6. Model Evaluation 2

2022-06-25

Contents

```r
# Load caTools package for data partitioning
library(caTools)

# load Caret package for computing Confusion matrix
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```r
# Packages for SVM, Random Forest and GBM
library(e1071)
library(randomForest)
```

```
## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(gbm)
```

```
## Loaded gbm 2.1.8
```

```
library(readr)

# Import churndata.csv
churndata <- read_csv("/Users/kuchunyeh/Documents/WBS term 1/AIP Seminars(Analytics in Practice)/churnd
```

```
## Rows: 19239 Columns: 13

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (13): ID, COLLEGE, INCOME, AVERAGE_OVERCHARGE, LEFTOVER, HOUSE, HANDSET_...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# Display the structure of the dataset
str(churndata)
```

```
## spec_tbl_df [19,239 x 13] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ ID                         : num [1:19239] 1 2 3 4 5 6 7 8 9 10 ...
##  $ COLLEGE                    : num [1:19239] 0 1 1 0 1 0 0 1 0 0 ...
##  $ INCOME                     : num [1:19239] 31953 36147 27273 120070 29215 ...
##  $ AVERAGE_OVERCHARGE         : num [1:19239] 0 0 230 38 208 64 224 0 0 174 ...
##  $ LEFTOVER                   : num [1:19239] 6 13 0 33 85 48 0 20 7 18 ...
##  $ HOUSE                      : num [1:19239] 313378 800586 305049 788235 224784 ...
##  $ HANDSET_PRICE              : num [1:19239] 161 244 201 780 241 626 191 357 190 687 ...
##  $ OVER_15MINS_CALLS_PER_MONTH: num [1:19239] 0 0 16 3 21 3 10 0 0 25 ...
##  $ AVERAGE_CALL_DURATION      : num [1:19239] 4 6 15 2 1 2 5 5 5 4 ...
##  $ REPORTED_SATISFACTION      : num [1:19239] 2 2 2 2 1 2 1 1 5 5 ...
##  $ REPORTED_USAGE_LEVEL       : num [1:19239] 2 2 1 5 2 4 2 2 2 2 ...
##  $ CONSIDERING_CHANGE_OF_PLAN : num [1:19239] 1 4 3 4 2 1 5 4 5 2 ...
##  $ CLASS                      : num [1:19239] 0 0 0 1 0 0 0 0 0 1 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   ID = col_double(),
##   ..   COLLEGE = col_double(),
##   ..   INCOME = col_double(),
##   ..   AVERAGE_OVERCHARGE = col_double(),
##   ..   LEFTOVER = col_double(),
##   ..   HOUSE = col_double(),
##   ..   HANDSET_PRICE = col_double(),
##   ..   OVER_15MINS_CALLS_PER_MONTH = col_double(),
##   ..   AVERAGE_CALL_DURATION = col_double(),
##   ..   REPORTED_SATISFACTION = col_double(),
##   ..   REPORTED_USAGE_LEVEL = col_double(),
##   ..   CONSIDERING_CHANGE_OF_PLAN = col_double(),
##   ..   CLASS = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

```
# Remove customer Id column
churndata$ID <- NULL

# Update the data type of target variable if necessary
churndata$CLASS <- as.factor(churndata$CLASS)
```

remove the customer "ID" column, since it does not have any effect on the target variable.

```
# Check the levels of target variable
levels(churndata$CLASS)
```

need to make sure that "1" is identified as the positive class since our focus is on churn customers.

```
## [1] "0" "1"
```

```
# Set seed to 1
set.seed(1)

# Generate training and test sets
split = sample.split(churndata$CLASS, SplitRatio = 0.6)

trainingdata = subset(churndata, split == TRUE)
testdata = subset(churndata, split == FALSE)
```

Partition the dataset into training (60%) and test (40%) sets. Do not forget to set the seed for random variables. Load caTools package for data partitioning.

SVM

```
# Build SVM model and assign it to model_SVM
SVM_model <- svm(CLASS ~. , data=trainingdata, kernel= "radial", scale = TRUE, probability = TRUE)
```

build an SVM model by using all features. Note that by setting probability = TRUE in svm() function, the class probabilities and the predicted classes of the target variable can be obtained.

```
# predict(SVM model, test data, probability = TRUE)
```

```r
SVM_pred  <- predict(SVM_model, testdata, probability = TRUE)
```

```r
# Use confusionMatrix to print the performance of SVM model
confusionMatrix(SVM_pred, testdata$CLASS, positive = "1", mode = "prec_recall")
```

Predict the class of the test data.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2587 1188
##          1 1247 2673
##
##                Accuracy : 0.6836
##                  95% CI : (0.673, 0.6939)
##     No Information Rate : 0.5018
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.3671
##
##  Mcnemar's Test P-Value : 0.2398
##
##               Precision : 0.6819
##                  Recall : 0.6923
##                      F1 : 0.6871
##              Prevalence : 0.5018
##          Detection Rate : 0.3474
##    Detection Prevalence : 0.5094
##       Balanced Accuracy : 0.6835
##
##        'Positive' Class : 1
##
```

Random Forest

```r
# Set random seed
set.seed(10)

# Build Random Forest model and assign it to RF_model
RF_model <- randomForest(CLASS ~. , trainingdata, ntree = 800)
```

Follow the same steps as in the SVM model and build a Random Forest model. Set ntree= 800.

Predict the class of the test data and store the result as RF_pred and check the confusion matrix.

```r
# Predict the class of the test data
RF_pred <- predict(RF_model, testdata)

# Confusion matrix
confusionMatrix(RF_pred, testdata$CLASS, positive='1', mode = "prec_recall")
```

Use confusionMatrix() function to print model performance results.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2430  942
##          1 1404 2919
##
##                Accuracy : 0.6951
##                  95% CI : (0.6847, 0.7054)
##     No Information Rate : 0.5018
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.39
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##               Precision : 0.6752
##                  Recall : 0.7560
##                      F1 : 0.7133
##              Prevalence : 0.5018
##          Detection Rate : 0.3793
##    Detection Prevalence : 0.5618
##       Balanced Accuracy : 0.6949
##
##        'Positive' Class : 1
##
```

Gradient Boosting Machin

Gradient Boosting Machine (GBM) model

```r
# gbm(formula, data, distribution = "bernoulli", cv.folds = ...)
```

GBM is one of the most powerful techniques for building predictive models. It is an ensemble learning technique used for both classification and regression problems. The main idea of this method is to improve the decision trees sequentially and increase the model accuracy with a combined model.

Formula: shows which features are used in modelling to predict the target variable.

Data: the dataset that will be used for model building.

For classification problems with two classes, we set distribution = “bernoulli”. If the target variable has more than two classes, we set distribution = “multinomial”.

cv.folds: number of folds for cross-validation.

```
# Change the data type of the target variable
trainingdata$CLASS <- as.numeric(trainingdata$CLASS)-1
```

Since we have a binary classification problem, we set distribution = “bernoulli” for this case. Note that when distribution = “bernoulli”, GBM requires target variable as a numeric data type. Therefore, we first change the data type of the target variable to numeric.

The performance of GBM can be significantly improved with hyperparameter tuning.

n.trees: The total number of trees in the ensemble. The default value is 100. In this task, we will set n.trees = 500.

interaction.depth: The depth of the individual trees. Typical values range from 3 to 8. The default value is 1. In this task, we will set interaction.depth = 3.

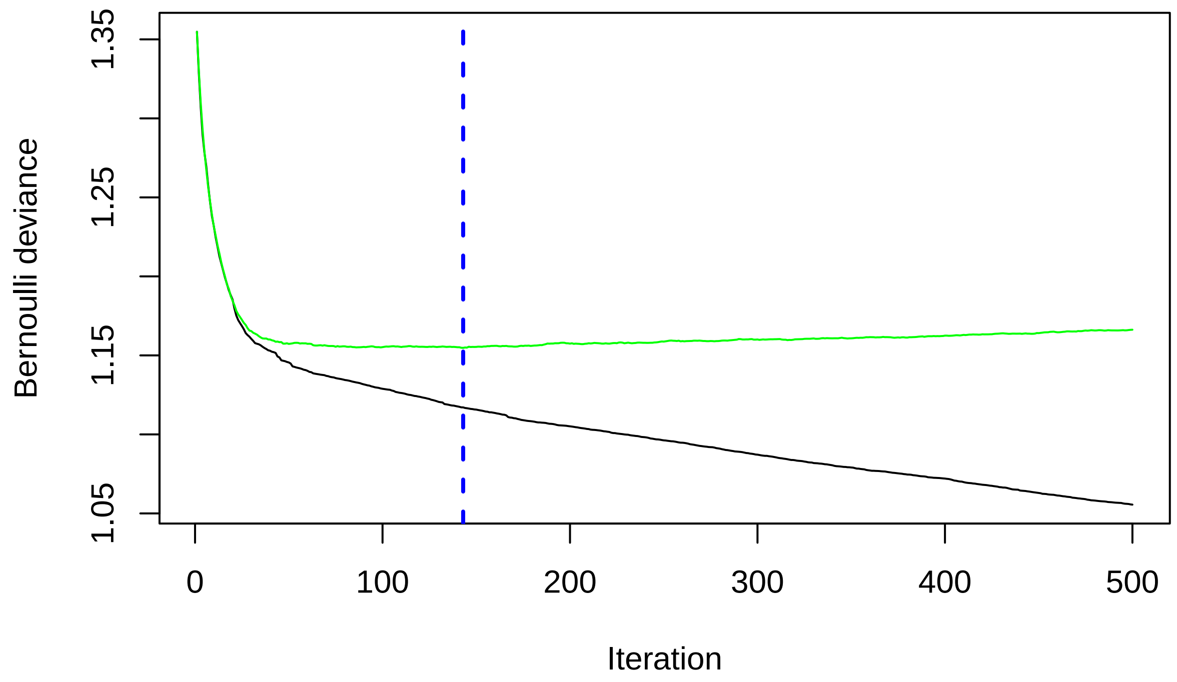
```
# Set random seed
set.seed(10)

# Build the GBM model
GBM_model <- gbm(CLASS ~. , trainingdata, distribution = "bernoulli", n.trees=500, interaction.depth=3,
```

Set the number of folds in K-fold cross validation to 5.

```
# Find the number of trees for the prediction
ntree_opt <- gbm.perf(GBM_model, method = "cv")
```

GBM model stops building decision trees when the number of trees reach to the limit defined by n.trees, which is 500 for this case. However, using all trees to make a prediction on the test data may deteriorate the performance of the model due to `overfitting`**.**

We can use gbm.perf function to find the best number of trees to use for prediction.



make predictions on the test data

GBM returns the probability that a target variable belongs to a particular class. Therefore, we use type="response" argument to obtain these values.

```
# Obtain prediction probabilities using ntree_opt
GBM_prob <-  predict(GBM_model, testdata, n.trees = 150, type = "response")

# Make predictions with threshold value 0.5
GBM_pred <- ifelse(GBM_prob >= 0.5, "1", "0")

# Save the predictions as a factor variable
GBM_pred <- as.factor(GBM_pred)

# Confusion matrix
confusionMatrix(GBM_pred, testdata$CLASS, positive='1', mode = "prec_recall")
```

GBM_prob will keep the class scores (or probabilities). In order to predict the class of a test data, we use default threshold value. If the probability of a record is greater than or equal to 0.5, it will be marked as churn "1", otherwise it will be marked as stay "0". We need to save these predictions as factor variable.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2409  853
##          1 1425 3008
##
##                Accuracy : 0.704
##                  95% CI : (0.6936, 0.7141)
##     No Information Rate : 0.5018
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4076
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##               Precision : 0.6785
##                  Recall : 0.7791
##                      F1 : 0.7253
##              Prevalence : 0.5018
##          Detection Rate : 0.3909
##    Detection Prevalence : 0.5761
##       Balanced Accuracy : 0.7037
##
##        'Positive' Class : 1
##
```

Model Evaluation Visualisation

visualise the performances of SVM, Random Forest and GBM by using ROC and Gain charts.

To plot these charts, pROC package should be loaded.

roc()

```
# roc(testset$target, probabilities)
```

roc() takes two arguments; predicted class probabilities (likelihood of belonging to a class) and actual values of the test data.

```
#load the ROCR package
#install.packages("pROC")
library(pROC)
```

load the pROC package.

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

Obtain class probabilities (likelihood of belonging to a class) for SVM and Random Forest models built in task 1.

Class probabilities of GBM are stored in GBM_prob. Therefore, we only need to extract probabilities predicted by SVM and Random Forest.

```
# Provide probabilities and generate input data
# SVM
ROC_SVM <- roc(testdata$CLASS, RF_prob[,2])
```

Use roc() function to generate input data for the ROC curve of these models.

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```

```
# Random Forest
ROC_RF <- roc(testdata$CLASS, SVM_prob[,2])
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
# GBM
ROC_GBM <- roc(testdata$CLASS, GBM_prob)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

Extract True Positive Rate (Sensitivities) and False Positive Rate (1-Specificities) for plotting.

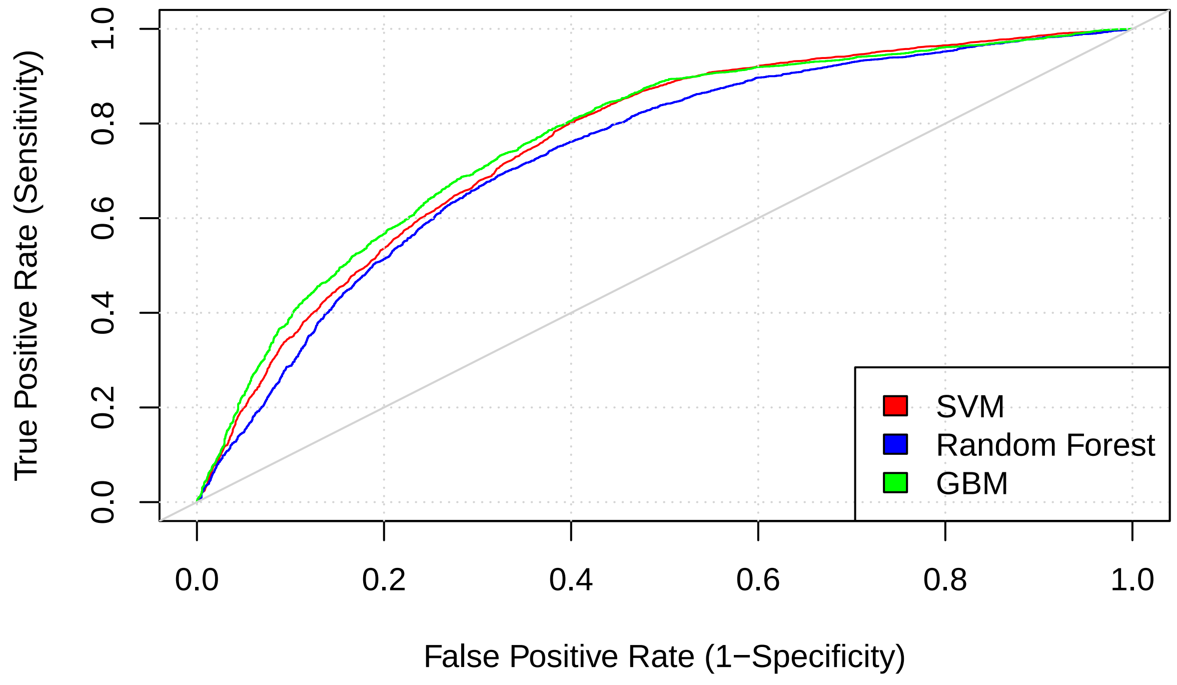
The true positive rate (TPR, also called sensitivity) = TP/TP+FN

```
# Extract required data from ROC_SVM
df_SVM = data.frame((1-ROC_SVM$specificities), ROC_SVM$sensitivities)

# Extract required data from ROC_RF
df_RF = data.frame((1-ROC_RF$specificities), ROC_RF$sensitivities)

# Extract required data from ROC_GBM
df_GBM = data.frame((1-ROC_GBM$specificities), ROC_GBM$sensitivities)
```

The false positive rate (FPR) = FP/FP+TN

```
#plot the ROC curve for Random Forest, SVM and GBM

plot(df_SVM, col="red", type="l",
xlab="False Positive Rate (1-Specificity)", ylab="True Positive Rate (Sensitivity)")
lines(df_RF, col="blue")                #adds ROC curve for RF
lines(df_GBM, col="green")              #adds ROC curve for GBM
grid(NULL, lwd = 1)

abline(a = 0, b = 1, col = "lightgray") #adds a diagonal line

legend("bottomright",
c("SVM", "Random Forest", "GBM"),
fill=c("red","blue", "green"))
```

Plot the ROC curve for SVM, RF and GBM.



auc()

```
#Calculate the area under the curve (AUC) for SVM
auc(ROC_SVM)
```

Compute AUC values for these models by using auc() function. "roc" object obtained from roc() function can be used here to compute AUC value.

```
## Area under the curve: 0.759
```

```
#Calculate the area under the curve (AUC) for Random Forest
auc(ROC_RF)
```

```
## Area under the curve: 0.7354
```

```
#Calculate the area under the curve (AUC) for GBM
auc(ROC_GBM)
```

```
## Area under the curve: 0.7685
```

```r
#load the CustomerScoringMetrics package
#install.packages("CustomerScoringMetrics")
library(CustomerScoringMetrics)
```

Next, Cumulative Response (Gain) chart will be plotted for these models. Need to install and load CustomerScoringMetrics package.

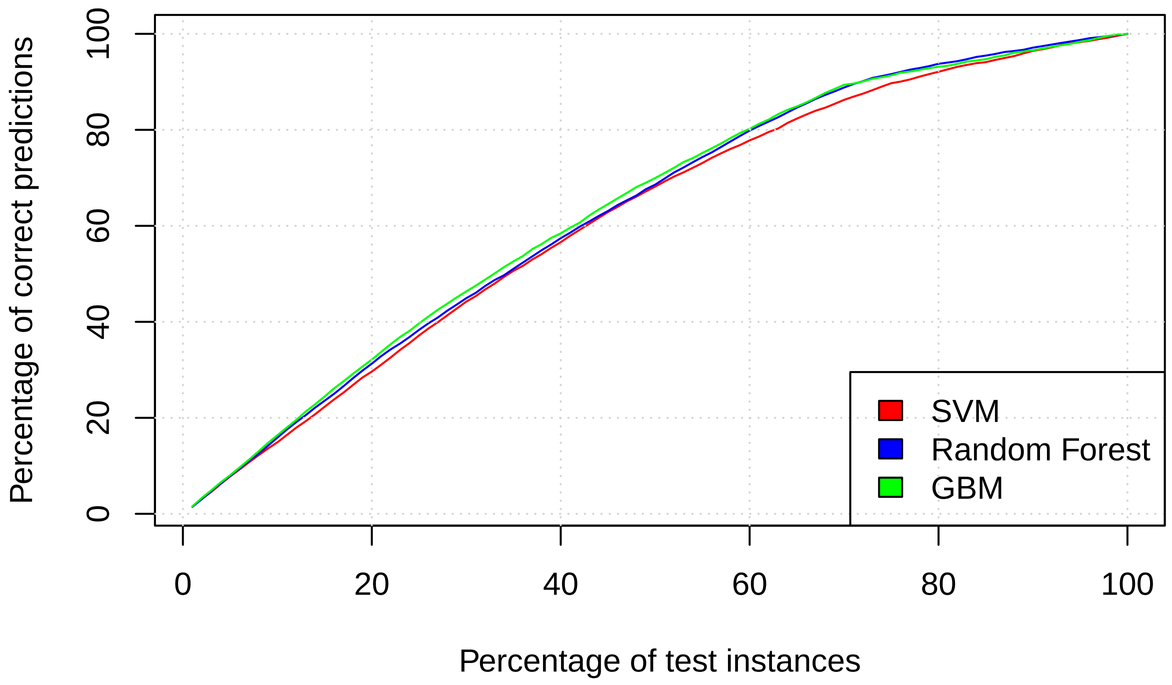
```r
# cumGainsTable(probabilities, actual value of the target variable, resolution)
```

Specifically, use cumGainsTable() function to calculate cumulative gain values for our chart. This function takes three arguments. The first one is the prediction probabilities (scores), the second one is the actual values of the target variables and the third one is the increment of the threshold value.

```r
# Provide probabilities for the outcome of interest and obtain the gain chart data

GainTable_SVM <- cumGainsTable(SVM_prob[,2], testdata$CLASS, resolution = 1/100)

GainTable_RF <- cumGainsTable(RF_prob[,2], testdata$CLASS, resolution = 1/100)

GainTable_GBM <- cumGainsTable(GBM_prob, testdata$CLASS, resolution = 1/100)
```

```r
plot(GainTable_SVM[,4], col="red", type="l",
xlab="Percentage of test instances", ylab="Percentage of correct predictions")
lines(GainTable_RF[,4], col="blue", type ="l")
lines(GainTable_GBM[,4], col="green", type ="l")
grid(NULL, lwd = 1)

legend("bottomright",
c("SVM", "Random Forest", "GBM"),
fill=c("red","blue", "green"))
```

Plot the gain chart with increment of 1/100.



Percentage of correct predictions
0
20
40
60
80
100
SVM
Random Forest
GBM
0
20
40
60
80
100
Percentage of test instances