5. Model Evaluation 1

2022-06-20

Contents

prepare data

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

# Load e1071 package for svm
library("e1071")

# Assign cancerdata.csv to bcdata
bcdata <- read.csv("cancerdata.csv")

# Check the structure of the dataset
str(bcdata)
```

```
## 'data.frame':    643 obs. of  32 variables:
##  $ id                     : num  8670 8913 8915 9047 85715 ...
##  $ diagnosis              : int  1 0 0 0 1 1 0 1 0 0 ...
##  $ radius_mean            : num  15.5 12.9 15 12.9 13.2 ...
##  $ texture_mean           : num  19.5 13.1 19.1 16.2 18.7 ...
##  $ perimeter_mean         : num  101.7 81.9 97 83.2 86 ...
##  $ area_mean              : num  749 516 687 508 535 ...
```

```
##  $ smoothness_mean        : num  0.1092 0.0696 0.0899 0.0988 0.1158 ...
##  $ compactness_mean       : num  0.1223 0.0373 0.0982 0.0884 0.1231 ...
##  $ concavity_mean         : num  0.1466 0.0226 0.0594 0.033 0.1226 ...
##  $ concave.points_mean    : num  0.0809 0.0117 0.0482 0.0239 0.0734 ...
##  $ symmetry_mean          : num  0.193 0.134 0.188 0.173 0.213 ...
##  $ fractal_dimension_mean : num  0.058 0.0558 0.0585 0.062 0.0678 ...
##  $ radius_se              : num  0.474 0.153 0.288 0.146 0.287 ...
##  $ texture_se             : num  0.786 0.469 0.948 0.905 0.894 ...
##  $ perimeter_se           : num  3.094 1.115 2.171 0.998 1.897 ...
##  $ area_se                : num  48.3 12.7 24.9 11.4 24.2 ...
##  $ smoothness_se          : num  0.00624 0.00473 0.00533 0.00289 0.00653 ...
##  $ compactness_se         : num  0.0148 0.0135 0.0211 0.0129 0.0234 ...
##  $ concavity_se           : num  0.0281 0.0165 0.0154 0.0161 0.029 ...
##  $ concave.points_se      : num  0.01093 0.0059 0.01187 0.00731 0.01215 ...
##  $ symmetry_se            : num  0.014 0.0162 0.0152 0.0187 0.0174 ...
##  $ fractal_dimension_se   : num  0.00246 0.00208 0.00281 0.00197 0.00364 ...
##  $ radius_worst           : num  19.3 13.6 16.2 13.9 15.7 ...
##  $ texture_worst          : num  26 15.5 26.2 23 27.9 ...
##  $ perimeter_worst        : num  124.9 87.4 109.1 89.7 102.8 ...
##  $ area_worst             : num  1156 577 810 581 759 ...
##  $ smoothness_worst       : num  0.1546 0.0962 0.1313 0.1172 0.1786 ...
##  $ compactness_worst      : num  0.239 0.115 0.303 0.196 0.417 ...
##  $ concavity_worst        : num  0.379 0.119 0.18 0.181 0.501 ...
##  $ concave.points_worst   : num  0.1514 0.0537 0.1489 0.0839 0.2088 ...
##  $ symmetry_worst         : num  0.284 0.231 0.296 0.33 0.39 ...
##  $ fractal_dimension_worst: num  0.0802 0.0692 0.0847 0.0783 0.1179 ...
```

Remove patient "id" column. It does not affect the target variable.

```
# Remove patient id
bcdata$id <- NULL

# Update the data type if necessary
bcdata$diagnosis <- as.factor(bcdata$diagnosis)
```

Update the data type of the target variable if necessary.

```
# Set seed to 123
set.seed(123)

# Partition the data
split <- sample.split(bcdata$diagnosis, SplitRatio = 0.80)

# Generate training and test sets and save as trainingset and testset
training <- subset(bcdata, split == TRUE)
test <- subset(bcdata, split == FALSE)
```

Partition the dataset into training (80%) and test (20%) sets.

Random Forest

```
# install.packages("randomForest)

# Load randomForest package
library(randomForest)
```

Random Forest consists of a large number of individual decision trees that operate as a group. It is a popular ensemble method that can be used to build predictive models for both classification and regression problems.

```
## randomForest 4.6-14
```

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

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

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

Build and print our Random Forest model.

```
# randomForest(formula, data)
```

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

# Build Random Forest model and assign it to model_RF
model_RF <- randomForest(diagnosis ~., training, mtry= 7, nodesize = 7, sampsize= 257)

# Print model_RF
print(model_RF)
```

Print the importance weights of attributes by using importance(modelname) function from this package.

```
##
## Call:
##  randomForest(formula = diagnosis ~ ., data = training, mtry = 7,      nodesize = 7, sampsize = 257)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 18.68%
## Confusion matrix:
```

```
##     0   1 class.error
## 0 300  55   0.1549296
## 1  41 118   0.2578616
```

```
# Check the important attributes by using importance() function
importance(model_RF)
```

```
##                         MeanDecreaseGini
## radius_mean                    5.0963822
## texture_mean                   2.9789913
## perimeter_mean                 4.2797390
## area_mean                      3.3398604
## smoothness_mean                1.1327607
## compactness_mean               1.0711131
## concavity_mean                 3.4815753
## concave.points_mean            7.8545189
## symmetry_mean                  1.3579859
## fractal_dimension_mean         0.8778948
## radius_se                      1.3662826
## texture_se                     1.0880019
## perimeter_se                   1.4155119
## area_se                        3.4999799
## smoothness_se                  0.9380653
## compactness_se                 1.0085534
## concavity_se                   1.1579855
## concave.points_se              1.0059969
## symmetry_se                    1.2478523
## fractal_dimension_se           1.1405312
## radius_worst                   9.2946081
## texture_worst                  2.9795269
## perimeter_worst               10.9372218
## area_worst                    10.0154208
## smoothness_worst               1.3185792
## compactness_worst              1.0822794
## concavity_worst                2.7436494
## concave.points_worst           8.0484159
## symmetry_worst                 1.6381712
## fractal_dimension_worst        0.9073238
```

mtry: the number of predictors to sample at each split. The default value for a classification problem is given as the square root of total number of the features in the training data.

nodesize: the minimum number of instances in a terminal node. The default value for a classification problem is 1.

sampsize: the size of sample to draw. The default value for sample size is 2/3 of the training data.

```r
# List of possible values for mtry, nodesize and sampsize
mtry_val <- seq(3, 7, 2)
nodesize_val <- seq(1, 10, 2)
sampsize_val <- floor(nrow(training)*c(0.5, 0.65, 0.8)) #sampling training set's 50, 65, 80 percent of
# floor(): return the largest integer value that is not greater than (less than) or equal to a specific
```

sampsize: Balance unbalanced data set, e.g. unbalanced (71/900) -> sampsize=c(71,71) (needs to change the categorial data into factors)

expand.grid()

```r
# Create a data frame containing all combinations
setOfvalues <- expand.grid(mtry = mtry_val, nodesize = nodesize_val, sampsize = sampsize_val)

# Create an empty vector to store error values
err <- c()

# Write a  loop over the rows of setOfvalues to train random forest model for all possible values
for (i in 1:nrow(setOfvalues)){
    # Since random forest model uses random numbers set the seed
    set.seed(10)

    # Train a Random Forest model
    model <- randomForest(diagnosis~., training,
                          mtry = setOfvalues$mtry[i],
                          nodesize = setOfvalues$nodesize[i],
                          sampsize = setOfvalues$sampsize[i])

    # Store the error rate for the model
    err[i] <- model$err.rate[nrow(model$err.rate), "OOB"]
}

# Identify optimal set of hyperparameters based on error rate
best_comb <- which.min(err)
print(setOfvalues[best_comb,])
```

a data frame that stores all combinations

```
##    mtry nodesize sampsize
## 12    7        7      257
```

tuneRF()

```r
# Predict the class of the test data
prediction_RF <- predict(model_RF, test)
```

randomForest package have tuneRF() function for searching the best mtry values given for the data.

confusionMatrix() function

print performance metrics by confusionMatrix() function in caret package

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

confusionMatrix() function with three arguments; predicted target variable, actual target variable and the positive class. In order to set the positive class, we add positive='1' argument to this function.

```
## Loading required package: ggplot2

##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:randomForest':
##
##     margin

## Loading required package: lattice
```

```
# The last argument sets the positive class
confusionMatrix(prediction_RF, test$diagnosis, positive='1', mode = "prec_recall")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 67  5
##          1 22 35
##
##                Accuracy : 0.7907
##                  95% CI : (0.7103, 0.8573)
##     No Information Rate : 0.6899
##     P-Value [Acc > NIR] : 0.007150
##
##                   Kappa : 0.5621
##
##  Mcnemar's Test P-Value : 0.002076
##
##               Precision : 0.6140
##                  Recall : 0.8750
##                      F1 : 0.7216
##              Prevalence : 0.3101
##          Detection Rate : 0.2713
##    Detection Prevalence : 0.4419
##       Balanced Accuracy : 0.8139
##
##        'Positive' Class : 1
##
```

SVM

Load e1071 package to build SVM model.

Set kernel method as radial.

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

```
# Predict the class of the test data
prediction_SVM <- predict(model_SVM, test)

# Use confusionMatrix to print the performance of SVM model
confusionMatrix(prediction_SVM, test$diagnosis, positive='1', mode = "prec_recall")
```

Set probability argument to TRUE to obtain the class probabilities as well as the predicted classes of the target variable.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 66  3
##          1 23 37
##
##                Accuracy : 0.7984
##                  95% CI : (0.7188, 0.8639)
##     No Information Rate : 0.6899
##     P-Value [Acc > NIR] : 0.0039575
##
##                   Kappa : 0.5859
##
##  Mcnemar's Test P-Value : 0.0001944
##
##               Precision : 0.6167
##                  Recall : 0.9250
##                      F1 : 0.7400
##              Prevalence : 0.3101
##          Detection Rate : 0.2868
##    Detection Prevalence : 0.4651
##       Balanced Accuracy : 0.8333
##
##        'Positive' Class : 1
##
```

Load pROC package. Use roc() function to evaluate the results of the predictive models.

attr()

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

In order to extract probabilities for SVM, we use attr() function. This function takes two arguments; an object whose attributes are to be accessed and a string specifying which attribute is to be accessed. For SVM, the object is the output of predict() function and the string is "probabilities".

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

##
## Attaching package: 'pROC'

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

```
# Obtain class probabilities by using predict() and adding type = "prob" for Random Forest model_RF
prob_RF <- predict(model_RF, test, type = "prob")


# Add probability = TRUE for SVM; model_SVM
SVMpred <- predict(model_SVM, test, probability = TRUE)
str(SVMpred)
```

```
##  Factor w/ 2 levels "0","1": 1 2 2 1 2 1 1 1 1 2 ...
##  - attr(*, "names")= chr [1:129] "4" "8" "11" "12" ...
##  - attr(*, "probabilities")= num [1:129, 1:2] 0.0176 0.7346 0.7977 0.0265 0.6649 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:129] "4" "8" "11" "12" ...
##   .. ..$ : chr [1:2] "1" "0"
```

```
# Obtain predicted probabilities for SVM
prob_SVM <- attr(SVMpred, "probabilities")
```

roc()

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

```
# Use roc function to return some performance metrics
# Random Forest
ROC_RF <- roc(test$diagnosis, prob_RF[,2])
```

roc() returns a "roc" object which will be used to plot ROC curve.

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

```
## Setting direction: controls < cases
```

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

True Positive rate (TP = Sensitivity)

```
# Extract required data from ROC_RF
# ROC: x -> false positive rate; y -> true positive rate
df_RF = data.frame((1-ROC_RF$specificities), ROC_RF$sensitivities)
```

```
# Use roc function to return some performance metrics
# SVM
ROC_SVM <- roc(test$diagnosis, prob_SVM[,1])
```

False Positive rate (FP = 1 - Specificity).

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

```
## Setting direction: controls < cases
```

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

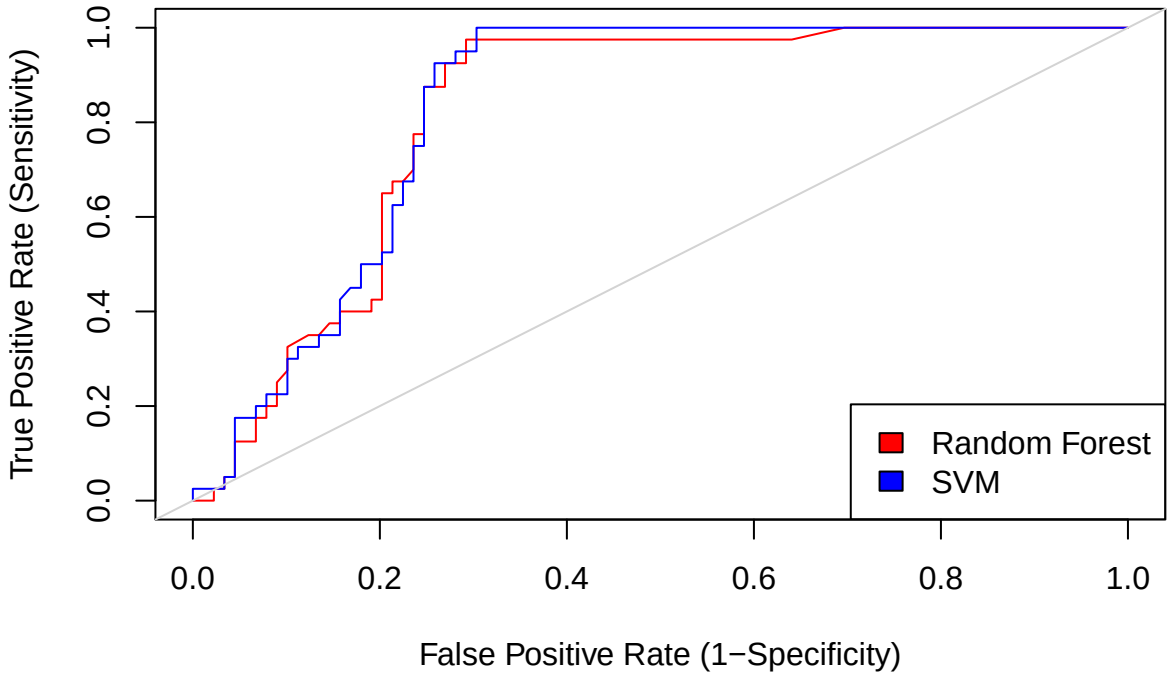
Visualisation

```
# Plot the ROC curve for Random Forest and SVM

plot(df_RF, col="red",
        type="l",        # first adds ROC curve for Random Forest
        xlab="False Positive Rate (1-Specificity)",
        ylab="True Positive Rate (Sensitivity)")
        lines(df_SVM, col="blue")               # adds ROC curve for SVM
abline(a = 0, b = 1, col = "lightgray") # adds a diagonal line

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

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



auc()

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

Calculate AUC values for Random Forest and SVM models by using auc() function.

```
## Area under the curve: 0.8191
```

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

```
## Area under the curve: 0.8299
```

cumGainsTable()

plot Cumulative Response (Gain) chart for Random Forest and SVM models.

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

cumGainsTable() function of CustomerScoringMetrics package

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

```
# Extract the gain values for Gain chart
GainTable_RF <- cumGainsTable(prob_RF, test$diagnosis, resolution = 1/100)

GainTable_SVM <- cumGainsTable(prob_SVM, test$diagnosis, resolution = 1/100)
```

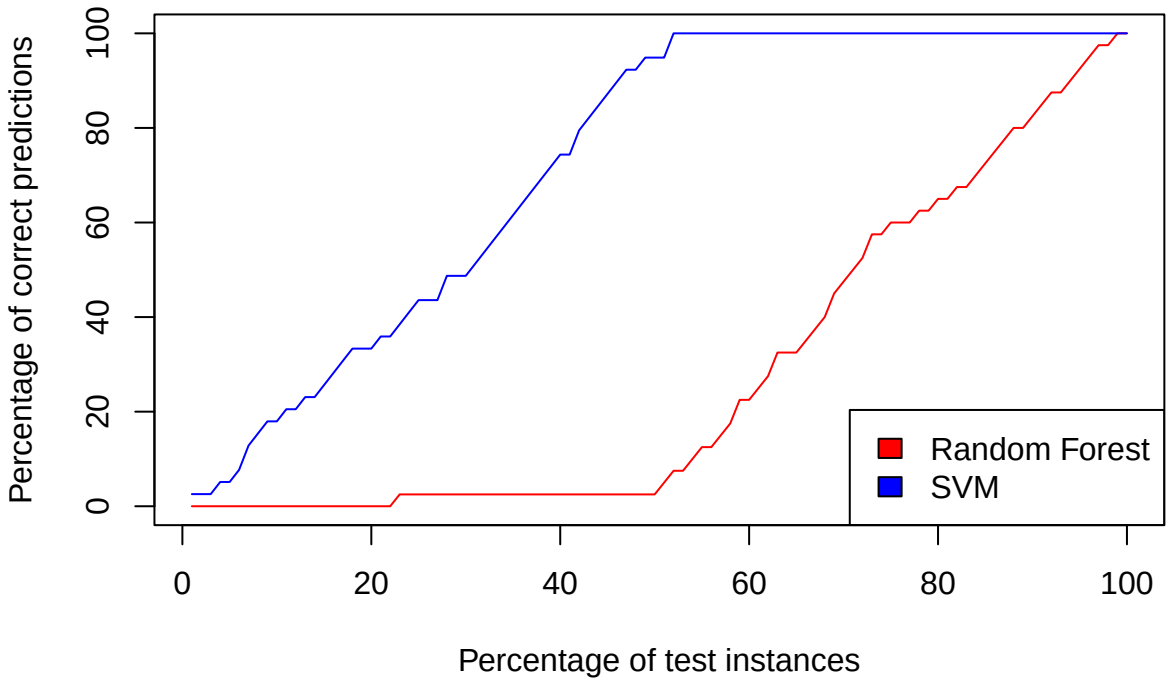
three arguments: the prediction probabilities (scores)/ the actual values of the target variables/ the increment of the threshold value

plot the gain chart

```
plot(GainTable_RF[,4], col="red", type="l",
  xlab="Percentage of test instances", ylab="Percentage of correct predictions")
  lines(GainTable_SVM[,4], col="blue", type="l")

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

plot the gain chart for Random Forest and SVM.


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