Predicting Manner of Barbell Exercises Using Data from Activity Tracking Devices

Chan Ching Chuen

2023-08-07

Executive Summary

In this report, we attempt to build a machine learning model to predict the manner of barbell exercise based on data collected by activity tracking devices, as part of the course work requirement of the Practical Machine Learning course. We are given a set of training data collected, for which we split into data set for training a model and the rest for validation purpose. Exploratory analysis and data cleaning were performed, and various models were fitted to achieve the best result. Considering accuracy, run time and requirements for cross-validations, we picked a random forest model which achieved over 99% accuracy on the validation data set. In the end we applied the model to a testing set of 20 problems, and discussed improvements and future works.

The Data Set, Exploratory Analysis and Data Cleaning

The data set involve asking participants to perform dumbbell exercise in different ways while wearing tracking devices: exactly according to the specification (Class A), and 4 other fashions according to common mistakes (Class B to E, corresponding to "classe" variable in the data set). That is, this problem is to identify the quality of the exercise performed, rather than focus on the duration or quantity of the exercise.

Due to lack of information on meaning of variables (the link to the paper and data on the internet were no longer valid), it was difficult to form intuitive views on relative importance of variables. We will try to follow common sense in data cleaning.

For example, cursory viewing of the data showed that there were quite a lot of missing or NA data field; upon closer inspection, those fields were mostly missing or NA except for a small percentage of observations; all of these observations corresponded to the "new_window" variable taking the "yes" value, which constituted around 2% of total observations. It would not be practical to impute values for the rest of 98% observations by extrapolating from such small samples. Hence we decided to delete all those variables with a lot of missing values, and also remove all observations with "new_window" equal to "yes".

We also removed variables corresponding to identity of participants and the time stamps for the activities - while it is unlikely that the identity of participants or the time of the exercise could improve classifications, it is also probably not the intention for the prediction to utilize such knowledge. In practical classification setting one would want to make accurate classification relying on the data from tracking devices, rather than knowing who is performing the exercise or when.

After this cleaning, there were 54 variables remaining. We also checked that there were no more missing values:

```r
print(dim(newtrain))
```

```
## [1] 19216    54
```

```r
#check that there are no more NAs and missing entries
```

```r
missingvalue <- is.na(newtrain)

print(sum(missingvalue))
```

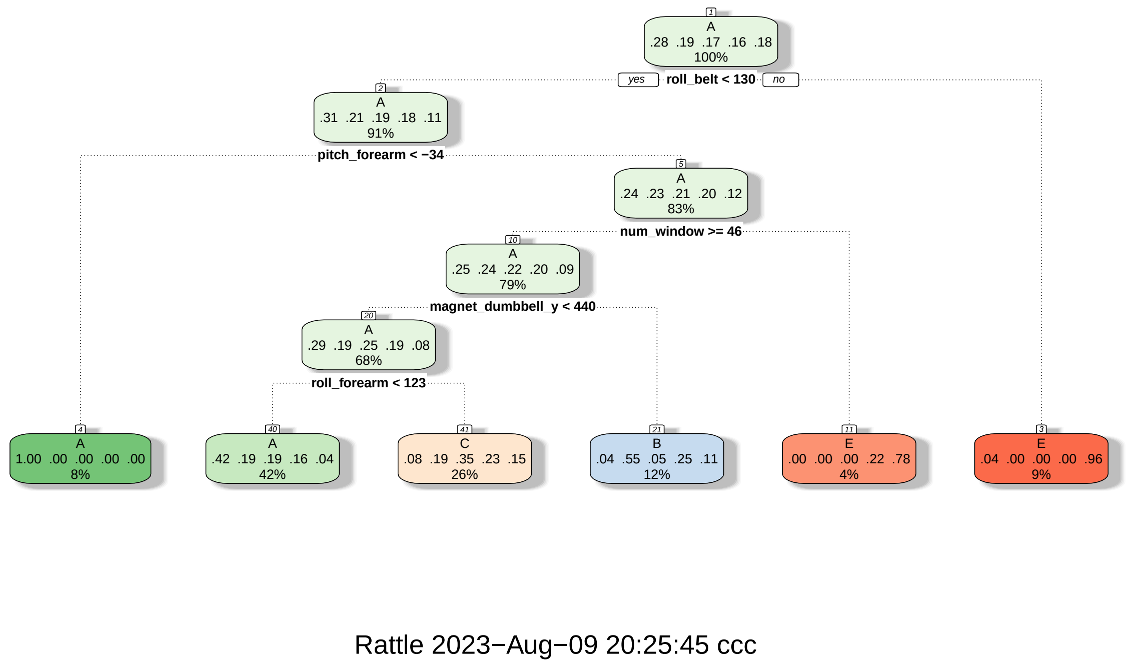
```
## [1] 0
```

We split the data set into training (70% of observations) and validation sets (30%) randomly. To gain some insight on whether some variables were dominating in predictions, we fitted a recursive partitioning and regression tree using the rpart method in the caret package:

```r
#split the training set into train / validation, 70/30
#set random seed for reproducibility

set.seed(721)

trainobs <-createDataPartition(newtrain$classe, p=0.7, list = FALSE)

traindata <- newtrain[trainobs,]
testdata <- newtrain[-trainobs,]

#set random seed for reproducibility
set.seed(1997)

# Set the hyperparameters for the training model
trainCtrl <- trainControl(method="repeatedcv",
                          number=10,
                          repeats=3,
                          savePredictions="final",
                          classProbs = TRUE)

# Train the rpart model
rpart_model <- train(classe ~ ., data=traindata,
                     method="rpart", trControl=trainCtrl)

#evaluate the model of validation data
pred_rpart <- predict(rpart_model, newdata=testdata)
accuracy_rpart <- mean(pred_rpart == testdata$classe)

# Print the accuracy of the model
print(accuracy_rpart)
```

```
## [1] 0.5240326
```

```r
fancyRpartPlot(rpart_model$finalModel)
```



The resulting tree is shown in the plot. The accuracy of this model was only around 50%, which seemed to indicate that linear method may not work very well.

Fitting Models

To verify our guess of non-linearity of the classification, we fitted Linear Discrimination Analysis (LDA) and Quadratic Discrimination Analysis (QDA) models, using the same train control parameters as in the rpart model. We pre-processed the data by centering and scaling the variables, observing that there were quite big variation of magnitude of across variables:

```r
#pre-process data with center and scale

preproc_train <- preProcess(newtrain, method=c("center","scale"))

traindata <- predict(preproc_train,newtrain[trainobs,])

testdata <- predict(preproc_train,newtrain[-trainobs,])

#train LDA
set.seed(1997)

lda_model <- train(classe ~., data=traindata, method="lda", trControl=trainCtrl)

pred_lda <- predict(lda_model, newdata=testdata)

accuracy_lda <- mean(pred_lda == testdata$classe)
print(accuracy_lda)
```

```
## [1] 0.7173347
```

```r
#train QDA
set.seed(1997)

qda_model <- train(classe ~., data=traindata, method="qda", trControl=trainCtrl)
```

```r
pred_qda <- predict(qda_model, newdata=testdata)

accuracy_qda <- mean(pred_qda == testdata$classe)
print(accuracy_qda)
```

```
## [1] 0.9019608
```

The QDA method have an accuracy of 90% on the validation set, a significant improvement on rpart and LDA (71%). It seemed non-linear or non-parametric methods would work better for this problem.

To explore ways to improve accuracy to above 90%, we fitted a Support Vector Machine (svm) model, and find that the resulting accuracy is lower than that of QDA at 81% (using the same train control parameters for comparability):

```r
#svm
set.seed(1997)

st_time_svm <- Sys.time()

svm_model <- train(classe ~., data=traindata, method="svmRadial", trControl=trainCtrl)

pred_svm <- predict(svm_model, newdata=testdata)

accuracy_svm <- mean(pred_svm == testdata$classe)

end_time_svm <- Sys.time()

elapsed_time_svm <- end_time_svm - st_time_svm

print(accuracy_svm)
```

```
## [1] 0.8138123
```

```r
print(elapsed_time_svm)
```

```
## Time difference of 29.2638 mins
```

Rather than fine-tuning the svm parameters, we tried a Random Forest (rf) model using the same train control parameters, and we get an accuracy of 99.7%:

```r
#random forest
set.seed(1997)

st_time_rf <- Sys.time()

rf_model <- train(classe ~., data=traindata, method="rf", trControl=trainCtrl)

pred_rf <- predict(rf_model, newdata=testdata)

accuracy_rf <- mean(pred_rf == testdata$classe)

end_time_rf <- Sys.time()

elapsed_time_rf <- end_time_rf - st_time_rf

print(accuracy_rf)
```

```
## [1] 0.9963561
```

```r
print(elapsed_time_rf)
```

```
## Time difference of 29.63796 mins
```

While the rf model is harder to interpret, the advantage on accuracy seems overwhelming.

Considerations on Cross-Validation

Before deciding which model to use for the prediction, we would like to explore the impact of cross-validation. We fitted another random forest model, this time doubling the number of partitions from 10 to 20:

```r
#more cross validation for rf

trainCtrl <- trainControl(method="repeatedcv",
                          number=20,
                          repeats=3,
                          savePredictions="final",
                          classProbs = TRUE)

set.seed(1997)

st_time_rf1 <- Sys.time()

rf1_model <- train(classe ~., data=traindata, method="rf", trControl=trainCtrl)

pred_rf1 <- predict(rf1_model, newdata=testdata)

accuracy_rf1 <- mean(pred_rf1 == testdata$classe)

end_time_rf1 <- Sys.time()

elapsed_time_rf1 <- end_time_rf1 - st_time_rf1

print(accuracy_rf1)
```

```
## [1] 0.9965296
```

```r
print(elapsed_time_rf1)
```

```
## Time difference of 1.030791 hours
```

The accuracy was almost the same with the previous rf model, with much longer run time. We decided that the rf model with 10 partition in cross-validation worked good enough.

Prediction on Testing Set

Feeding the testing set of data to our rf model yield the following predictions:

```r
#pre-process testing data
testing_preproc <- predict(preproc_train, testing)

pred_final <- predict(rf_model, testing_preproc)

print(pred_final)
```

```
##  [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
```

Conclusions and Future Works

We tried and fitted various models on the data set and successfully identified a model that had high accuracy on the validation set, and produced predictions on the testing data as required by the course work.

Due to lack of documentation of the meanings of the data field, there was little we can do to gain intuitive insights on relative importance of the variables; due to the same reason, the interpretation for predictions from the models were difficult to obtain. To improve on future work, we will need to get our hand on detailed information about how the data were collected and precise meanings of the variables.

It was a little surprising that svm performed relatively poorly comparing to qda model. It would be insightful to dig deeper on the reason causing this and deepen our understanding on the relative strength of the models in similar classification problems.