Practical Machine Learning Project

Code

KK

2021-05-10

Load libraries

Hide

```
install_load <- function (package1, ...)  {

  # convert arguments to vector
  packages <- c(package1, ...)

  # start loop to determine if each package is installed
  for(package in packages){

    # if package is installed locally, load
    if(package %in% rownames(installed.packages()))
      do.call('library', list(package))

    #else use install.packages then load
    else {
      install.packages(package, repos = "http://cran.stat.unipd.it/")
      do.call("library", list(package))
    }

  }
}
libs = c("caret", "dplyr", "VIM")
install_load(libs)
```

Get the data

Hide

```
data_dir = "./data"
training_url = "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
test_url = "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
training_file = "pml-training.csv"
test_file = "pml-test.csv"
if (!file.exists(data_dir)) {
  dir.create(data_dir)
}
if (!file.exists(file.path(data_dir, training_file))) {
  download.file(training_url, destfile=file.path(data_dir, training_file))
}
if (!file.exists(file.path(data_dir, test_file))) {
  download.file(test_url, destfile=file.path(data_dir, test_file))
}
```

Read the Data

Load the data into 2 different data frames

Hide

```
train <- read.csv(file.path(data_dir, training_file))
test <- read.csv(file.path(data_dir, test_file))
dim(train)
dim(test)
head(train)
```

Clean the data

Check if in the observations are present NA values or missing OBS that can raise errors/bias during the model training.

Hide

```
sum(complete.cases(train))
```

Too few observation to have a correct training.

Eliminate the columns with NA/missing values

Let's see colnames

Hide

```
colnames(train)
plot(colMeans(is.na(train)))
```

There are columns with a lot of missing values.

We will reatain only the columns without NA values

First covert all the data in NUMERIC form to coerce the empty factor to NA

Hide

```
trainClasse = train$classe
trainRaw = train[, sapply(train, is.numeric)]
testRaw = test[, sapply(test, is.numeric)]
```

Remove columns with NA values

Hide

```
trainFilter <- trainRaw[, colSums(is.na(trainRaw)) == 0]
# Attach Classe variable
trainFilter$classe = trainClasse
testFilter <- testRaw[, colSums(is.na(testRaw)) == 0]
```

Dimension

Hide

```
dim(trainFilter)
dim(testFilter)
```

Removing other unuseful columns like username, timestamp and ID

```
unwanted = !grepl("X|timestamp", colnames(trainFilter))
cols = colnames(trainFilter)[unwanted]
trainFilter = trainFilter %>%
  select(cols)
unwanted = !grepl("X|timestamp", colnames(testFilter))
cols = colnames(testFilter)[unwanted]
testFilter = testFilter %>%
  select(cols)
```

Get dimension of the filtered dataset

Hide

```
dim(trainFilter)
dim(testFilter)
```

Slice the data

We will slice the Training data into **Training** and **Validation** set using the 80-20 rule.

Hide

```
set.seed(12022018) # Today's date
inTrain <- createDataPartition(trainFilter$classe, p=0.70, list=F)
trainData <- trainFilter[inTrain, ]
validationData <- trainFilter[-inTrain, ]
dim(trainData)
```

Data modeling

We will fit a model using **Random Forest** and **XGBoost** (very popular in challange like kaggle.com) for several reasons:

1. With tree-based models, **you can safely ignore** predictors correlation issues
2. Zero- and Near Zero-Variance Predictors **does not** imply on tree-based models
3. As each feature is processed separately, and the possible splits of the data don't depend on scaling, no preprocessing like normalization or standardization of features is needed for decision tree algorithms.

Random forest

Model

Hide

```
controlRf <- trainControl(method="cv", 5, allowParallel = TRUE)
modelRf <- train(classe ~ ., data=trainData, method="rf", trControl=controlRf, ntree=250)
modelRf
```

Hide

```
controlRf <- trainControl(method="cv", 5, allowParallel = TRUE)
modelRf <- train(classe ~ ., data=trainData, method="rf", trControl=controlRf, ntree=250)
modelRf
```

Performance of the model on the validation data set

Hide

```
predict_rf <- predict(modelRf, validationData)
confusionMatrix(validationData$classe, predict_rf)
```

Very accurate model to classify **classe** feature

XGBoost

Hide

```
controlXGB <- trainControl(method="cv", 5, allowParallel = TRUE)
modelXGB <- train(classe ~ ., data=trainData, method="xgbTree", trControl=controlXGB)
```

Hide

```
modelXGB
```

Performance of the model on the validation data set

Hide

```
predict_XGB <- predict(modelXGB, validationData)
confusionMatrix(validationData$classe, predict_XGB)
```

With XGB we reach a better accuracy on validation data.

Only 2 mislabeled prediction A->B

Compare models

Hide

```
# collect resamples
model_results <- resamples(list(RF=modelRf, XGB=modelXGB))
# summarize the distributions
summary(model_results)
# boxplots of results
bwplot(model_results)
# dot plots of results
dotplot(model_results)
```

Predict Test data with RF and XGB

Hide

```
resultRf <- predict(modelRf, testFilter[, -length(names(testFilter))])
resultXGB <- predict(modelXGB, testFilter[, -length(names(testFilter))])
resultRf
resultXGB
confusionMatrix(resultRf, resultXGB)
```

Finally the model predict the TEST data in the same way, but we noticed that XGB works better with the trainig set