Practical Machine Learning Project

Wei Zhang

10/10/2020

Synopsis

This project uses data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants to predict the manner in which they did the exercise.

Random Forest is the best prediction model which yields accuracy of 99.51%.

Data Source

The data for this project come from this source: http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har)
The training data: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv)
The test data: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv)

Loading Data

```
traingDataURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
validationDataURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
training.raw <- read.csv(traingDataURL, header = T, na.strings = c("", "NA"))
validation.raw <- read.csv(validationDataURL, header = T, na.strings = c("", "NA"))
```

Loading libraries

```
library(caret)
library(rattle)
library(randomForest)
library(formattable)
```

Data Exploration and Preprocessing

1. Explore raw data sets, convert manner variable “classe” to factor
   There are 19622 rows of observation and 160 variables in the original training data, 20 rows of observation and 160 variables in the validation data.

```
str(training.raw)
head(training.raw)
summary(training.raw)
```

```
dim(training.raw);dim(validation.raw)
```

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

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

```
training.raw$classe <- as.factor(training.raw$classe)
```

2. Remove variables not related
   This project uses data from accelerometers on the belt, forearm, arm, and dumbell to predict the excercise quality. Remove unrelated variables “X”, “user_name”, “raw_timestamp_part_1”, “raw_timestamp_part_2”, “cvtd_timestamp”, “new_window”, and “num_window”.

```
training.cleaned <- training.raw[, -c(1:7)]
```

3. Remove variables with too many missing values
   As a rule of thumb, when the data goes missing on 60–70 percent of the variable, dropping the variable should be considered.

```
training.cleaned <- training.cleaned[, -which(colMeans(is.na(training.cleaned)) > 0.7)]
```

4. Remove zero covariates, which are basically have no variability in them
   No zero covariates are found after the previous cleanup.

```
nsv <- nearZeroVar(training.cleaned,saveMetrics=TRUE)
table(nsv$nzv)
```

```
##
## FALSE
##    53
```

We have 53 variables and no missing values after the cleanup.

```
dim(training.cleaned);
```

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

```
any(is.na(training.cleaned))
```

```
## [1] FALSE
```

5. Data Partitioning: The partioning allocates 75% of the clean testing data into Testing set and 25% into Test set.

```
set.seed(1234)
inTrain <- createDataPartition(y=training.cleaned$classe, p=0.75, list=FALSE)
training <- training.cleaned[inTrain,]
testing <- training.cleaned[-inTrain,]
dim(training); dim(testing)
```

```
## [1] 14718    53
```

```
## [1] 4904   53
```

6. Variables that have high correlation coefficients > 0.8

```
M <- abs(cor(training.cleaned[,-53]))
diag(M) <- 0
which(M > 0.8,arr.ind=T)
```

```
##                  row col
## yaw_belt           3   1
## total_accel_belt   4   1
## accel_belt_y       9   1
## accel_belt_z      10   1
## accel_belt_x       8   2
## magnet_belt_x     11   2
## roll_belt          1   3
## roll_belt          1   4
## accel_belt_y       9   4
## accel_belt_z      10   4
## pitch_belt         2   8
## magnet_belt_x     11   8
## roll_belt          1   9
## total_accel_belt   4   9
## accel_belt_z      10   9
## roll_belt          1  10
## total_accel_belt   4  10
## accel_belt_y       9  10
## pitch_belt         2  11
## accel_belt_x       8  11
## gyros_arm_y       19  18
## gyros_arm_x       18  19
## magnet_arm_x      24  21
## accel_arm_x       21  24
## magnet_arm_z      26  25
## magnet_arm_y      25  26
## accel_dumbbell_x  34  28
## accel_dumbbell_z  36  29
## gyros_dumbbell_z  33  31
## gyros_forearm_z   46  31
## gyros_dumbbell_x  31  33
## gyros_forearm_z   46  33
## pitch_dumbbell    28  34
## yaw_dumbbell      29  36
## gyros_forearm_z   46  45
## gyros_dumbbell_x  31  46
## gyros_dumbbell_z  33  46
## gyros_forearm_y   45  46
```

Multicollinearity exist in the data. Since the most important thing for this project is to find the best performing model and interpreting predictor importance can be sacrificed, PCA may be used for pre-processing. PCA is most useful in linear-type models. However, it can reduce the number of predictors for the Random Forest to process, therefore may help speed up the training of Random Forest model. Note that computational cost is one of the biggest drawbacks of Random Forest.

```
preProc <- preProcess(training[, -53], method="pca", thresh=0.95)
preProc
```

```
## Created from 14718 samples and 52 variables
##
## Pre-processing:
##   - centered (52)
##   - ignored (0)
##   - principal component signal extraction (52)
##   - scaled (52)
##
## PCA needed 24 components to capture 95 percent of the variance
```

```
trainPC <- predict(preProc, training)
testPC <- predict(preProc, testing)
```

Prediction Models

Regression and classification are categorized under the same umbrella of supervised machine learning. The main difference between them is that the output variable in regression is numerical (or continuous) while that for classification is categorical (or discrete). The target variable to predict for this project is “classe”, a categorical variable. The 5 values of “classe” are described as:

- A: exactly according to the specification
- B: throwing the elbows to the front
- C: lifting the dumbbell only halfway
- D: lowering the dumbbell only halfway
- E: throwing the hips to the front

We will use classification algorithms to build 3 prediction models.

Model 1: Decision Tree

```
modelFit_rpart <- train(classe~.,method="rpart",data=training)
predict_rpart <- predict(modelFit_rpart,newdata=testing)
confusionMatrix_rpart <- confusionMatrix(predict_rpart, testing$classe)
confusionMatrix_rpart
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1274  390  401  386  141
##          B   18  344   36  131  127
##          C  100  215  418  287  244
##          D    0    0    0    0    0
##          E    3    0    0    0  389
##
## Overall Statistics
##
##                Accuracy : 0.4945
##                  95% CI : (0.4804, 0.5086)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3385
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9133  0.36249  0.48889   0.0000  0.43174
## Specificity            0.6244  0.92111  0.79106   1.0000  0.99925
## Pos Pred Value         0.4915  0.52439  0.33070      NaN  0.99235
## Neg Pred Value         0.9477  0.85758  0.87995   0.8361  0.88652
## Prevalence             0.2845  0.19352  0.17435   0.1639  0.18373
## Detection Rate         0.2598  0.07015  0.08524   0.0000  0.07932
## Detection Prevalence   0.5285  0.13377  0.25775   0.0000  0.07993
## Balanced Accuracy      0.7688  0.64180  0.63997   0.5000  0.71550
```

Model 2: Random Forest without PCA

My computer crashes when building RF model using: modelFit_rf<-train(classe~., method="rf", data=training)

The alternative is to use randomForest library (https://www.kaggle.com/general/7951 (https://www.kaggle.com/general/7951)):

```
mtry <- tuneRF(training[,-53], training$classe, ntreeTry=500, stepFactor=1.5,improve=0.01,
               plot=FALSE, trace=TRUE, dobest=FALSE)
```

This will give a few values of mtry, the best is the one with the least OOB error. Now we can train Random Forest using:

```
system.time(modelFit_rf<-randomForest(classe~., data=training, mtry=15, ntree=500))
```

```
##    user  system elapsed
##   45.01    0.49   45.58
```

```
predict_rf <- predict(modelFit_rf,newdata=testing)
confusionMatrix_rf <- confusionMatrix(predict_rf, testing$classe)
confusionMatrix_rf
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    7    0    0    0
##          B    0  941    8    0    0
##          C    0    1  846    5    0
##          D    0    0    1  799    2
##          E    0    0    0    0  899
## 
## Overall Statistics
## 
##                Accuracy : 0.9951
##                  95% CI : (0.9927, 0.9969)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
## 
##                   Kappa : 0.9938
## 
##  Mcnemar's Test P-Value : NA
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9916   0.9895   0.9938   0.9978
## Specificity            0.9980   0.9980   0.9985   0.9993   1.0000
## Pos Pred Value         0.9950   0.9916   0.9930   0.9963   1.0000
## Neg Pred Value         1.0000   0.9980   0.9978   0.9988   0.9995
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2845   0.1919   0.1725   0.1629   0.1833
## Detection Prevalence   0.2859   0.1935   0.1737   0.1635   0.1833
## Balanced Accuracy      0.9990   0.9948   0.9940   0.9965   0.9989
```

Model 3: Random Forest With PCA

```
system.time(modelFit_rf_PCA <-randomForest(classe~., data=trainPC, mtry=15, ntree=500))
```

```
##    user  system elapsed 
##   25.92    0.49   26.49
```

```
predict_rf_PCA <- predict(modelFit_rf_PCA, newdata=testPC)
confusionMatrix_rf_PCA <- confusionMatrix(predict_rf_PCA, testing$classe)
confusionMatrix_rf_PCA
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1382   18    1    2    0
##          B    5  918   21    2    4
##          C    4   13  819   31    5
##          D    4    0   13  766    3
##          E    0    0    1    3  889
##
## Overall Statistics
##
##                Accuracy : 0.9735
##                  95% CI : (0.9686, 0.9778)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9665
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9907   0.9673   0.9579   0.9527   0.9867
## Specificity            0.9940   0.9919   0.9869   0.9951   0.9990
## Pos Pred Value         0.9850   0.9663   0.9392   0.9746   0.9955
## Neg Pred Value         0.9963   0.9922   0.9911   0.9908   0.9970
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2818   0.1872   0.1670   0.1562   0.1813
## Detection Prevalence   0.2861   0.1937   0.1778   0.1603   0.1821
## Balanced Accuracy      0.9923   0.9796   0.9724   0.9739   0.9928
```

Accuracy comparison:

- Decision Tree: 49.45%
- Random Forest without PCA: 99.51%
- Random Forest with PCA: 97.35%

Conclusion

Compare with Random Forest without PCA(RF), Random Forest with PCA has reduced half of the model building time but its accuracy is lower. RF takes less than 2 minutes to build. It has an accuracy of 99.51% and its out-of-sample-error is about 0.49%. RF performs the best prediction among the 3 models.

Predicton with Random Forest (without PCA) on validation data

```
predict(modelFit_rf,newdata=validation.raw)
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  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
```