Microsoft

Building Predictive Models with Microsoft R Server

Contents

Overview

Summary

This lab explores predictive modeling and testing the performance of the model with modules from ScaleR

It will walk through loading data from a csv file to an external data frame, manipulating the data frame and building models with a training data set. Then the performance of the models will be evaluated using a test data set.

Business Case

Many use cases involve the usage of sounds. This may be in the realm of music classification, speaker identification, and analysis of sound emitted by machines. Experienced service technicians are able to predict machine failure by listening to the noise these machines emit.

In our case, we will use data from 550.000 songs. These are a subset of the freely available Million Song Dataset (http://labrosa.ee.columbia.edu/millionsong/). The core of the dataset is the feature analysis and metadata for one million songs, provided by The Echo Nest (http://the.echonest.com/). The dataset does not include any audio, only the derived features.

We will predict the year of the song from the metadata provided in the data set. We also will classify the data and see if the algorithm can learn if the song is form our millennium or the previous one.

Objectives

Upon completing this lab, you will have hands-on experience with the following functions and concepts related the Microsoft R server:

- Data preparation
  - Reading data from a csv data set into an external data frame (xdf)
  - Getting information about the data (rxSummary, rxGetInfo, rxHistogram)
  - Changing Metadata of xdf
  - Processing the data with rxDataStep
  - Splitting into training and test set
- Building predictive models
  - Building a linear model, a general linear model (logistic regression), a random forest and a boosted decision tree
  - Predicting value on test sets
  - Calculate errors, accuracy and AUC values

Lab Requirements/Prerequisites

A Microsoft R server installation is required.

To speed-up the lab, the participants may download the million song data set as described in the next paragraph “Get the data”.

Use Case

Get the data

Download the YearPredictionMSD.txt.zip file containing the data set from here:

http://archive.ics.uci.edu/ml/machine-learning-databases/00203/

and extract it into data directory contained under the R project folder for this lab. Already contained in this folder is a file called songs_15k.xdf, therefore, you should now have two files called YearPredictionMSD.txt and songs_15k.xdf in this directory.

Load the data

Execute the code contained in **01_BuildSupervisedModels.R** (available in the R Project folder for this lab), and follow the code description below.

```
# Optionally: clean the workspace
rm(list=ls())

# replace the directory: Where did you store the data files, use \\ instead of \
# e.g.; bigDataDir <- "data"
bigDataDir <- "data"
setwd(bigDataDir)
songs_15k <- RxXdfData("songs_15k.xdf")
```

Until this point you just have set a pointer into the file. No data has been loaded into the R environment.

To get a first overview over the data, run the following code:

```
rxGetInfo(songs_15k, getVarInfo = TRUE)
```

We have a data set with 12 “timbre averages”, 78 “timbre covariances” and a year as columns.

Each song has been divided into short segments and 12 “timbre values” have been created per segment. The site is not clear about the exact algorithm used to create these values (see, e.g., https://developer.echonest.com/forums/thread/155).They look like some spectral values and higher order moments. The data set contains per song:

- The year of the song
- 12 averages over the segments,
- (12*11)/2 +12 covariances between the segments

In the next step we will look intro the distribution of the years. To do this we will perform two steps at once:

We will create a new variable “year10” via a transforms which will

```
rxSummary(~ year10,
        data=songs_15k,
        summaryStats=c("validObs"),
        transforms=list(year10 = cut(year,seq(1920,2010,10))))
```

indicate if a song was created in a certain 10-year interval, e.g., between 1950 and 1960.

Then we will count the valid observations in this interval.

So we can see that there are about 300 songs for the intervals before 1950 and about 2000 songs per interval for the later years.

As a first look to understand which may be the important columns to predict the year, we will build a decision tree. To do this we need a formula, which indicates that we want to have the year dependent on the other variables. (The formula “year ~ . “ is not supported.) So we create it programmatically:

```
# A first look into the data:
# Build a decision tree and look which variable influences the year.
# to do this we need a formula.
# Since we have a lot of variables, we will create it programmatically
(avgs <- paste("timbre_avg",1:12,sep="_"))
covs <- paste("timbre_cov",1:78,sep="_")
(a <- paste(avgs, collapse=" + "))
c <- paste(covs, collapse=" + ")

(form1 <- as.formula(paste("year ~ ",paste(a,c,sep=" + "))))
# Build the tree
tree <- rxDTree(form1,data=songs_15k,maxDepth = 3)
tree
```

The data set would fit in the memory, but for instructional purposes, it has been created in blocks of roughly 1500 rows.

From the result we can see that the timbre averages 6, 1 and 2 seem to be the crucial ones.

Process data for analysis

As a first step we will split the data into a training set (80%) and a test set (20%). To do this we create a random variable between 1

```
# First create a random variable from 1 to 10 and select afterwards
rxDataStep(inData=songs_15k,
        outFile = "songs.xdf",
        transforms=list(splitvar = as.integer(runif(.rxNumRows,1,11))),
        overwrite=TRUE)
rxGetInfo("songs.xdf",getVarInfo = TRUE)
# Create training and test set
rxDataStep(inData="songs.xdf",
        outFile="train_15k",
        rowSelection = (splitvar < 9),
        varsToDrop = "splitvar",
        overwrite=TRUE)
rxDataStep(inData="songs.xdf",
        outFile="test_15k",
        rowSelection = (splitvar > 8),
        varsToDrop = "splitvar",
        overwrite=TRUE)
```

and 10. The rows with value 1-8 will be the training set, the others the test set.

We do not keep the splitvar in the training and test set (varsToDrop).

In the next step we create a linear model, using the same formula

```
# Create a linear regression model
year_lin <- rxLinMod(form1, data = "train_15k")
year_lin

# predict
rxPredict(modelObject=year_lin,
          data="test_15k",
          outData="lin_test",
          predVarNames="Y_Pred",
          overwrite = TRUE)

rxMerge(inData1="test_15k",inData2="lin_test",outFile="test_r2",
        type="oneToOne",
        varsToKeep1 = "year",varsToKeep2 = "Y_Pred",
        overwrite = TRUE)
rxGetInfo("test_r2",getVarInfo=TRUE,numRows=5)
```

as in the decision tree creation. Then we predict the year value from the test set. We merge the prediction result with the original year column of the test set.

As next step we want to calculate error values. To do this we again use rxDataStep to add columns for the absolute error and the squared error and then look for the mean of these columns via rxSummary.

```
# Calculate the errors (absolute and squared)
rxDataStep(inData="test_r2",
           outFile="reglin_error",
           transforms=list(
             abs_err = abs(Y_Pred-year),
             sq_err = (Y_Pred-year)^2
           ),
           varsToKeep = c("Y_Pred","year"),
           overwrite=TRUE)
rxGetInfo("reglin_error",getVarInfo=TRUE,numRows=5)
rxSummary(~ abs_err+sq_err, data="reglin_error",summaryStats=c("mean"))
```

We have a mean absolute error of 11 years and a squared error of 205, which means a RMSE of about 14.
This should be compared to the standard deviation of the test set,

```
rxSummary(~ year,data="test_15k")
```

which is about 20.

Exercise:
Use a random forest:
rxDForest(form1, data = "train_15k", maxDepth = 3,nTree = 20)
and perform the equivalent steps.
The rxDForest will run in several iterations, about 5 minutes on a laptop equipped with ssd. The variables (maxDepth = 3 and

nTree = 20) are much too low to get a good regression. Better values will be maxDepth = 7 and nTree = 500), but this is not working in the time limits of this exercise. A RMSE of <10 can be reached with higher values.

Large Data Set for Classification

The previous exercise could have been performed in memory with standard R libraries. They even may have been faster, since the reading and writing of blocks to disk can be avoided.

Now we will run a model on a large data set. We will use the 550k rows from the million song data. This data set is structured for the following train / test split:
train: first 463,715 examples
test: last 51,630 examples
It avoids the 'producer effect' by making sure no song from a given artist ends up in both the train and test set.

Data preparation

First we read the training and test set into xdf files:

```
infile_csv <- "YearPredictionMSD.txt"

# Select the first 463715 rows as train, the last ones as test
rxTextToXdf(inFile = infile_csv, outFile = "train.xdf",  stringsAsFactors = F,
rowsPerRead = 50000, overwrite = T,numRows = 463715)

rxTextToXdf(inFile = infile_csv, outFile = "test.xdf",  stringsAsFactors = F,
rowsPerRead = 10000, overwrite = T,rowsToSkip = 463715)

test <- RxXdfData( "test.xdf" )
train <- RxXdfData("train.xdf")

rxGetInfo(data = "train.xdf", getVarInfo = TRUE)
rxGetInfo(data = "test.xdf", getVarInfo = TRUE)
```

The next step is to change the variable names to meaningful names. We again create it programmatically through a function. We first create a new entry var_info[[name]]$newName and set it to the new column name and then use rxSetVarInfo to rename the columns.

```
# First create column info: year, 12 timbre averages, 78 timbre covariances
(avgs <- paste("timbre_avg",1:12,sep="_"))
covs <- paste("timbre_cov",1:78,sep="_")
new_names <- c("year",avgs,covs)

changeColNames <- function(data){
  var_info <- rxGetVarInfo(data)
  i=1
  for (name in names(var_info)){
    var_info[[name]]$newName <- new_names[i]
    i<- i+1
  }
  rxSetVarInfo(var_info,data)
}
changeColNames(test)
changeColNames(train)

# look onto the data
rxGetInfo(data = "train.xdf", getVarInfo = TRUE)
rxGetInfo(data = "test.xdf", getVarInfo = TRUE)

# A histogram of the years
rxHistogram(~year,data=train)
rxHistogram(~year,data=test)
(quantiles <- rxQuantile(varName="year",data=train,probs=seq(0,1.0,0.1)))
```

We see that the data is not evenly distributed over the years. This creates the well-known problem to train values which are underrepresented in the data set. (This is the reason why we did the regression on a more evenly distributed subset of 15k rows).

From the quantiles we can see that roughly half of the data is from the new millennium. We create a variable “new_millenium” which indicates if the song is created in our present millennium. Now we build a boosted decision tree to predict the

```
# We create a variable "new_millenium
rxDataStep(inData="train",
          outFile="train_millenium",
          transforms=list(new_millenium = (year > 1999)),
          overwrite=TRUE)
rxGetInfo(data="train_millenium",getVarInfo = TRUE)
rxSummary(~ new_millenium, data="train_millenium")

rxDataStep(inData="test",
          outFile="test_millenium",
          transforms=list(new_millenium = (year > 1999)),
          overwrite=TRUE)
rxGetInfo(data="test_millenium",getVarInfo = TRUE)
rxSummary(~ new_millenium, data="test_millenium")
```

new_millenium. Again we create the formula first. To save time, we set nTree to 10. However this is a very low value.

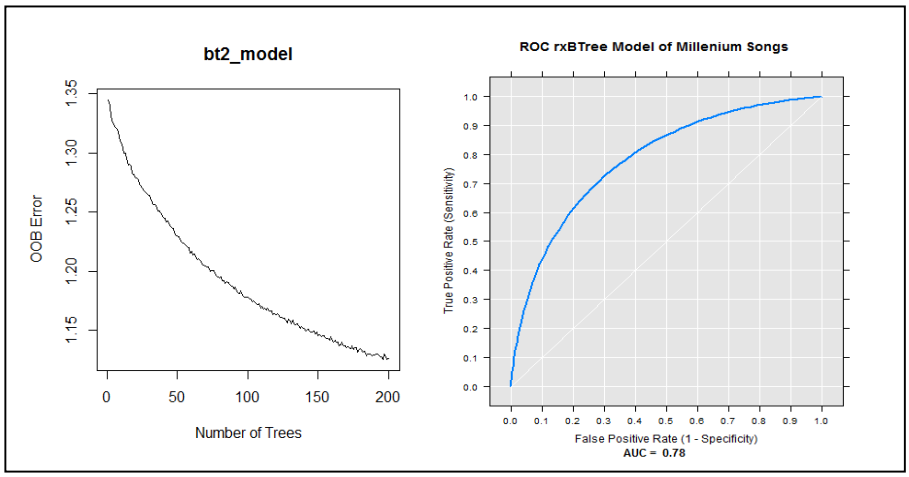
Now we predict the Millenium, calculate the Area Under the Curve (AUC) and plot a Receiver Operating Characteristic (ROC) curve.

The AUC is 0.66, a quite low value mostly due to the small number of trees. For comparison: Building a model with 100 Trees yields the following plot which shows that nTree = 10 is too low. The model was created on an 8GB Surface Pro. It just took more time to run.

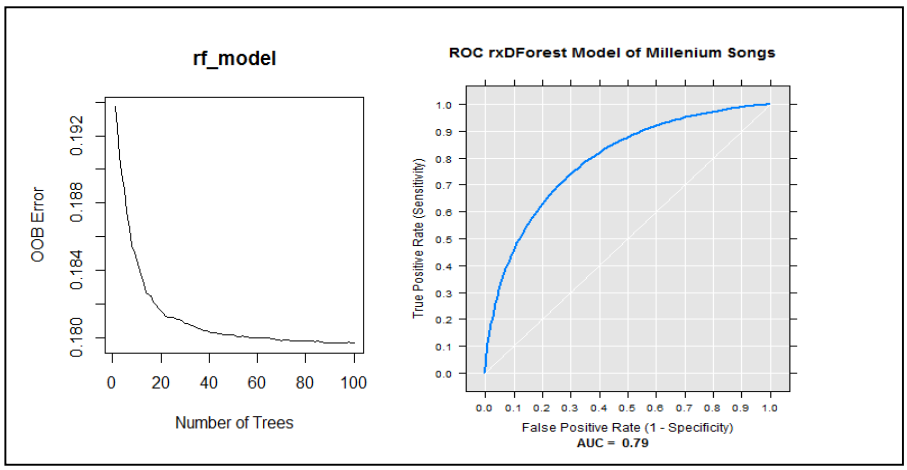
```
# predict
rxPredict(modelObject=bt_model,
          data="test_millenium",
          predVarNames="Y_Prob",
          overwrite = TRUE)
rxGetInfo("test_millenium",getVarInfo=TRUE)

# ROC curve
ROC <-rxRoc(actualVarName = "new_millenium",
            predVarNames = "Y_Prob",
            data = "test_millenium")
rxAuc(ROC)      # Compute the AUC

rxRocCurve (actualVarName = "new_millenium",
            predVarNames = "Y_Prob",
            data = "test_millenium",
            title ="ROC rxBTrees Model of Millenium Songs")
```



For a random Forest with 100 trees we get:



Exercises:

- Do the same classification with a General Linear Model (use rxGlm). It will perform better and run much faster.

Clustering

Execute the code contained in **02_BuildUnsupervisedModels.R** (available in the R Project folder for this lab), and follow the code description below.

We will create clusters from the 15k songs and look if they correlate with the decades the songs were written.

To start we use the 15k songs, programmatically create a formula and perform the clustering. To make it easier for the participant, we do create the formula from scratch.

```
songs_15k <- RxXdfData("songs_15k.xdf")
rxGetInfo("songs_15k",getVarInfo = T)

# build 10 clusters. We again need the formula
# Since we have many variables, we build the formula:
(avgs <- paste("timbre_avg",1:12,sep="_"))
covs <- paste("timbre_cov",1:78,sep="_")
(a <- paste(avgs, collapse=" + "))
(c <- paste(covs, collapse=" + "))
(form <- as.formula(paste(" ~ ",paste(a,c,sep=" + "))))

clust <- rxKmeans(form,data= songs_15k,numClusters = 9,
algorithm = "lloyd",
                outFile = "song_clust", outColName = "Cluster",
                seed = 42,overwrite = TRUE)
song_clust <- RxXdfData( "song_clust.xdf" )
rxGetInfo(song_clust,getVarInfo = T)
```

Then we create a factor for each decade of the data set and merge it with the cluster:

```
# Create a factor for each interval of 10 years:
rxDataStep(inData = songs_15k, outFile = "decades.xdf",
          transforms = list(
            decade = cut(year,breaks =
seq(1920,2010,10),labels=as.character(2:10))),
          overwrite = T)
rxGetInfo("decades.xdf",getVarInfo = T)

rxMerge(inData1="decades.xdf",inData2="song_clust",outFile=
"pairs",
       type="oneToOne",
       varsToKeep1 = "decade",varsToKeep2 = "Cluster",
       overwrite = TRUE)
rxGetInfo("pairs",getVarInfo = T)
pairs <- RxXdfData( "pairs.xdf" )
```

As last step we cross-tabulate the cluster with the decade:

```
clust_songs_tab <- rxCrossTabs(~decade:as.factor(Cluster),
data = pairs)

print(clust_songs_tab,output="counts")
```

From the result, we can see that the clusters 6 and 7 have some focus on the 50es, whereas the later decades are focused in the

```
       as.factor.Cluster.
decade   1  2  3   4   5   6   7   8   9
    2   20  1  7  22   8  18  35  95  58
    3    6  4 14  54  12  18  44  90  22
    4   55  2 21  78  31  23  47  61  28
    5  341 28 25 297 186 107 313 506 265
    6  465  8 16 167 106  46 285 667 687
    7  394  3 23 165 100  38 165 701 805
    8  287  9 26 156  85  35 148 578 960
    9  347 22 27 192 144  49 201 521 882
    10 511 34 50 240 186  42 139 470 926
```

clusters 9, also when considering the absolute sizes of the clusters, which differ considerably.

Terms of Use

By using this Hands-on Lab, you agree to the following terms:

The technology/functionality described in this Hands-on Lab is provided by Microsoft Corporation in a “sandbox” testing environment for purposes of obtaining your feedback and to provide you with a learning experience. You may only use the Hands-on Lab to evaluate such technology features and functionality and provide feedback to Microsoft. You may not use it for any other purpose. You may not modify copy, distribute, transmit, display, perform, reproduce, publish, license, create derivative works from, transfer, or sell this Hands-on Lab or any portion thereof.

COPYING OR REPRODUCTION OF THE HANDS-ON LAB (OR ANY PORTION OF IT) TO ANY OTHER SERVER OR LOCATION FOR FURTHER REPRODUCTION OR REDISTRIBUTION IS EXPRESSLY PROHIBITED.

THIS HANDS-ON LAB PROVIDES CERTAIN SOFTWARE TECHNOLOGY/PRODUCT FEATURES AND FUNCTIONALITY, INCLUDING POTENTIAL NEW FEATURES AND CONCEPTS, IN A SIMULATED ENVIRONMENT WITHOUT COMPLEX SET-UP OR INSTALLATION FOR THE PURPOSE DESCRIBED ABOVE. THE TECHNOLOGY/CONCEPTS REPRESENTED IN THIS HANDS-ON LAB MAY NOT REPRESENT FULL FEATURE FUNCTIONALITY AND MAY NOT WORK THE WAY A FINAL VERSION MAY WORK. WE ALSO MAY NOT RELEASE A FINAL VERSION OF SUCH FEATURES OR CONCEPTS. YOUR EXPERIENCE WITH USING SUCH FEATURES AND FUNCITONALITY IN A PHYSICAL ENVIRONMENT MAY ALSO BE DIFFERENT.

FEEDBACK. If you give feedback about the technology features, functionality and/or concepts described in this Hands-on Lab to Microsoft, you give to Microsoft, without charge, the right to use, share and commercialize your feedback in any way and for any purpose. You also give to third parties, without charge, any patent rights needed for their products, technologies and services to use or interface with any specific parts of a Microsoft software or service that includes the feedback. You will not give feedback that is subject to a license that requires Microsoft to license its software or documentation to third parties because we include your feedback in them. These rights survive this agreement.

MICROSOFT CORPORATION HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS WITH REGARD TO THE HANDS-ON LAB, INCLUDING ALL WARRANTIES AND CONDITIONS OF MERCHANTABILITY, WHETHER EXPRESS, IMPLIED OR STATUTORY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. MICROSOFT DOES NOT MAKE ANY ASSURANCES OR REPRESENTATIONS WITH REGARD TO THE ACCURACY OF THE RESULTS, OUTPUT THAT DERIVES FROM USE OF THE VIRTUAL LAB, OR SUITABILITY OF THE INFORMATION CONTAINED IN THE VIRTUAL LAB FOR ANY PURPOSE.

DISCLAIMER

This lab contains only a portion of the features and enhancements in Microsoft Azure. Some of the features might change in future releases of the product.