modeling Lifecycle:

- Start with a data set
- Split into a training set and validation set(s)
- Use the `ScaleR` modeling functions on the train set to estimate your model
- Use `rxPredict` to validate/score your results

# Mortgage Dataset

- We will work with a mortgage dataset, which contains mortgage and credit profiles for various mortgage holders

```
mort_path <- paste(rxGetOption("sampleDataDir"), "mortDefau
file.copy(mort_path, "mortgage.xdf", overwrite = TRUE)
```

```
## [1] TRUE
```

```
mort_xdf <- RxXdfData("mortgage.xdf")
rxGetInfo(mort_xdf, getVarInfo = TRUE, numRows = 5)
```

```
## File name: /home/alizaidi/mr4ds/Student-Resources/rmarkd
## Number of observations: 1e+05
## Number of variables: 6
## Number of blocks: 10
## Compression type: zlib
## Variable information:
## Var 1: creditScore, Type: integer, Low/High: (470, 925)
```

# Transform Default to Categorical

- We might be interested in estimating a classification model for predicting defaults based on credit attributes

```
rxDataStep(inData = mort_xdf,
           outFile = mort_xdf,
           overwrite = TRUE,
           transforms = list(default_flag = factor(ifelse(d
                                                          "
                                                          "
                                 )
           )
rxGetInfo(mort_xdf, numRows = 3, getVarInfo = TRUE)
```

```
## File name: /home/alizaidi/mr4ds/Student-Resources/rmark
## Number of observations: 1e+05
## Number of variables: 7
## Number of blocks: 10
## Compression type: zlib
```

# Modeling

# Generating Training and Test Sets

- The first step to estimating a model is having a tidy training dataset.
- We will work with the mortgage data and use `rxSplit` to create partitions.
- `rxSplit` splits an input `.xdf` into multiple `.xdfs`, similar in spirit to the `split` function in base R
- output is a list
- First step is to create a split variable
- We will randomly partition the data into a train and test sample, with 75% in the former, and 25% in the latter

# Partition Function

```
create_partition <- function(xdf = mort_xdf,
                             partition_size = 0.75, ...) {
  rxDataStep(inData = xdf,
             outFile = xdf,
             transforms = list(
               trainvalidate = factor(
                   ifelse(rbinom(.rxNumRows,
                                 size = 1, prob = splitperc
                          "train", "validate")
               )
           ),
           transformObjects = list(splitperc = partition_si
           overwrite = TRUE, ...)

  splitDS <- rxSplit(inData = xdf,
                     #outFilesBase = ,
                     outFileSuffixes = c("train", "validate
                     splitByFactor = "trainvalidate",
```

# Minimizing IO

## Transforms in rxSplit

While the above example does what we want it to do, it's not very efficient. It requires two passes over the data, first to add the `trainvalidate` column, and then another to split it into train and validate sets. We could do all of that in a single step if we pass the transforms directly to `rxSplit`.

```
create_partition <- function(xdf = mort_xdf,
                             partition_size = 0.75, ...) {

  splitDS <- rxSplit(inData = xdf,
                     transforms = list(
                       trainvalidate = factor(
                         ifelse(rbinom(.rxNumRows,
                                       size = 1, prob = sp
                                "train", "validate")
                         )
                       )
```

# Generating Training and Test Sets

## List of xdfs

- The `create_partition` function will output a list xdfs

```
mort_split <- create_partition(reportProgress = 0)
names(mort_split) <- c("train", "validate")
lapply(mort_split, rxGetInfo)
```

```
## $train
## File name: /home/alizaidi/mr4ds/Student-Resources/rmarkd
## Number of observations: 75063
## Number of variables: 8
## Number of blocks: 10
## Compression type: zlib
##
## $validate
## File name: /home/alizaidi/mr4ds/Student-Resources/rmarkd
## Number of observations: 24937
```

# Build Your Model

## Model Formula

- Once you have a training dataset, the most appropriate next step is to estimate your model
- `RevoScaleR` provides a plethora of modeling functions to choose from: decision trees, ensemble trees, linear models, and generalized linear models
- All take a formula as the first object in their call

```
make_form <- function(xdf = mort_xdf,
                      resp_var = "default_flag",
                      vars_to_skip = c("default", "trainval

  library(stringr)

  non_incl <- paste(vars_to_skip, collapse = "|")

  x_names <- names(xdf)
```

# Build Your Model

## Modeling Function

- Use the make_form function inside your favorite rx modeling function
- Default value will be a logistic regression, but can update the model parameter to any rx modeling function

```
make_form()
```

```
## default_flag ~ creditScore + houseAge + yearsEmploy + c
##     year
## <environment: 0x115080db8>
```

```
estimate_model <- function(xdf_data = mort_split[["train"]]
                           form = make_form(xdf_data),
                           model = rxLogit, ...) {
```

# Build Your Model

## Train Your Model with Our Modeling Function

- Let us now train our logistic regression model for defaults using the `estimate_model` function from the last slide

```
default_model_logit <- estimate_model(mort_split$train,
                                      reportProgress = 0)
summary(default_model_logit)
```

```
## Call:
## model(formula = form, data = xdf_data, reportProgress =
##
## Logistic Regression Results for: default_flag ~ creditSc
##     houseAge + yearsEmploy + ccDebt + year
## Data: xdf_data (RxXdfData Data Source)
## File name:
##     /home/alizaidi/mr4ds/Student-Resources/rmarkdown/mor
```

# Building Additional Models

- We can change the parameters of the estimate_model function to create a different model relatively quickly

```
default_model_tree <- estimate_model(mort_split$train,
                                     model = rxDTree,
                                     minBucket = 10,
                                     reportProgress = 0)
summary(default_model_tree)
```

```
##                     Length Class      Mode
## frame                9     data.frame list
## where                0     -none-     NULL
## call                 5     -none-     call
## cptable             30     -none-     numeric
## method               1     -none-     character
## parms                3     -none-     list
## control              9     -none-     list
## splits              65     -none-     numeric
```

# Validation

# How Does it Perform on Unseen Data

## rxPredict for Logistic Regression

```
## [1] TRUE
```

- Now that we have built our model, our next step is to see how it performs on data it has yet to see
- We can use the `rxPredict` function to score/validate our results

```
default_logit_scored <- rxPredict(default_model_logit,
                                  mort_split$validate,
                                  "scored.xdf",
                                 writeModelVars = TRUE,
                                 extraVarsToWrite = "defau
                                 predVarNames = c("pred_lo
rxGetInfo(default_logit_scored, numRows = 2)
```

# Visualize Model Results

```r
rxRoc(actualVarName = "default",
      predVarNames ="pred_logit_default",
      data = default_logit_scored)
```

```
##    threshold        predVarName sensitivity specificity
## 1       0.00 pred_logit_default 1.000000000   0.0000000
## 2       0.01 pred_logit_default 0.886178862   0.9407995
## 3       0.02 pred_logit_default 0.780487805   0.9648988
## 4       0.03 pred_logit_default 0.699186992   0.9751753
## 5       0.04 pred_logit_default 0.650406504   0.9804143
## 6       0.05 pred_logit_default 0.609756098   0.9844040
## 7       0.06 pred_logit_default 0.593495935   0.9871847
## 8       0.07 pred_logit_default 0.536585366   0.9889175
## 9       0.08 pred_logit_default 0.471544715   0.9902877
## 10      0.09 pred_logit_default 0.455284553   0.9914967
## 11      0.10 pred_logit_default 0.430894309   0.9925848
## 12      0.11 pred_logit_default 0.430894309   0.9935117
## 13      0.12 pred_logit_default 0.414634146   0.9941162
```

# Testing a Second Model

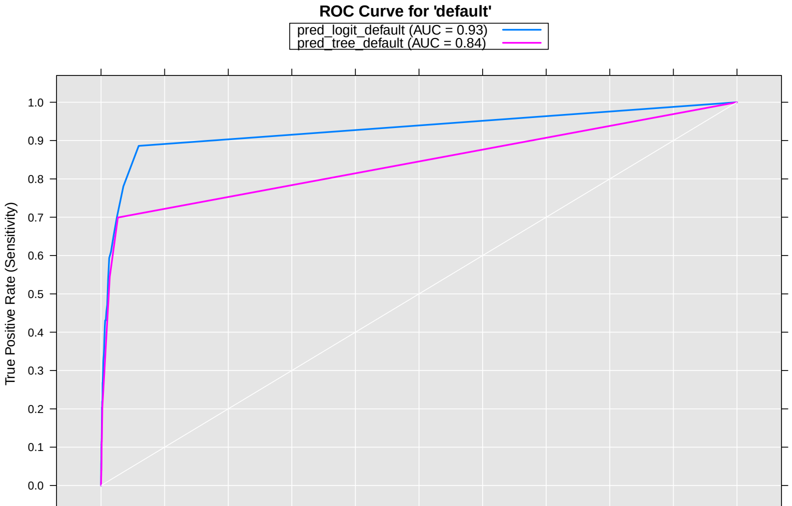
## rxPredict for Decision Tree

- We saw how easy it was to train on different in the previous sections
- Similary simple to test different models

```
default_tree_scored <- rxPredict(default_model_tree,
                                  mort_split$validate,
                                  "scored.xdf",
                                  writeModelVars = TRUE,
                                 predVarNames = c("pred_tre
                                                  "pred_tre
```

# Visualize Multiple ROCs

```r
rxRocCurve("default",
           c("pred_logit_default", "pred_tree_default"),
           data = default_tree_scored)
```



# Lab - Estimate Other Models Using the Functions Above

# Ensemble Tree Algorithms

Two of the most predictive algorithms in the RevoScaleR package are the rxBTrees and rxDForest algorithms, for gradient boosted decision trees and random forests, respectively.

Use the above functions and estimate a model for each of those algorithms, and add them to the default_tree_scored dataset to visualize ROC and AUC metrics.

```
## Starter code

default_model_forest <- estimate_model(mort_split$train,
                                       model = rxDForest,
                                       nTree = 100,
                                       importance = TRUE,
                                       reportProgress = 0)

default_forest_scored <- rxPredict(default_model_forest,
                                   mort_split$validate,
                                  "scored.xdf",
```

# More Advanced Topics

# Scoring on Non-XDF Data Sources

## Using a CSV as a Data Source

- The previous slides focused on using xdf data sources
- Most of the `rx` functions will work on non-xdf data sources
- For training, which is often an iterative process, it is recommended to use xdfs
- For scoring/testing, which requires just one pass through the data, feel free to use raw data!

```
csv_path <- paste(rxGetOption("sampleDataDir"),
                  "mortDefaultSmall2009.csv",
                  sep = "/")
file.copy(csv_path, "mortDefaultSmall2009.csv", overwrite =
```

```
## [1] TRUE
```

```
mort_csv <- RxTextData("mortDefaultSmall2009.csv")
```

# Regression Tree

- For a slightly different model, we will estimate a regression tree.
- Just change the parameters in the `estimate_model` function

```
tree_model_ccdebt <- estimate_model(xdf_data = mort_split$t
                                    form = make_form(mort_s
                                                     "ccDeb
                                                     vars_t

                                    model = rxDTree)
# plot(RevoTreeView::createTreeView(tree_model_ccdebt))
```

# Test on CSV

```
if (file.exists("mort2009predictions.xdf")) file.remove("mo
```

```
## [1] TRUE
```

```
rxPredict(tree_model_ccdebt,
          data = mort_csv,
          outData = "mort2009predictions.xdf",
          writeModelVars = TRUE)

mort_2009_pred <- RxXdfData("mort2009predictions.xdf")
rxGetInfo(mort_2009_pred, numRows = 1)
```

```
## File name: /home/alizaidi/mr4ds/Student-Resources/rmarko
## Number of observations: 10000
## Number of variables: 7
## Number of blocks: 1
## Compression type: zlib
## Data (1 row starting with row 1):
```

# Multiclass Classification

# Convert Year to Factor

- We have seen how to estimate a binary classification model and a regression tree
- How would we estimate a multiclass classification model?
- Let's try to predict mortgage origination based on other variables
- Use `rxFactors` to convert *year* to a *factor* variable

```
mort_xdf_factor <- rxFactors(inData = mort_xdf,
                             factorInfo = c("year"),
                             outFile = "mort_year.xdf",
                             overwrite = TRUE)
```

# Convert Year to Factor

```
rxGetInfo(mort_xdf_factor, getVarInfo = TRUE, numRows = 4)
```

```
## File name: /home/alizaidi/mr4ds/Student-Resources/rmarkd
## Number of observations: 1e+05
## Number of variables: 7
## Number of blocks: 10
## Compression type: zlib
## Variable information:
## Var 1: creditScore, Type: integer, Low/High: (470, 925)
## Var 2: houseAge, Type: integer, Low/High: (0, 40)
## Var 3: yearsEmploy, Type: integer, Low/High: (0, 14)
## Var 4: ccDebt, Type: integer, Low/High: (0, 14094)
## Var 5: year
##        10 factor levels: 2000 2001 2002 2003 2004 2005 2
## Var 6: default, Type: integer, Low/High: (0, 1)
## Var 7: default_flag
##        2 factor levels: current default
## Data (4 rows starting with row 1):
```

# Estimate Multiclass Classification

- You know the drill! Change the parameters in `estimate_model`:

```
tree_multiclass_year <- estimate_model(xdf_data = mort_xdf_
                                       form = make_form(mort_x
                                                        "year'
                                                        vars_t

                                       model = rxDTree)
```

# Predict Multiclass Classification

- Score the results

```
multiclass_preds <- rxPredict(tree_multiclass_year,
                              data = mort_xdf_factor,
                              writeModelVars = TRUE,
                              outData = "multi.xdf",
                              overwrite = TRUE)
```

# Predict Multiclass Classification

- View the results
- Predicted/scored column for each level of the response
- Sum up to one

```
rxGetInfo(multiclass_preds, numRows = 3)
```

# Conclusion

## Thanks for Attending!

- Any questions?
- Try different models!
- Try modeling with `rxDForest`, `rxBTrees`: have significantly higher predictive accuracy, somewhat less interpretability