Practical Machine Learning Project Write-Up

Xing Learner

Thursday, June 18, 2015

Introduction

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is now possible to collect a large amount of data about personal activity relatively inexpensively. These type of devices are part of the quantified self movement - a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks. 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.

The Human Activity Recognition dataset we use for this project is 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. The goal of this project is to predict the manner in which they did the exercise. This is the "classe" variable in the dataset. The aim is to select and build an optimal prediction model to recognize human activities (such as sitting-down and standing-up) on 20 test cases.

Data Preprocessing

The training data and the testing data for this project are downloaded from the course website of Coursera.

```
mydata = read.csv("./pml-training.csv", na.strings = c("NA", ""))
dim(mydata)
```

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

```
missing = apply(mydata, 2, function(x) { sum(is.na(x)) })
mydata = mydata[, which(missing == 0)]
dim(mydata)
```

```
## [1] 19622    60
```

Note that the missing values are represented in the forms of both `NA` and blank `""` in the original datasets. To establish our model, we will first focus on the trainData, which consists of 19622 observations with 160 variables.
For simplicity, we are going to remove all variables with missing values. We now have 60 variables, including the one to be predicted, `classe`.

Additionally, after taking a glance of the dataset, we notice that the first six variables `user_name`, `raw_timestamp_part_1`, `raw_timestamp_part_2`, `cvtd_timestamp`, `new_window`, `num_window` are simply administrative parameters and seem to be irrelevant to our prediction model. Thus, we will delete these 6 variables as well. We will perform this truncation in our data split stage below.

```
# cross validation
library(caret)
set.seed(123)
inTrain = createDataPartition(y = mydata$classe, p = .7, list = FALSE)
```

```
training = mydata[inTrain, -(1:6)]
testing = mydata[-inTrain, -(1:6)]
dim(training)
```

```
## [1] 13737    54
```

We load the `caret` package, and for **cross validation** purpose, we partition our data into two parts, 70% for training and 30% for testing. Note now we have 54 variables only, and our training set has 13737 observations.

Model Fitting

```
table(mydata$classe)
```

```
##
##    A    B    C    D    E
## 5580 3797 3422 3216 3607
```

We see that `classe` is a factor variable which has 5 levels: A, B, C, D, and E. Our aim is to build a predictive model to predict the classe level for a given observation. This is a classification problem, and we are going to fit two models. The two commonly used classification algorithms **Random Forest** `rf` and **Gradient Boosting Machine** `gbm` algorithms are used. To reduce the risk of overfitting, a 6-fold cross validation is employed in `fitControl`, rather than the default 4-fold CV.

```
set.seed(666)
fitControl = trainControl(method = "cv", number = 6)
# gbm Model Fitting
set.seed(666)
gbmMod = train(classe ~ ., data = training, method = "gbm",
               trControl = fitControl, verbose = FALSE)
# Random Forest Model Fitting
set.seed(666)
rfMod = train(classe ~ ., data = training, method = "rf",
              trControl = fitControl)
```

The model fitting process takes quite a while.

Model Evaluation

In this section, we are going to take a look at the prediction performance both on the training data, and on the testing data in case of overfitting.

First, let's look at the consusion matrices for the `gbm` model:

```
gbmPredTrain = predict(gbmMod)
confusionMatrix(gbmPredTrain, training$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 3901   16    0    1    0
##          B    3 2625    5    6    8
##          C    0   13 2385   21    2
##          D    2    4    3 2224   14
##          E    0    0    3    0 2501
##
## Overall Statistics
##
##                Accuracy : 0.9926
##                  95% CI : (0.9911, 0.994)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9907
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9987   0.9876   0.9954   0.9876   0.9905
## Specificity            0.9983   0.9980   0.9968   0.9980   0.9997
## Pos Pred Value         0.9957   0.9917   0.9851   0.9898   0.9988
## Neg Pred Value         0.9995   0.9970   0.9990   0.9976   0.9979
## Prevalence             0.2843   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2840   0.1911   0.1736   0.1619   0.1821
## Detection Prevalence   0.2852   0.1927   0.1762   0.1636   0.1823
## Balanced Accuracy      0.9985   0.9928   0.9961   0.9928   0.9951
```

```
gbmPredTest = predict(gbmMod, newdata = testing)
confusionMatrix(gbmPredTest, testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1672   11    0    0    0
##          B    2 1116   11    3    2
##          C    0    8 1011   19    4
##          D    0    4    4  942    8
##          E    0    0    0    0 1068
##
## Overall Statistics
##
##                Accuracy : 0.9871
##                  95% CI : (0.9839, 0.9898)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9837
##  Mcnemar's Test P-Value : NA
```

```
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9988   0.9798   0.9854   0.9772   0.9871
## Specificity            0.9974   0.9962   0.9936   0.9967   1.0000
## Pos Pred Value         0.9935   0.9841   0.9702   0.9833   1.0000
## Neg Pred Value         0.9995   0.9952   0.9969   0.9955   0.9971
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2841   0.1896   0.1718   0.1601   0.1815
## Detection Prevalence   0.2860   0.1927   0.1771   0.1628   0.1815
## Balanced Accuracy      0.9981   0.9880   0.9895   0.9870   0.9935
```

We see that on the training data, `gbm` model prediction has an accuracy of 99.26% with Kappa value 99.07%. As we perform this model on the testing data, it has an accuracy of 98.71% with Kappa value 98.37%.

Next, let's look at the confusion matrices for the `rf` model:

```
rfPredTrain = predict(rfMod)
confusionMatrix(rfPredTrain, training$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 3906    0    0    0    0
##          B    0 2658    0    0    0
##          C    0    0 2396    0    0
##          D    0    0    0 2252    0
##          E    0    0    0    0 2525
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9997, 1)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   1.0000   1.0000   1.0000   1.0000
## Specificity            1.0000   1.0000   1.0000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Prevalence             0.2843   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2843   0.1935   0.1744   0.1639   0.1838
## Detection Prevalence   0.2843   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      1.0000   1.0000   1.0000   1.0000   1.0000
```

```
rfPredTest = predict(rfMod, newdata = testing)
confusionMatrix(rfPredTest, testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    3    0    0    0
##          B    0 1136    4    0    0
##          C    0    0 1022    2    0
##          D    0    0    0  961    0
##          E    0    0    0    1 1082
##
## Overall Statistics
##
##                Accuracy : 0.9983
##                  95% CI : (0.9969, 0.9992)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9979
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9974   0.9961   0.9969   1.0000
## Specificity            0.9993   0.9992   0.9996   1.0000   0.9998
## Pos Pred Value         0.9982   0.9965   0.9980   1.0000   0.9991
## Neg Pred Value         1.0000   0.9994   0.9992   0.9994   1.0000
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2845   0.1930   0.1737   0.1633   0.1839
## Detection Prevalence   0.2850   0.1937   0.1740   0.1633   0.1840
## Balanced Accuracy      0.9996   0.9983   0.9978   0.9984   0.9999
```

Thus, on the training data, the `rf` model prediction yields a confidence interval of accuracy of (99.97%, 100%), with Kappa value 100%. As we perform this model on the testing data, it still has an accuracy of 99.83% with Kappa value 99.79%.

The performance of `rf` model is better than that of the `gbm` model. Therefore, we would select the **Random Forest** model to predict our 20 testing cases.

Results

Let's take a look at our final model first, it uses 27 variables at each split.

```
rfMod$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: 27
##
##         OOB estimate of  error rate: 0.23%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3904    1    0    0    1 0.0005120328
## B    7 2647    4    0    0 0.0041384500
## C    0    2 2394    0    0 0.0008347245
## D    0    0   10 2242    0 0.0044404973
## E    0    0    0    6 2519 0.0023762376
```

Finally, we use the fitted **Random Forest** model to predict our 20 testing cases.

```
test20 = read.csv("./pml-testing.csv", na.strings = c("NA", ""))
test20 = test20[, which(missing == 0)]
pred20 = predict(rfMod, newdata = test20[-(1:6)])
data.frame(pred20)
```

```
##    pred20
## 1       B
## 2       A
## 3       B
## 4       A
## 5       A
## 6       E
## 7       D
## 8       B
## 9       A
## 10      A
## 11      B
## 12      C
## 13      B
## 14      A
## 15      E
## 16      E
## 17      A
## 18      B
## 19      B
## 20      B
```

All of the 20 prediction results turn out to be correct after submitting onto the PML course project grading platform.