Predict Activities with Sensor Data

MOOCJJC

January 22, 2017

Summary

This a coursework project to create a model for predictiong of the type of activities by using collected sensor data and other variables. In each training data point, the activity type has been specified, therefore it is a supervised learning problem. The training data set will be partitioned into 60 to 40 ratio for a training set and a validation set. The training set will be use to create the model and the validation set will provide an insight on how well the model works. If necessary, the validation set will also be used to optimized the parameters in the models.

There is no activity type in the test dataset. The best model will be used to predict the activity types in the twenty test data, and the results will be submitted through coursera website.

Exploratory Data Study

Below command load the raw training and testing data into the system:

```
library(caret)
library(rpart)
library(rpart.plot)
library(randomForest)
training_raw <- read.csv("pml-training.csv", na.string = c("NA", "", "#DIV/0!"))
testing_raw <- read.csv("pml-testing.csv", na.string = c("NA", "", "#DIV/0!"))
```

We used `str()` function in console to look at the data first (the results were shown in this manuscript because of limited space). It was found that there are many variables that are without data (occupied by NA). In next steps, we will tidy up the datasets and separate the training data for model builing and validation.

Data Cleaning

Remove predictors with more than 50% `NA`

Most likely the predictors have many `NA` values cannot contribute the prediction well, so removing them may simplify the model as well as the computation. The selection of 50% is kind of arbitrary at this moment. There may be some mechansim for choosing this threshold. The following commands selected the variables with less than 50% `NA` as predictors for the model building.

```
idx_select <- integer()
for (i in 1:length(training_raw)) {
    if (sum(is.na(training_raw[, i]))/nrow(training_raw) <= 0.5) {
        idx_select <- c(idx_select, i)
    }
}

training_raw1 <- training_raw[, idx_select]
```

Remove predictors with near Zero variance

It is also a common practice to remove the variables with zero or near zero variances because they are more like a constant and contribute little to the models. But we can expect there is not many of this kind variables for sensor data.

```
idx_n0var <- nearZeroVar(training_raw1, saveMetrics = TRUE)
training_raw2 <- training_raw1[, idx_n0var$nzv == FALSE]
```

Remove index column and duplicated timestamp

The first column, sample index, and the text-formatted timestamp was also removed. Please note the timestamp information has already included the other two timestamp columns represented by integers. Intermediate variables are also removed.

```
training_clean <- training_raw2[, -which(names(training_raw2) %in% c("X", "cvtd_timestamp"))]
rm(training_raw1)
rm(training_raw2)
```

Remove unnecessary predictors in testing set

To match the predictors between testing set and training set, the unnecessary predictors in testing set are also removed:

```
testing_clean <- testing_raw[, head(names(training_raw), -1)]
```

Data Partition for traing set

The following commands separated the raw training set into two parts: one for modeling building, `training_model`; the other for validation `val_model` with 60 to 40 ratio.

```
idx_inTrain <- createDataPartition(training_clean$classe, p = 0.6, list = FALSE)
training_model <- training_clean[idx_inTrain, ]
val_model <- training_clean[-idx_inTrain, ]
dim(training_model)
```

```
## [1] 11776    57
```

```
dim(val_model)
```

```
## [1] 7846   57
```

Prediction Models

1. Decision Trees

```
set.seed(100)
treeMod <- rpart(classe ~ ., method = "class", data = training_model)
treePred <- predict(treeMod, newdata = val_model, type = "class")
confusionMatrix(treePred, val_model$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2107  136   14   21   22
```

```
##          B   55 1106   46   12   24
##          C    8   59 1182   68   97
##          D   54  117   34 1025  105
##          E    8  100   92  160 1194
##
## Overall Statistics
##
##                Accuracy : 0.843
##                  95% CI : (0.8347, 0.851)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8013
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9440   0.7286   0.8640   0.7970   0.8280
## Specificity            0.9656   0.9784   0.9642   0.9527   0.9438
## Pos Pred Value         0.9161   0.8898   0.8359   0.7678   0.7683
## Neg Pred Value         0.9775   0.9376   0.9711   0.9599   0.9606
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2685   0.1410   0.1507   0.1306   0.1522
## Detection Prevalence   0.2931   0.1584   0.1802   0.1702   0.1981
## Balanced Accuracy      0.9548   0.8535   0.9141   0.8749   0.8859
```

According to an internet post [1], there is problem when passing "class" mentod to `train` function, so `rpart` function is used instead.

From the confusionMatrix, we can find that though the overall accuracy is not bad, `82.4%`, there are still a lot of miscategorized prediction with Decision Tree Model for `val_model`

2. Random Forest

```
set.seed(101)
rfMod <- randomForest(classe ~ ., data = training_model)
rfPred <- predict(rfMod, newdata = val_model, type = "class")
confusionMatrix(rfPred, val_model$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2232    0    0    0    0
##          B    0 1518    6    0    0
##          C    0    0 1362    1    0
##          D    0    0    0 1282    2
##          E    0    0    0    3 1440
##
## Overall Statistics
##
##                Accuracy : 0.9985
##                  95% CI : (0.9973, 0.9992)
##     No Information Rate : 0.2845
```

```
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9981
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   1.0000   0.9956   0.9969   0.9986
## Specificity            1.0000   0.9991   0.9998   0.9997   0.9995
## Pos Pred Value         1.0000   0.9961   0.9993   0.9984   0.9979
## Neg Pred Value         1.0000   1.0000   0.9991   0.9994   0.9997
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1935   0.1736   0.1634   0.1835
## Detection Prevalence   0.2845   0.1942   0.1737   0.1637   0.1839
## Balanced Accuracy      1.0000   0.9995   0.9977   0.9983   0.9991
```

The Random Forest Model has an impressive accuracy in the `val_model` dataset, `99.9%`. Because we didn't use the `val_model` to optimize the model parameters and the sample sizes for both `training_model` and `val_model` are not very small, we can expect we will get a good prediction on the test dataset.

3. Generalized Boosted Regression Models

```
set.seed(102)
fitControl <- trainControl(## 10-fold CV
                           method = "repeatedcv",
                           number = 10,
                           ## repeated one time only
                           repeats = 1)
gbmMod <- train(classe ~ ., data = training_model, method = "gbm"
                , verbose = FALSE, trControl = fitControl)
gbmPred <- predict(gbmMod, newdata = val_model)
confusionMatrix(gbmPred, val_model$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2227    3    0    0    0
##          B    5 1513    3    0    0
##          C    0    2 1358    3    0
##          D    0    0    7 1274    7
##          E    0    0    0    9 1435
##
## Overall Statistics
##
##                Accuracy : 0.995
##                  95% CI : (0.9932, 0.9965)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9937
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
```

```
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9978   0.9967   0.9927   0.9907   0.9951
## Specificity            0.9995   0.9987   0.9992   0.9979   0.9986
## Pos Pred Value         0.9987   0.9947   0.9963   0.9891   0.9938
## Neg Pred Value         0.9991   0.9992   0.9985   0.9982   0.9989
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2838   0.1928   0.1731   0.1624   0.1829
## Detection Prevalence   0.2842   0.1939   0.1737   0.1642   0.1840
## Balanced Accuracy      0.9986   0.9977   0.9960   0.9943   0.9969
```

The "Generalized Boosted Regression" model also has a very high accuracy, 99.5%. As similar to the "Random Forest" model, we didn't used the `val_model` to optimize the model parameters, so we can expect good accuracy of this model in test dataset.

Predict the results on test set for Quiz

The Random Forest Model was selected for the test dataset for Quiz question submission because it has the best performance among the three models we have used.

```
testPredict <- predict(rfMod, testing_clean)
testPredict
```

```
##  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
```

Conclusion

Three models, "Decision Trees", "Random Forest", and "Generalized Boosted Regression Model", have been used to predict the activity types by using the labeled sensor data. "Random Forest" model has the best performance in the `val_model` data (40 % of the training data). The accuracy of "Generalized Boosted Regression" model is just slightly lower than "Random Forest". This may be caused by the randomness of the modeling processes. The significantness of this difference is not discussed in this report.

References

1. [Internet]. Available from: http://stackoverflow.com/questions/27551863/r-caret-package-rpart-constructing-a-classification-