Practical Machine Learning Assignment (Coursera)

Rodolfo Kirch Veiga

9/30/2020

Background

In this project data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants was used to predict the manner in which they did weight lifting exercises.

Data

Datasets for training and testing machine learning models were provided by a Brazilian university (PUC-RIO) and are available for download on the following links: Training Testing. Extra information about the PUC-RIO project can be found Here.

Overview

In order to develp a predictive model, the following steps were executed:

1. Load the data
2. Clean up
3. Pre-process
4. Train
5. Evaluate
6. Validate

---

1. Load the data

The training and testing data was loaded in the R Session, in order to use them to develop the machine learning models.

```
# define urls to download the data
url_train = 'https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv'
url_test = 'https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv'
# load training and testing data
data = read.csv(url_train)
dim(data)
```

```
## [1] 19622   160
```

```r
# load validation data
data_valid = read.csv(url_test)
dim(data_valid)
```

```
## [1]  20 160
```

2. Clean up

A cleaning process was carried out to remove variables with missing or low-level information.

First, unuseful labels such as the participants names, was removed and blank/empty fields were filled with NA's.

```r
# load package
library(dplyr)
# clean train set
data = data %>%
  select(-c(1:7)) %>% # remove unuseful labels (columns 1 to 7)
  mutate_if(is.character, list(~na_if(., ''))) # convert empty fields into na
# clean validation set
data_valid = data_valid %>%
  select(-c(1:7)) %>% # remove unuseful labels (columns 1 to 7)
  mutate_if(is.character, list(~na_if(., ''))) # convert empty fields into na
```

Machine learning techniques do not recognize NA, therefore, rows and columns (considering extreme cases) were removed.

```r
# check for na values
sum(sapply(data, function(x) any(is.na(x))))
```

```
## [1] 100
```

```r
# 100 columns contain na values
unique(colSums(is.na(data))/nrow(data))
```

```
## [1] 0.0000000 0.9793089
```

```r
# some variables have no na values, while others have 97.9% of na values

# remove variables containing na values
data[, colSums(is.na(data)) > 0] = NULL
dim(data)
```

```
## [1] 19622    53
```

```r
data_valid[, colSums(is.na(data_valid)) > 0] = NULL
dim(data_valid)
```

```
## [1] 20 53
```

3. Pre-process

After data was cleaned up, it was necessary to split it into training and validation set.

```
# load package
library(caret)

# split training data into training and validation sets
index = createDataPartition(y = data$classe, p = 0.75, list = FALSE)
data_train = data[index, ]
nrow(data_train)
```

```
## [1] 14718
```

```
data_test = data[-index, ]
nrow(data_test)
```

```
## [1] 4904
```

To avoid extra bias, all the pre-process had to be carried out on the training set and replied on the validation set.

Every column of the training data set was inspected to check if there is any variable with very low variance or `character` class variables. Low (or near to zero) variance would be removed, while `character` variables would be transformed into dummy variables.

```
# check the variable classes
sum(sapply(data_train[, -ncol(data_train)], class) == 'character')
```

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

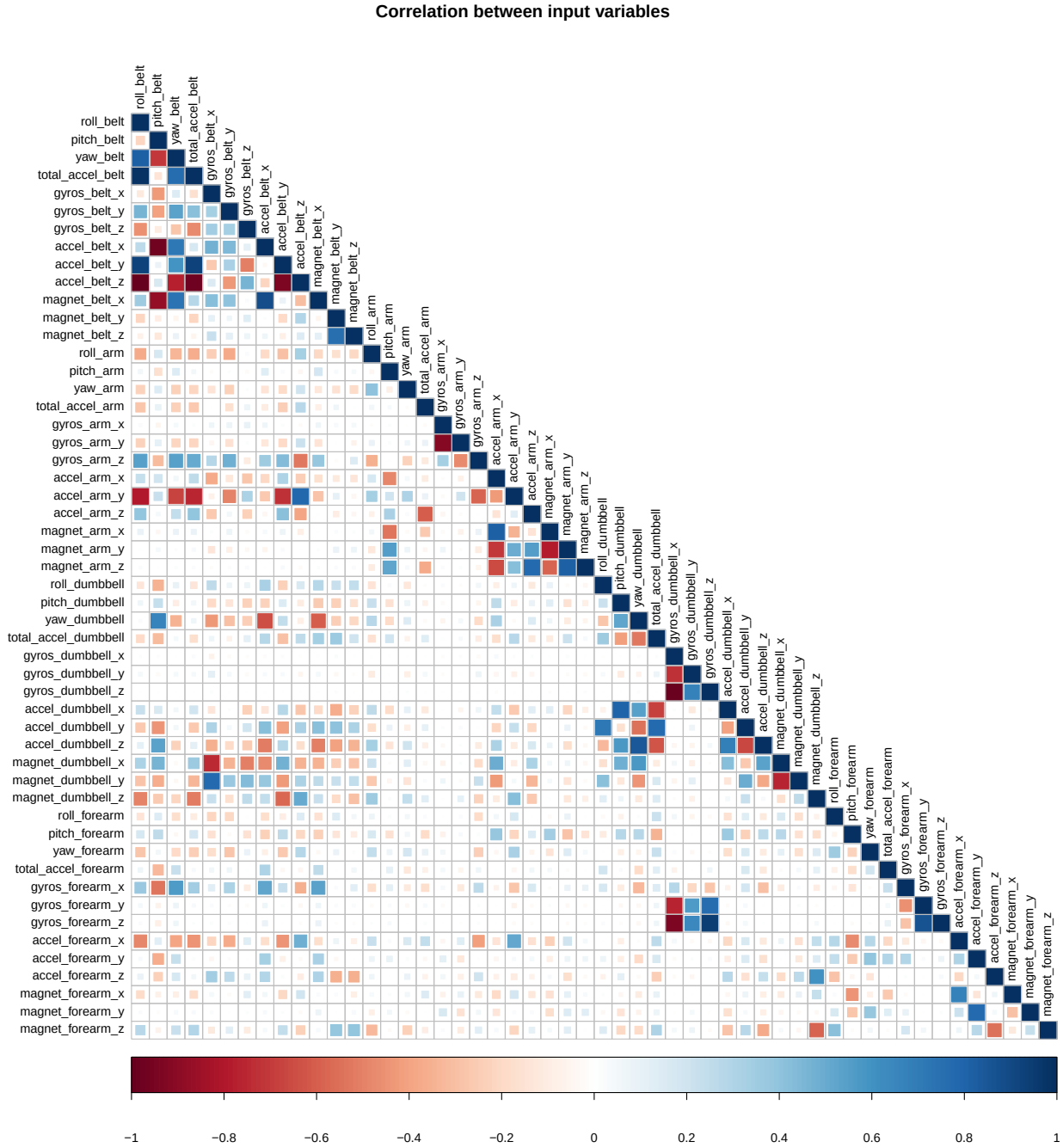
```
# check for near to zero variables
length(nearZeroVar(data_train))
```

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

It was recognized that all classes are `numeric` or `integer` and dummy variables did not have to be created. Also, no variables with near to zero variance were found and no columns were removed.

Correlation between the input variables (all but `classe` variable) was checked and the variables/columns presenting high correlation were removed.

```
library(corrplot)
# check for correlation between the input variables
correlations = cor(data_train[, -ncol(data_train)])
corrplot(correlations, title = 'Correlation between input variables', mar = c(0, 0, 1, 0),
         method = 'square', type = 'lower', tl.col = 'black', tl.cex = 0.8)
```



As it was shown in the graph, some variables have high correlation. A `cutoff = 0.9` was defined as a `findCorrelation()` argument to find the index of high correlated variables. If necessary, these variables were removed from the data sets.

```
# remove high correlated variables
cols <- findCorrelation(correlations, cutoff = 0.9)
```

```r
colnames(data)[cols]
```

```
## [1] "accel_belt_z"     "roll_belt"        "accel_belt_y"     "accel_belt_x"
## [5] "gyros_dumbbell_x" "gyros_dumbbell_z" "gyros_arm_x"
```

```r
data_train = data_train[, -cols]
dim(data_train)
```

```
## [1] 14718    46
```

```r
data_test = data_test[, -cols]
dim(data_test)
```

```
## [1] 4904   46
```

4. Train

Four models were training with the following characteristics:

- Training algorithms: decision tree, random forest, extreme gradient boosted tree and support vector machine.
- Sampling method: cross-validation with 5 folders
- Pre-process technique: BoxCox

A training function was defined considering the common characteristics between the models. The models were trained using the `caret` package and the user-defined function (`FitModel()`).

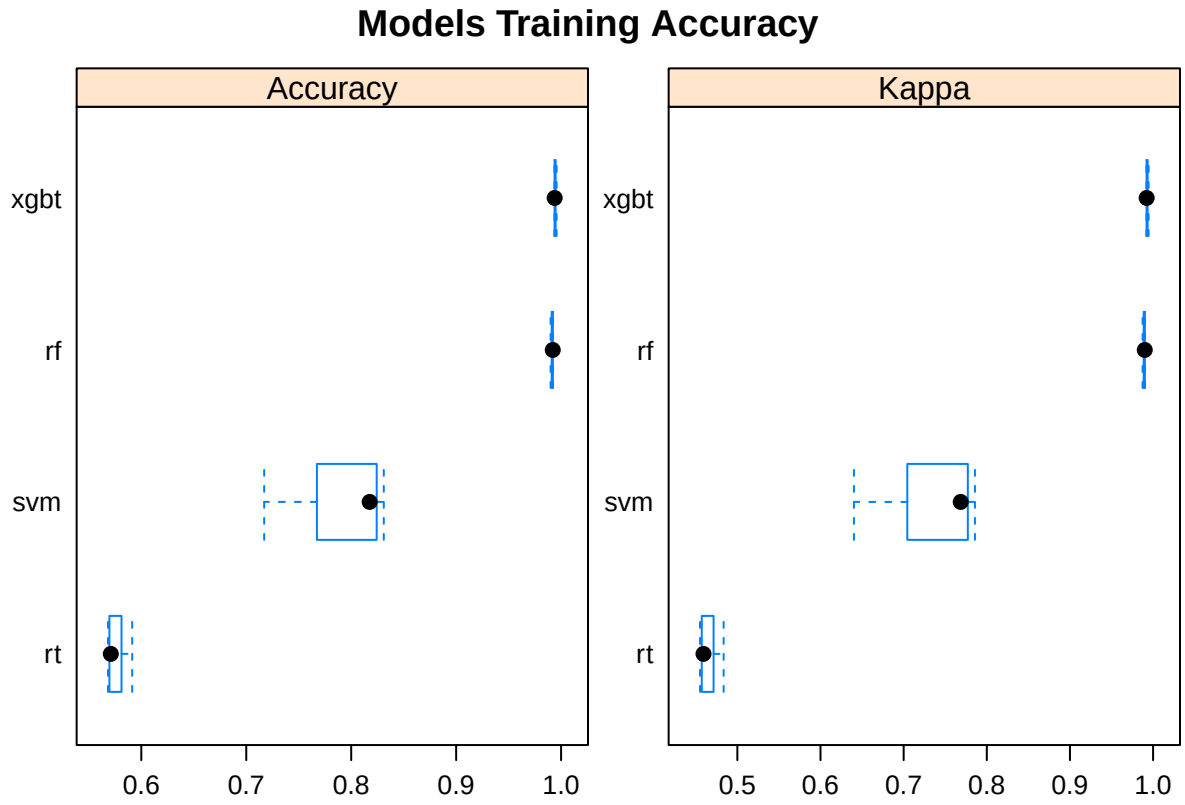
```r
# define a function to fit the models
FitModel = function(method) {
  train_control = trainControl('cv', 3, returnData = FALSE, verboseIter = TRUE)
  model = train(classe ~ ., data_train, method, trControl = train_control,
                tuneLength = 5, preProcess = 'BoxCox')
  return(model)
}

# train models
methods = c(rt = 'rpart', rf = 'rf', xgbt = 'xgbTree', svm = 'lssvmRadial')
# set seed for reproducibility
set.seed(10)
models = lapply(methods, FitModel)
# save model to access later
saveRDS(models, 'models.rds')
```

5. Evaluate

After training models with several different methods/techniques, a comparison between the developed models was performed and the best model was chosen.

```
# plot training accuracy
comparison = resamples(models)
names(comparison$values)[-1] = names(comparison$values)[-1]
bwplot(comparison, main = 'Models Training Accuracy',
       scales = list(x = list(relation = 'free'), y = list(relation = 'free')))
```




```
# calculate the numeric value for the accuracy, since rf and xgbt accuracies are pretty similar
mean(comparison$values$'rf~Accuracy')
```

```
## [1] 0.9914392
```

```
mean(comparison$values$'xgbt~Accuracy')
```

```
## [1] 0.9944286
```

Since RF and XGBT models presented pretty much the same accuracy, RF was chosen, do to its high interpretability. Then, the confusion matrix of the final model for the testing set was printed out.

```
# define final model
model = models$rf

# predict the validation set
preds_test = predict(model, newdata = data_test)
```

```r
# print out the confusion matrix for the testing set
confusion_mtx = confusionMatrix(preds_test, as.factor(data_test$classe))
confusion_mtx$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    0    0    0    0
##          B    0  948    2    0    0
##          C    0    1  852    0    1
##          D    0    0    1  804    0
##          E    0    0    0    0  900
```

```r
confusion_mtx$overall[1]
```

```
##  Accuracy
## 0.9989804
```

The final model achieved a 99.96% accuracy on the testing set, which means that this model can be used to predict the validation set.

6. Validate

Finally, the final model was used to predict the validation set values and the confusion matrix was generated.

```r
# predict the validation set
predict(model, newdata = data_valid)
```

```
##  [1] B A B A A E D B A A B C B A E E A B B B
## Levels: A B C D E
```