JHU Practical Machine Learning - Advanced R

DCrowley

2023-09-16

Contents

JHU Applied Machine Learning with Advanced R Principles

The Machine Learning proces: Data Science perspective

Project Overview:

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is now possible to collect a large amount of data about personal gym training activity relatively inexpensively. One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it. In this project, the goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. More information is available from the website here:

```
*[Source Training Data:] (http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.
```

Project Requirements:

The goal of this project is to predict the manner in which the participants did the exercise. This is the "classe" variable (the Y predicted variable - outcome). The R CARET package (Classification And REgression Training) was the base package used to meet the project's

requirements. Parallel processing capabilities have also been implemented:

- Re-sampling is the primary approach for optimizing predictive models with tuning parameters
- The Caret package itself includes a set of Classes to streamline the process for creating predictive models.

The practical machine learning Process steps using R for this project are as follows:

1.) Initial Data Pre Procesing

- Loading the Training and Testing data sets (.....)

- Cleaning the data sets of non needed dimensions and modifying certain columsn to produce more usable predictors.

2.) Exploratory Data Analysis:

- The preProcess class was used for centering and scaling:

  - One class method, also called preProcess estimated the required parameters for each operation.

    - preProcessStep - pre-process the predictor data: Center and Scale

      - It centers and scales a variable to mean 0 and standard deviation 1.

      - It ensures that the criterion for finding linear combinations of the predictors is based on how much variation they explain and therefore improves the numerical stability

3.) Data Splitting ():

- Simple Splitting based on the outcome patterns was use

- This mean random sampling occurs within each class and should preserve the overall class distribution of the data.

4.) Model Training and Tuning:

- The train function was used to:

  - Evaluate, using resampling, the effect of model tuning parameters on performance

  - Choose the "optimal" model across these parameters

  - Estimate model performance from a training set

```
#trainDS <- predict(preProcessStep, train)

#validateDS <- predict(preProcessStep, validate)

#test <- predict(preProcessStep, testingY)
```

- The trainControl function and "repeatedcv" capability was used to:

  - Implement K-Fold to evaluate the performance of the generated model on the data set

- This will give a measure of how well the predictions made by the model match the observed

- Create the object that will be used later in the caret train function

(Due to the intensive CPU/Memory resource required and noticed in early testing the number/repeats was reduced to 2 from an already low number of 10)

- In the preProcessStep Parallel Computing was enabled:

```
cl <- makePSOCKcluster(6)

registerDoParallel(cl)
```

5.) Evaluate and Measure Model Ac curacies

- Models implemented in this project fo Accuracy:

- Random Forrest

- Regression Trees

- Glmnet

Load pml-training data

```
#
# Read In and Data Reprocessing Class goes Here
#
trainingX <- read_csv("pml-training.csv",
                                        na=c("", "#DIV/0!", "NA"),
                                        progress = F,
                                        col_types = cols())
```

```
## New names:
## * `` -> `...1`
```

```
testingY  <- read_csv("pml-testing.csv",
                                        na=c("", "#DIV/0!", "NA"),
                                        progress = F,
                                        col_types = cols())
```

```
## New names:
## * `` -> `...1`
```

Build Project Classes

```
EDA <- R6Class(
        classname = "EDA",
        public = list(
                trainingX = NULL,
                testingY = NULL,
                initialize = function(tX = NA, tY = NA) {
                        self$trainingX <- tX
                        self$testingY <- tY

                },
                clean_training_data = function(){
                         missVal <- sapply(self$trainingX, function(x) sum(is.na(x)))
                         missVal <- missVal[missVal>0]
                         return(missVal)
                },
                create_unique_names = function(){
                        userNames <- unique(self$trainingX$user_name)
                        return(userNames)
                },
                preProcessStepT = function(trainingObj, userNames, missingValues){
                                trainingObj$cvtd_timestamp <- trainingObj$raw_timestamp_part_1 + trainir
                                trainingObj$raw_timestamp_part_1 <- NULL
                                trainingObj$raw_timestamp_part_2 <- NULL
                                trainingObj$X1 <- NULL
                                trainingObj$classe <- as.factor(trainingObj$classe)
                                trainingObj$user_name <- factor(trainingObj$user_name,userNames)
                                trainingObj$new_window <- NULL
                                trainingObj <- trainingObj %>% select(-matches(names(missingValues)))
                        return(trainingObj)
                },
                preProcessStepF = function(trainingObj, userNames, missingValues){
                                trainingObj$cvtd_timestamp <- trainingObj$raw_timestamp_part_1 + trainir
                                trainingObj$raw_timestamp_part_1 <- NULL
                                trainingObj$raw_timestamp_part_2 <- NULL
                                trainingObj$X1 <- NULL
                                trainingObj$user_name <- factor(trainingObj$user_name,userNames)
                                trainingObj$new_window <- NULL
                                trainingObj <- trainingObj %>% select(-matches(names(missingValues)))
                        return(trainingObj)
                }
        )
)
```

```
CARETMOD <- R6Class(
        classname = "CARETMOD",
        public = list(
                trainingX2 = NULL,
                initialize = function(tX2 = NA) {
                        self$trainingX2 <- tX2
                },
                preProcessStep2 = function(){
                        preProcessStp <- preProcess(self$trainingX2, method = c("center", "scale"))
                return(preProcessStp)
```

```
                },
                fitContr = function(){
                        fitC <- trainControl(method = "repeatedcv",
                           number = 2,
                           repeats = 2,
                           savePredictions=TRUE,
                           classProbs=TRUE)
                        return(fitC)
                },
                parallCompStart = function(cl){
                        cl <- makePSOCKcluster(4)
                        return(registerDoParallel(cl))
                },
                inTrain = function(){
                        inTraining <- createDataPartition(self$trainingX2$classe, p = .8, list = T)
                        return(inTraining)
                }
        )
)
```

Create a new Exploratory Data Analysis Object

```
machLearn <- EDA$new(trainingX,testingY)
machLearnMissVal <- machLearn$clean_training_data()
machLearnUnqNM <- machLearn$create_unique_names()

#
# Run the training and the test data sets from the EDA object through the preProcess Steps
#
trainingX2 <- machLearn$preProcessStepT(trainingX, machLearnUnqNM, machLearnMissVal)
testingY2 <- machLearn$preProcessStepF(testingY, machLearnUnqNM, machLearnMissVal)
```

Create the Caret Package pre processing object

```


# Create caret object
caretPreProc <- CARETMOD$new(trainingX2)

# Center and Scale
preProcessCtrScale <- caretPreProc$preProcessStep2()

# Data Partition
trainingSplit <- caretPreProc$inTrain()

# Finalize Data set contents
trainDS <- trainingX2[trainingSplit$Resample1,]
validateDS <- trainingX2[-trainingSplit$Resample1,]

# Add Data Split and Center/Split and send to the predict function
```

```
trainDSPred <- predict(preProcessCtrScale, trainDS)
validateDSPred <- predict(preProcessCtrScale, validateDS)
testDSPred <- predict(preProcessCtrScale, testingY2)
```

- Get Training and Validation Sets ready for the Caret package prediction function

Implement K-Fold to evaluate the performance of the generated model

```
fitContr <- caretPreProc$fitContr()
cl <- makePSOCKcluster(5)
registerDoParallel(cl)
set.seed(820)
```

Setup Parallel Processing

Run Caret "train" function on the Training Models

```
# Mode: Regression Trees
        rpartPredFit <- train(classe ~ .,
                              data = trainDSPred,
                              method = 'rpart',
                              trControl = fitContr,
                              metric = 'Accuracy',
                              tuneLength  = 6)
        save(rpartPredFit, file="model_rpartPredFit.rda")


# Model: Random Forrest

        rfFitPredMod <- train(classe ~ .,
                             data = trainDSPred,
                             method = 'rf',
                             trControl = fitContr,
                             metric = 'Accuracy',
                             tuneLength  = 6)
        save(rfFitPredMod, file="model_rfFitPredMod.rda")


# Model: GLMNET

        glmNetPredModFit <- train(classe ~ .,
                              data = trainDSPred,
                              method = "glmnet",
                              trControl = fitContr,
                              metric = 'Accuracy',
                              tuneLength  = 6,
                              family = "multinomial",
                              type.multinomial = "grouped")
```

```
## Warning: from glmnet C++ code (error code -38); Convergence for 38th lambda
## value not reached after maxit=100000 iterations; solutions for larger lambdas
## returned
```

```
        save(glmNetPredModFit, file="model_glmnet.rda")
```

```
# Stop the Cluster

stopCluster(cl)
```

```
load("model_rpartPredFit.rda")
load("model_rfFitPredMod.rda")
load("model_glmnet.rda")
```

Evaluate, Model Predictions - Accuraciues and ggplot/Boxplot Outputs

Load previous Saved Model Outputs

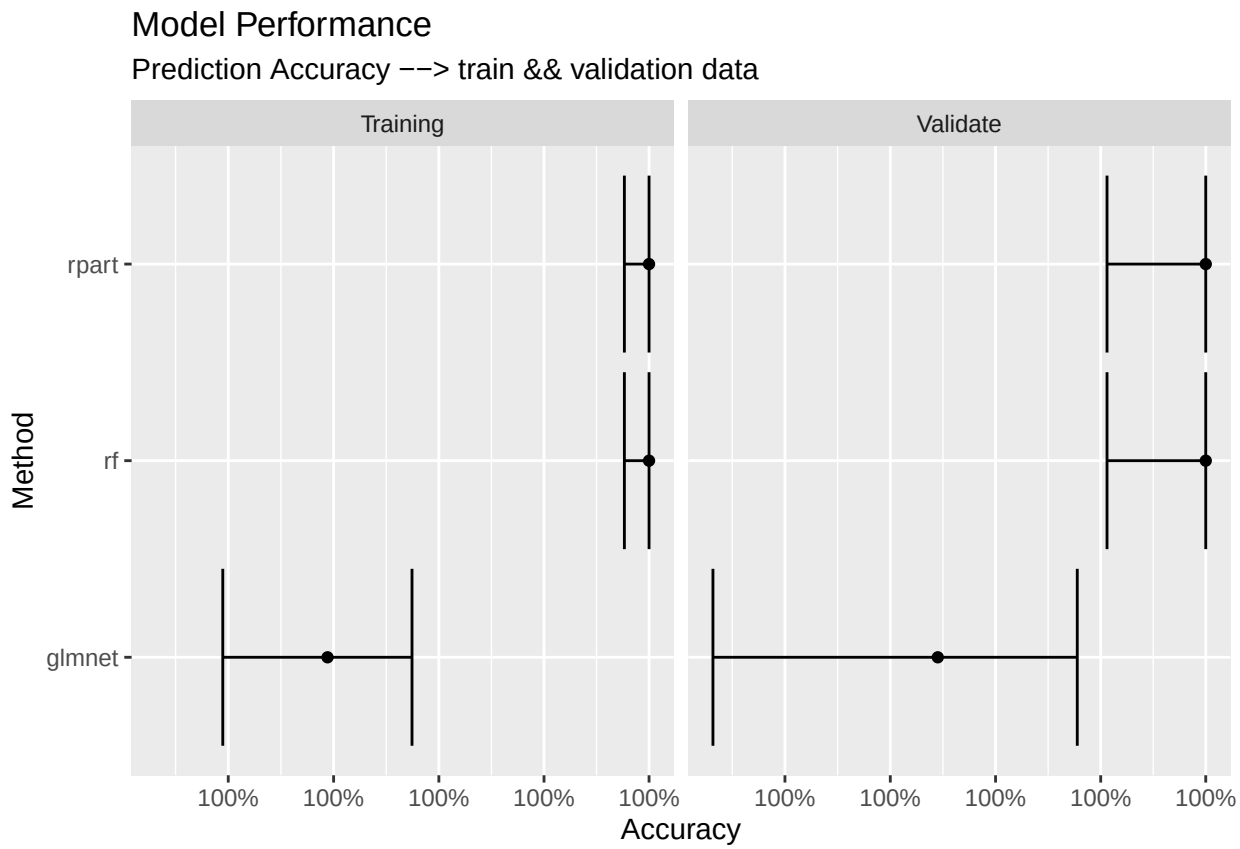
####(Model Outputs are saved - Prediction Model runs are time consuming and heavy on resoure)
Each prediction model algorithim has 2 different parameter sets will be tuned automatically on a 2x2 (repeated) cross-validated training data set. Running time and limited CPU/Memory usage is key so reduced trainControl parameters (number and repeats) had to be configured.
Prediction Accuracy is the meaurement criteria used to understand Model output.

```
evalData <- function(model, name, data) {
        tibble(Method = model$method,
               Dataset = name,
               broom::tidy(confusionMatrix( data=predict(model, data),
                                            reference=data$classe)))
}

#
# Load the previously generated and saved models
#
load("model_rpartPredFit.rda")
load("model_rfFitPredMod.rda")
load("model_glmnet.rda")

#
# Model Evaluation: Create the datframes for use in ggplot and plot for Accuracy
#
performance <- tibble() %>%
        bind_rows(evalData(rpartPredFit, "Training", trainDSPred),
                  evalData(rpartPredFit, "Validate", validateDSPred),
                  evalData(rfFitPredMod, "Training", trainDSPred),
                  evalData(rfFitPredMod, "Validate", validateDSPred),
                  evalData(glmNetPredModFit, "Training", trainDSPred),
                  evalData(glmNetPredModFit, "Validate", validateDSPred))
```

```
ggplot(performance %>% filter(term == "accuracy"),
       aes(x=estimate,
           xmin=conf.low,
           xmax=conf.high,
           y=Method)) +
        geom_errorbar() +
        geom_point() +
        facet_wrap(.~Dataset) +
        xlab("Accuracy") +
        scale_x_continuous(labels=scales::label_percent(accuracy = 1),
                           trans="log10") +
        labs(title="Model Performance",
             subtitle = "Prediction Accuracy --> train && validation data")
```



Testing Set Prediction Outcomes

```
tibble(N = 1:nrow(testDSPred),
       classe=predict(rfFitPredMod, testDSPred))
```

```
## # A tibble: 20 x 2
##        N classe
##    <int> <fct>
##  1     1 A
```

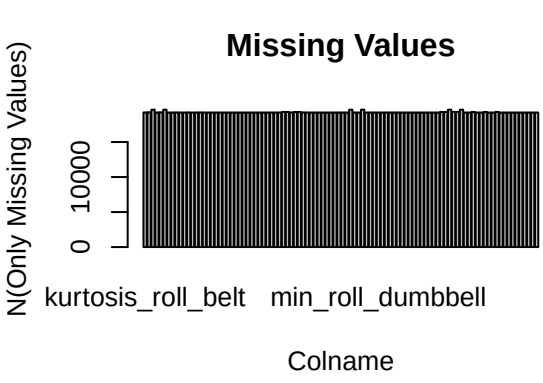
```
##  2     2 A
##  3     3 A
##  4     4 A
##  5     5 A
##  6     6 A
##  7     7 A
##  8     8 A
##  9     9 A
## 10    10 A
## 11    11 A
## 12    12 A
## 13    13 A
## 14    14 A
## 15    15 A
## 16    16 A
## 17    17 A
## 18    18 A
## 19    19 A
## 20    20 A
```

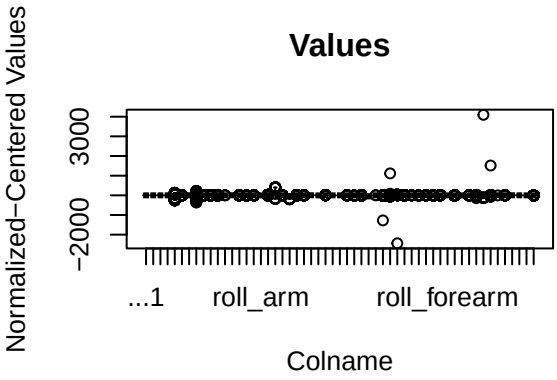
Initial Exploratory plot

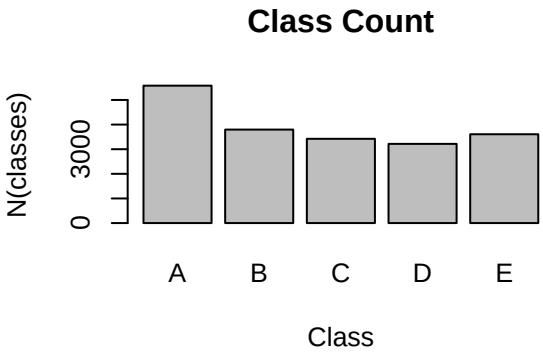
This exploratory plot gives an understanding of the state of the original dataset and the number of missing values in multiple columns which were excluded analysis.

```
nclasses <- trainingX2 %>% count(classe)

par(mfrow=c(2,2))
barplot(machLearnMissVal, ylab="N(Only Missing Values)", xlab="Colname", main="Missing Values")
boxplot(sapply(trainingX2 %>%
                      select_if(is.numeric),
               function(x) (x - sd(x)) / (mean(x))),
        ylab="Normalized-Centered Values", xlab="Colname", main="Values")
barplot(n ~ classe, data=nclasses, ylab="N(classes)", xlab="Class", main="Class Count")
```







Randon Forrest Model Metrics

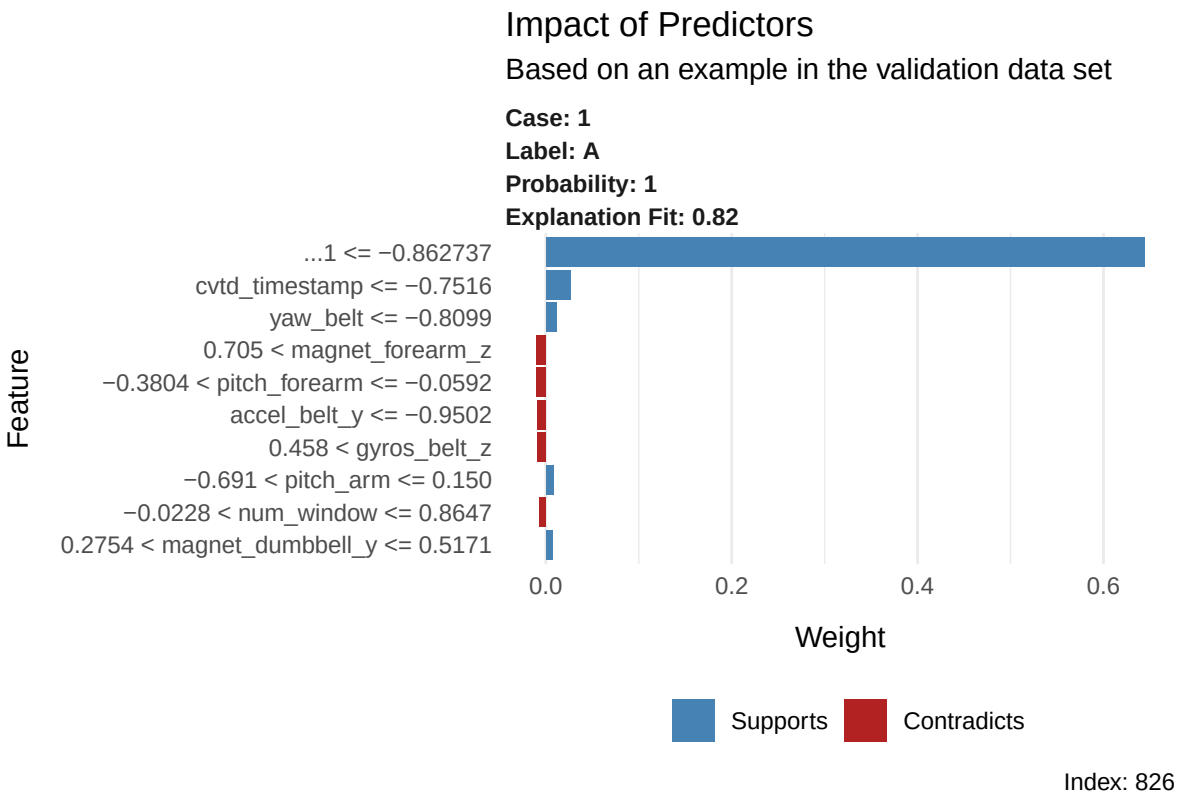
```
rfFitPredMod$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 13
##
##         OOB estimate of  error rate: 0%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 4464    0    0    0    0           0
## B    0 3038    0    0    0           0
## C    0    0 2738    0    0           0
## D    0    0    0 2573    0           0
## E    0    0    0    0 2886           0
```

Measuing the Predictor Impact

```
samples <- c(800, 1170, 2215, 3001, 3321)
limeDSTrain <- lime(trainDSPred, rfFitPredMod)
```

```
impactPlot <- explain(validateDSPred[c(samples[1]),], limeDSTrain, n_labels = 1, n_features = 10)
plot_features(impactPlot) +
  labs(title="Impact of Predictors",
       subtitle="Based on an example in the validation data set",
       caption="Index: 826")
```



Conclusion

###For this set of collected information and the code base, the random forrest model performed better, on the training and validation datasets. Better performance fot the glmnet model might be further improved by investing more time in using more hyper parameters.

The constant issue of potentially introducing bias into the source data set by dropping certain columns of data and futher modifying others seems more of an art. A good next step would be ot investigate how certain hyper parameters could be used and what ranges of values they can be in order to make all models more accurate.

Whilst random tree did perfprm very well, there is also the concern of overfitting has ocurred and if more pertinent data should also be included into the dataset.