Building and deploying a model with the Causata R package

Justin Hemann, www.causata.com

Version 4.2-0, July 18, 2013

This vignette shows how functions in the Causata R package are used to build a logistic regression model in R and deploy it to Causata.

1 A simple example

A very simple example is shown below, illustrating a complete model build process from data extraction to model migration. Note that several of the steps below require a Causata server. The `Connect` and `GetData` functions extract data from a Causata server, and the `UploadModel` function migrates a model to a Causata server.

The first step is to extract data using the Causata SQL interface. The focal point is set at page view events, meaning that the variables are built with respect to customers' page views. The variables returned in the query are:

- `mortgage_application_approved__n30d` is a Boolean variable indicating if a mortgage application is approved in the 30 days after a page view. This is used as the dependent variable.
- `total_assets__AP` is a numeric variable representing the total deposits for a customer over all past.
- `web_has_browsed_on_ipad__l7d` is a Boolean variable indicating if a customer has visited the website with an iPad in the 7 days before the page view.

```
library(Causata)
library(glmnet)

# Load data using the SQL interface.
# A page view event is selected as a focal point.
conn <- Connect(group='fsdemo')
query <- paste(
  'SELECT mortgage_application_approved__n7d, total_assets__AP,',
```

```
  'web_has_browsed_on_ipad__17d',
  'FROM Scenarios S, page_view P',
  'WHERE where S.profile_id = P.profile_id',
  'AND S.focal_point = P.timestamp',
  'LIMIT 20000')
df <- GetData(conn, query)
```

Next a formula is constructed, missing values in the data are replaced, and a glmnet model is trained. The model predicts the probability of a mortgage approval in the 30 days following a page view as a function of total assets and whether a customer has used an iPad.

```
formula.str <- paste('mortgage_application_approved__n30d ~',
  'total_assets__AP + web_has_browsed_on_ipad__AP')

# Create a CausataData object to track preprocessing steps.
causataData <- CausataData(df, 'mortgage_application_approved__n30d')

# Remove missing values.
causataData <- CleanNaFromContinuous(causataData)
causataData <- CleanNaFromFactor(causataData)

# build a model
modelmatrix <- model.matrix(formula(formula.str), data=causataData$df)
cv.glmnet.obj <- cv.glmnet(modelmatrix,
  causataData$df$mortgage_application_approved__n30d,
  alpha=0.0, family='binomial')
```

Last, the model is assigned a variable name and label, and the model is uploaded to Causata.

```
# upload model
model.def <- ModelDefinition(
  cv.glmnet.obj, causataData, formula(formula.str),
  cv.glmnet.obj$lambda.min)
variable.def <- VariableDefinition(
  name='score-mortgage-application',
  display.name='Score: Mortgage Application',
  description='Logistic regression model for demo.', author='Justin',
  labels='Scores')
result <- UploadModel(CausataConfig(group='fsdemo'), model.def,
  variable.def)
```

2 A longer example

The next example builds on the first one, adding several steps. The R code shown below is all executable provided that the Causata R package is installed.

2.1 Extract and load data

The first step in a model building process is to extract and load model training data. In Causata the easiest way to extract data is to use the Causata-SQL interface, which requires a connection to a Causata server. This section of the vignette is intended to work without a connection, so an example data set from the Causata package is used instead of the SQL interface. The example data set includes records for individuals who were shown a mobile banner ad.

First packages are loaded, along with the example data. The data resides in a data frame `df.causata`.

```
library(Causata)
library(glmnet)
library(pROC)

# Set a random seed so random numbers are repeated
set.seed(87352)

# Load example data from the Causata package
data(df.causata)
```

Next a dependent variable is selected. This example uses binary classification, so the dependent variable has only two possible values indicating of a customer has or has not clicked an ad for a mobile banking product.

```
# Set the dependent variable
dvname <- "has.responded.mobile.logoff_next.hour_466"
```

2.2 Preprocessing

Next the data set is divided into 75% training and 25% test.

```
# Create an index of training data.
idx.train <- sample.int(nrow(df.causata), round(nrow(df.causata)*0.75))
# Split data into training and testing data frames by row.
df.train <- df.causata[ idx.train, ]
df.test  <- df.causata[-idx.train, ]
```

Some of the variables may not be valid predictors (all missing values, single value, etc.), or may have very weak predictive power. The `BinaryPredictor` function is used to quickly assess the univariate predictive power and robustness for each variable.

```
# Run a univariate analysis for each variable in the data frame
binaryPredictorSummary <- BinaryPredictor(df.train, df.train[[dvname]],
  min.robustness = 0.1)

# Generate summary information and store it in a data frame.
df.summary <- print(binaryPredictorSummary, silent=TRUE)
```

Next a `CausataData` object is created, which tracks preprocessing steps that will be repeated during model scoring. Only variables that pass the `BinaryPredictor` screening are used.

```
# Create a CausataData object with training data.
causataData <- CausataData(df.train[, df.summary$keep], dvname)
```

2.2.1 Linearization and outliers

Model accuracy may be improved by linearizing variables with a transformation such as the Weight of Evidence (WOE). Causata supports the discretization of continuous variables into bins. Here the continuous data is discretized and mapped to WOE values, which linearizes the response.

```
# Loop for each variable, apply discretization to nonlinear variables
for (varname in names(causataData$df)){
  if (varname %in% c(dvname)){next} # skip the dependent variable
  # Get summary information for this variable
  idx <- which(varname == df.summary$name)
  # Discretize nonlinear variables where the predictive power is > 0.3
  # and the linearity is < 0.9
  if (df.summary$class[idx] %in% c('integer','numeric') &
      df.summary$predictivePower[idx] > 0.03 &
      abs(df.summary$linearity[idx]) < 0.9){
    # Set outlier limits before discretizing
    causataData <- ReplaceOutliers(causataData, varname,
      lowerLimit=min(causataData$df[[varname]], na.rm=TRUE),
      upperLimit=max(causataData$df[[varname]], na.rm=TRUE))
    # Find breakpoints
    bclist <- BinaryCut(causataData$df[[varname]],
      causataData$df[[dvname]], minBin=0, bins=TRUE)
    # Compute weight of evidence
    woe <- Woe(bclist$fiv, causataData$df[[dvname]])
    # Discretize
    causataData <- Discretize(causataData, varname, bclist$breaks,
      woe$woe.levels)
  }
}
```

This step increases the model predictive power by approximately 3%. The benefit will be proportional to the amount of nonlinearity in the training data.

Note that outlier limits are set using the `ReplaceOutliers` function. This step is required before running `Discretize`. If new data has values outside the min / max of the training data, then this step will replace the outliers with the min / max values of the training data.

2.2.2 Missing values

Two functions from the Causata R package are used to replace missing values in factors (categorical variables) and in numeric variables. The replacement values are stored in `causataData`.

```
# Replace missing values
causataData <- CleanNaFromContinuous(causataData)
causataData <- CleanNaFromFactor(causataData)
```

2.2.3 Categorical variables

Categorical variables with many levels will generate many dummy variables, which can be problematic during model training. One way to mitigate this problem is to merge smaller levels into an "Other" level. In the example below, the number of levels is capped at 10, and the mapping of original levels to the new levels is stored in `causataData`.

```
# Merge levels in factors with # levels exceeding a threshold
causataData <- MergeLevels(causataData, max.levels=10)
```

2.2.4 Other preprocessing steps

Other preprocessing steps may include:

- **Removing colinear variables:** Highly correlated variables can make model interpretation and variable importance calculations more difficult. If multiple variables are correlated then selecting only one for the model may be beneficial.
- **Interaction terms:** Interaction terms are supported during scoring. They can be added to the model formula in R and will automatically be translated into the Causata model.

2.3 Train a glmnet model

Next the `model.matrix` function will be used to convert the data frame into a design matrix where factors are encoded as dummy variables. This step requires a model formula, which is simply the addition of all independent variables in the data frame.

```
# Extract a set of independent variable names, exclude the dependent
# variable.
indep.vars <- colnames(causataData$df)
indep.vars <- indep.vars[!(indep.vars == dvname) ]
# Build a formula string
formula.string <- paste(dvname, "~", paste(indep.vars, collapse=" + "))
```

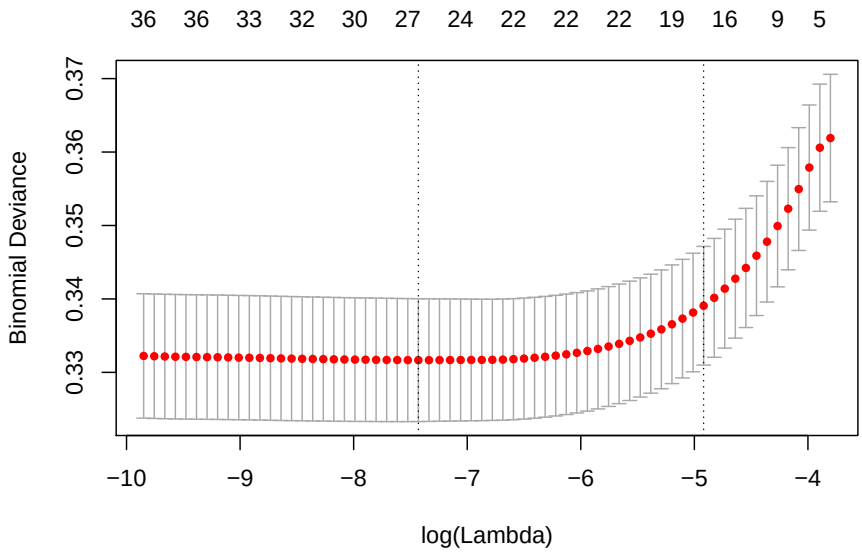
```
formula.object <- formula(formula.string)
# Build a model matrix
x.matrix.train <- model.matrix(formula.object, data=causataData$df)
# Set the dependent variable
y.train <- causataData$df[[dvname]]
y.test  <- df.test[[dvname]]
```

The preprocessing steps applied to the training data must be re-applied to the testing data. The `GetTransforms` function makes this process easy.

```
# Get a function that applies transformations to a data frame.
Transformer <- GetTransforms(causataData)
# Apply the transformations to the test data.
df.test.transformed <- Transformer(df.test)
```

Next the training data is supplied to `cv.glmnet` (from the `glmnet` package), which builds multiple models while varying the regularization parameter Lambda. The `plot` command generates a diagnostic plot of the model error (deviance) with respect to Lambda. The error bars indicate the standard error across the ten-fold cross validation, which appears to be high with respect to the overall change in the mean deviance. The standard error could be reduced by using a larger data set or reducing multicolinearity in the training data.

```
# Build model, select alpha value near 1
# since we expect most coefficients to be zero
cv.glmnet.obj <- cv.glmnet(x.matrix.train, y.train, alpha=0.8,
  family=c("binomial"))
plot(cv.glmnet.obj)
```



Now the model is applied to the training and test data. Predicted values are calculated for each data set.

```
# Use the model to predict responses for training data
predicted.train <- predict(cv.glmnet.obj, newx=x.matrix.train,
  type="response", s="lambda.min")

# Compute predictions for test data.
model.def <- ModelDefinition(cv.glmnet.obj, causataData,
  formula.object, cv.glmnet.obj$lambda.min )
predict.result <- predict(model.def, df.test.transformed)
predicted.test <- predict.result$predicted
```

The next step is to assess the model accuracy using the area under the ROC curve. A value of 0.72 for the test data is considered fair, though not strong. Ideally the area under the ROC curve will be very close for the train / test data sets, which indicates a robust model that is not over fit. In this example the areas are within 1%.

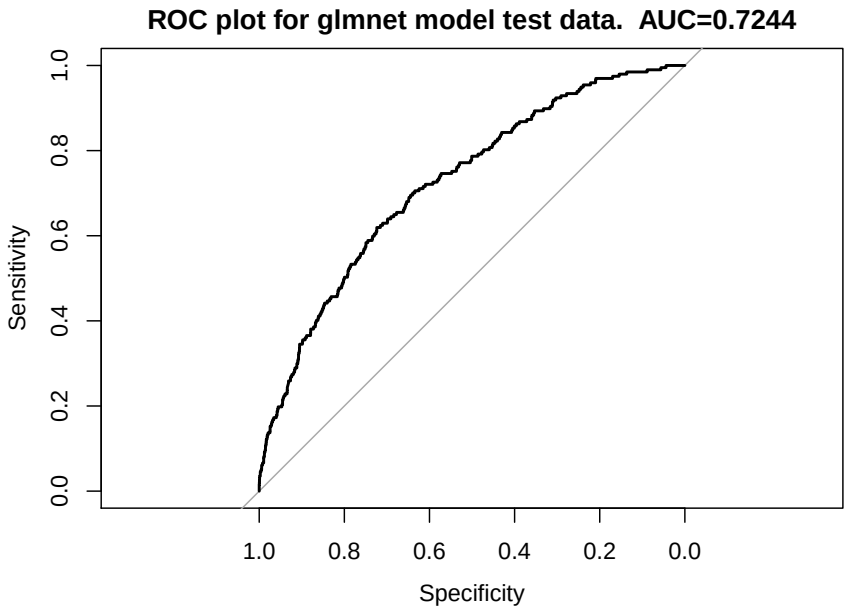
```
# Compute area under the ROC curve using the pROC package
roc.train <- roc(y.train, predicted.train)

## Warning in roc.default(y.train, predicted.train):  Deprecated use a matrix
as predictor.  Unexpected results may be produced, please pass a numeric vector.

roc.test  <- roc(y.test,  predicted.test )
cat("Training / testing area under ROC curve: ",
  roc.train$auc, ", ", roc.test$auc, "\n")
```

```
## Training / testing area under ROC curve:  0.7277744 ,  0.7243998

plot(roc.test,
  main=sprintf("ROC plot for glmnet model test data.  AUC=%6.4f",
  roc.test$auc))
```




```
##
## Call:
## roc.default(response = y.test, predictor = predicted.test)
##
## Data: predicted.test in 4029 controls (y.test false) < 197 cases (y.test true).
## Area under the curve: 0.7244
```

In the final steps the model name and description are set, and the Causata server parameters are provided so that the model can be uploaded.

```
# Prepare to import the model into Causata
variable.def <- VariableDefinition(
  name = "score-test-model",
  display.name = "Score: Test Model",
  description = "A logistic regression model.",
  author = "Test User" )

# Generate a string of PMML representing the model and
# preprocessing transformations.
# This step is not required to upload a model,
# it's shown for illustration purposes only
```

```
pmml.string <- ToPmml(model.def, variable.def)

# Upload model to Causata.
# The parameters below are for illustration only.
causata.config <- CausataConfig(
  config.server.host = "123.456.789.10",
  config.server.port = 8002,
  config.username = "testuser",
  config.password = "1234abcd")
```

The final command `UploadModel` requires a live connection to a Causata server. The command creates a new variable in Causata that calculates the model score.

```
result <- UploadModel(causata.config, model.def, variable.def)
```

The model coefficients exported to Causata will match the set obtained from the `cv.glmnet` object. This code extracts the nonzero coefficients.

```
# extract nonzero coefficients
coefs.all <- as.matrix(coef(cv.glmnet.obj, s="lambda.min"))
idx <- as.vector(abs(coefs.all) > 0)
coefs.nonzero <- as.matrix(coefs.all[idx])
rownames(coefs.nonzero) <- rownames(coefs.all)[idx]
```

The model coefficients are printed below.

```
print(coefs.nonzero)
```

```
##                                                                  [,1]
## (Intercept)                                                2.4483420610
## decisions.number.of.decisions_all.past_2                   0.5098600804
## has.decision.logout.prompt_all.past_349true               -0.0585343920
## has.decision.logout.prompt_last.30.days_349true           -0.3659227222
## has.responded.mobile.logoff_all.past_466true               1.0768969209
## has.responded.mobile.logoff_last.30.days_466true           1.0101260963
## ib.has.enrollment.event_all.past_145true                  -0.5889689673
## ib.has.forgotten.password_last.30.days_417true             0.7293430759
## ib.has.forgotten.password_last.7.days_417true             -0.0211965283
## impression.count.credit.cards.logoff_all.past_355          0.0600554752
## impression.count.mobile.logoff_last.30.days_361            0.1881350211
## impression.count.mobile.logoff_last.7.days_361            -0.0006321068
## mobile.has.any.mobile.app.activity_all.past_324true       -0.3504908227
## mobile.has.used.sms_last.30.days_43true                    0.4002729734
## online.average.authentications.per.month_all.past_406     -0.0015968151
## online.average.authentications.per.month_last.30.days_406 -0.0712628090
## online.has.online.activity_last.7.days_49true              2.5331785781
## online.has.used.mobile.device_all.past_418true             0.4222523359
## online.has.used.mobile.device_last.7.days_418true          0.6282495349
## online.number.of.page.views_all.past_3                     0.5834574367
## online.number.of.page.views_last.30.days_3                -0.2232070140
## online.number.of.page.views_last.7.days_3                  0.4031460330
## online.product.category.view.count.credit.lending_last.7.days_446 -0.6019162922
## online.product.category.view.count.mobile.banking_last.30.days_447  0.0843627369
## online.time.since.last.authentication_focal.point_408      0.2256956795
## os.os.disclosure.accepted_last.30.days_388true             1.2560973910
## time.since.last.authentication_focal.point_472             0.2181061121
## time.since.last.session_focal.point_404                    0.4721834215
```