Human Activity Recognition using R

2023-05-02

Abstract

Human Activity Recognition - HAR - has emerged as a key research area in the last years. There are many potential applications for HAR, like: elderly monitoring, life log systems for monitoring energy expenditure and for supporting weight-loss programs, and digital assistants for weight lifting exercises. In the present article we will create and compare several models to create a prediction function for a dataset with 5 classes (sitting-down, standing-up, standing, walking, and sitting) collected on 8 hours of activities of 4 healthy subjects. See *Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements*. Full link in the *Reference* section below.

Data

Download

First we download the training and testing data:

```
setwd("/Users/cjr/coursera/DataScience_With_R/")
if (! file.exists("pml-training.csv")) {
  download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv", "pml-training.cs
}
if (! file.exists("pml-testing.csv")) {
  download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv", "pml-testing.csv
}
```

Now we can read it. We convert the strings representing missing values to the **NA** value.

```
setwd("/Users/cjr/coursera/DataScience_With_R/")
training <- read.csv("pml-training.csv", na.strings=c("NA", "", "#DIV/0!"))
testing <- read.csv("pml-testing.csv", na.strings=c("NA", "", "#DIV/0!"))
```

What we seek to predict is the class of training, which is represented by ['A', ..., 'E'] in the variable **classe**.

```
barplot(table(training$classe), main="Occurrences of the different classes of training exercises", xlab=
```

We see a difference in the number of occurrences. As all classes are represented by some thousands of samples, the degree to which the dataset is skewed is unlikely to adversely affect the outcome.

Remove columns not relevant for prediction

Let's look at the columns of the loaded data:

```
names(training)
```

```
##   [1] "X"                        "user_name"
##   [3] "raw_timestamp_part_1"     "raw_timestamp_part_2"
##   [5] "cvtd_timestamp"           "new_window"
##   [7] "num_window"               "roll_belt"
##   [9] "pitch_belt"               "yaw_belt"
##  [11] "total_accel_belt"         "kurtosis_roll_belt"
##  [13] "kurtosis_picth_belt"      "kurtosis_yaw_belt"
##  [15] "skewness_roll_belt"       "skewness_roll_belt.1"
##  [17] "skewness_yaw_belt"        "max_roll_belt"
##  [19] "max_picth_belt"           "max_yaw_belt"
##  [21] "min_roll_belt"            "min_pitch_belt"
##  [23] "min_yaw_belt"             "amplitude_roll_belt"
##  [25] "amplitude_pitch_belt"     "amplitude_yaw_belt"
##  [27] "var_total_accel_belt"     "avg_roll_belt"
##  [29] "stddev_roll_belt"         "var_roll_belt"
##  [31] "avg_pitch_belt"           "stddev_pitch_belt"
##  [33] "var_pitch_belt"           "avg_yaw_belt"
##  [35] "stddev_yaw_belt"          "var_yaw_belt"
##  [37] "gyros_belt_x"             "gyros_belt_y"
##  [39] "gyros_belt_z"             "accel_belt_x"
##  [41] "accel_belt_y"             "accel_belt_z"
##  [43] "magnet_belt_x"            "magnet_belt_y"
```

```
##  [45] "magnet_belt_z"            "roll_arm"
##  [47] "pitch_arm"                "yaw_arm"
##  [49] "total_accel_arm"          "var_accel_arm"
##  [51] "avg_roll_arm"             "stddev_roll_arm"
##  [53] "var_roll_arm"             "avg_pitch_arm"
##  [55] "stddev_pitch_arm"         "var_pitch_arm"
##  [57] "avg_yaw_arm"              "stddev_yaw_arm"
##  [59] "var_yaw_arm"              "gyros_arm_x"
##  [61] "gyros_arm_y"              "gyros_arm_z"
##  [63] "accel_arm_x"              "accel_arm_y"
##  [65] "accel_arm_z"              "magnet_arm_x"
##  [67] "magnet_arm_y"             "magnet_arm_z"
##  [69] "kurtosis_roll_arm"        "kurtosis_picth_arm"
##  [71] "kurtosis_yaw_arm"         "skewness_roll_arm"
##  [73] "skewness_pitch_arm"       "skewness_yaw_arm"
##  [75] "max_roll_arm"             "max_picth_arm"
##  [77] "max_yaw_arm"              "min_roll_arm"
##  [79] "min_pitch_arm"            "min_yaw_arm"
##  [81] "amplitude_roll_arm"       "amplitude_pitch_arm"
##  [83] "amplitude_yaw_arm"        "roll_dumbbell"
##  [85] "pitch_dumbbell"           "yaw_dumbbell"
##  [87] "kurtosis_roll_dumbbell"   "kurtosis_picth_dumbbell"
##  [89] "kurtosis_yaw_dumbbell"    "skewness_roll_dumbbell"
##  [91] "skewness_pitch_dumbbell"  "skewness_yaw_dumbbell"
##  [93] "max_roll_dumbbell"        "max_picth_dumbbell"
##  [95] "max_yaw_dumbbell"         "min_roll_dumbbell"
##  [97] "min_pitch_dumbbell"       "min_yaw_dumbbell"
##  [99] "amplitude_roll_dumbbell"  "amplitude_pitch_dumbbell"
## [101] "amplitude_yaw_dumbbell"   "total_accel_dumbbell"
## [103] "var_accel_dumbbell"       "avg_roll_dumbbell"
## [105] "stddev_roll_dumbbell"     "var_roll_dumbbell"
## [107] "avg_pitch_dumbbell"       "stddev_pitch_dumbbell"
## [109] "var_pitch_dumbbell"       "avg_yaw_dumbbell"
## [111] "stddev_yaw_dumbbell"      "var_yaw_dumbbell"
## [113] "gyros_dumbbell_x"         "gyros_dumbbell_y"
## [115] "gyros_dumbbell_z"         "accel_dumbbell_x"
## [117] "accel_dumbbell_y"         "accel_dumbbell_z"
## [119] "magnet_dumbbell_x"        "magnet_dumbbell_y"
## [121] "magnet_dumbbell_z"        "roll_forearm"
## [123] "pitch_forearm"            "yaw_forearm"
## [125] "kurtosis_roll_forearm"    "kurtosis_picth_forearm"
## [127] "kurtosis_yaw_forearm"     "skewness_roll_forearm"
## [129] "skewness_pitch_forearm"   "skewness_yaw_forearm"
## [131] "max_roll_forearm"         "max_picth_forearm"
## [133] "max_yaw_forearm"          "min_roll_forearm"
## [135] "min_pitch_forearm"        "min_yaw_forearm"
## [137] "amplitude_roll_forearm"   "amplitude_pitch_forearm"
## [139] "amplitude_yaw_forearm"    "total_accel_forearm"
## [141] "var_accel_forearm"        "avg_roll_forearm"
## [143] "stddev_roll_forearm"      "var_roll_forearm"
## [145] "avg_pitch_forearm"        "stddev_pitch_forearm"
## [147] "var_pitch_forearm"        "avg_yaw_forearm"
## [149] "stddev_yaw_forearm"       "var_yaw_forearm"
## [151] "gyros_forearm_x"          "gyros_forearm_y"
```

```
## [153] "gyros_forearm_z"          "accel_forearm_x"
## [155] "accel_forearm_y"          "accel_forearm_z"
## [157] "magnet_forearm_x"         "magnet_forearm_y"
## [159] "magnet_forearm_z"         "classe"
```

The columns:

- X
- user_name
- raw_timestamp_part_1
- raw_timestamp_part_2
- cvtd_timestamp
- new_window
- num_window

all serve to identify the recording and are not part of the measurement of actions.

We can therefor remove them. They have the first seven indexes.

```
training <- training[,-c(1:7)]
testing <- testing[,-c(1:7)]
dim(training)
```

```
## [1] 19622   153
```

Replace unlikely predictors in the data

We remove columns with near zero variance and columns with mostly (> 50%) NA values from the training set. Then we split our training set into two parts: one for actually training and the second, "out of sample" part, for testing the training result.

```
training <- training[,-nearZeroVar(training)]
training <- training[, -which(colMeans(is.na(training)) > 0.5)]
```

Creating an out-of-sample part

If we use all available training data for actually training the model, we will have no independent information on how well the model actually is, that is, it might be overfitting.

To get an informed idea of how well the model we create is doing we set aside a part of the data and use it for validation only. Here we use 70% of the data for training and 30% for validation because such a division conforms to a good heuristic.

```
train_rows <- createDataPartition(training$classe,p=0.7,list=FALSE)
trainPart <- training[train_rows,]
testPart <- training[-train_rows,]
```

Cross-validation

For the in-sample-part we set cross-validation with 5 folds. This means that different partitions of the data are made where a part used for validation will in the next partitioning be used for training etc. This serves to diminish the influence of outliers in specific parts of the data.

```
trctrl <- trainControl(method = "cv", number = 10)
```

Modeling the data

We will model the data using three algorithms:

- (Linear) Support Vector Machine
- Random Forst
- Stochastic Gradient Boosting

Support Vector Machine

```
# Support Vector Machine
svm <- train(classe ~., data=trainPart, method = "svmLinear", trControl=trctrl, verbose=FALSE)
svm
```

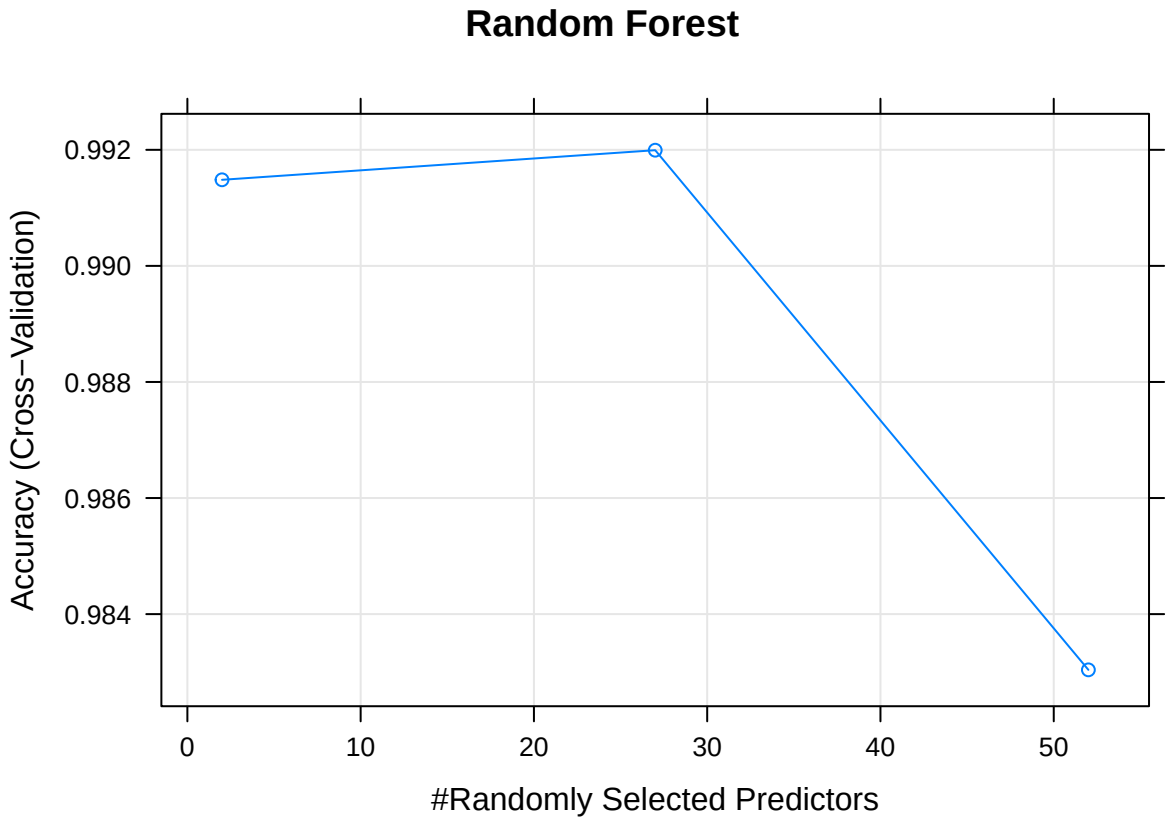
```
## Support Vector Machines with Linear Kernel
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 12363, 12365, 12365, 12363, 12364, 12364, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.7873664  0.729658
##
## Tuning parameter 'C' was held constant at a value of 1
```

Random Forest

```
# Random forest
rf <- train(classe ~., data=trainPart, method = "rf", trControl=trctrl, verbose=FALSE)
rf
```

```
## Random Forest
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 12364, 12364, 12363, 12363, 12364, 12363, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9914836  0.9892258
##   27    0.9919935  0.9898707
##   52    0.9830398  0.9785396
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 27.
```

```
plot(rf, main="Random Forest")
```



We see that the influence of adding additional parameters to make the prediction steeply decreases before the number of considered parameters is thirty.

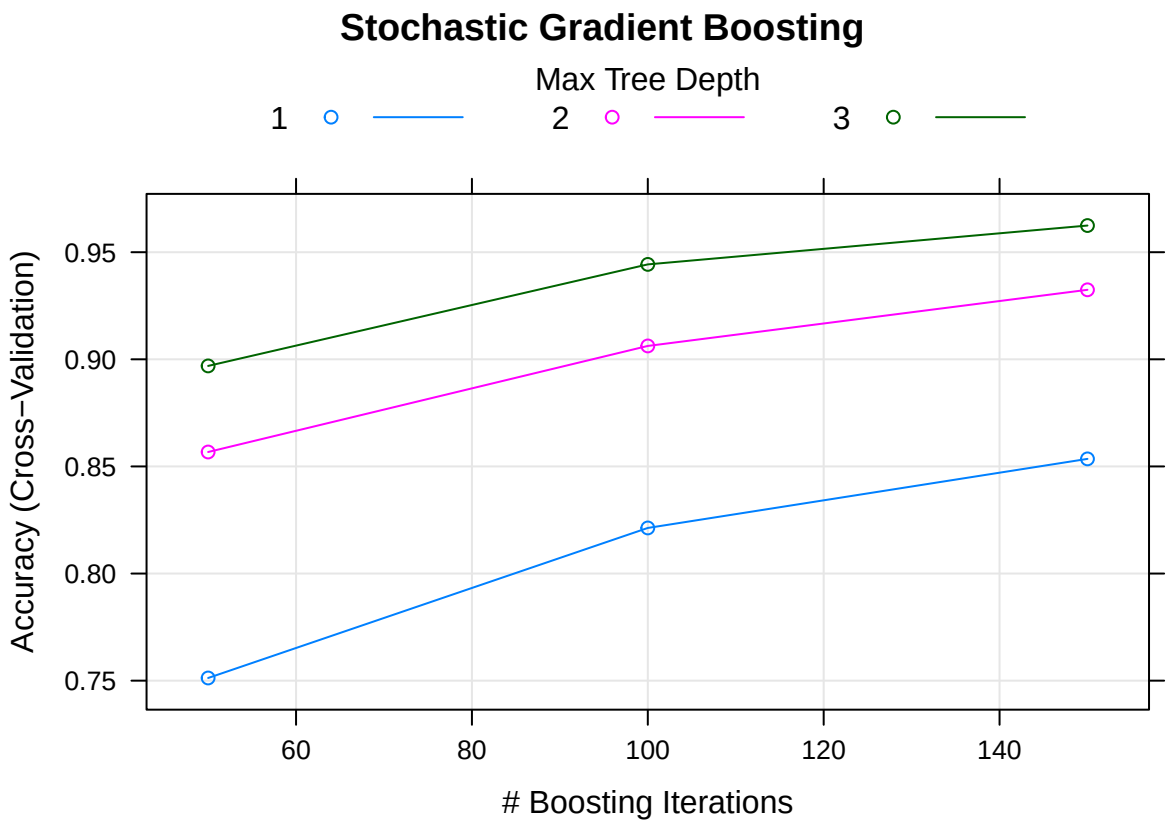
Stochastic Gradient Boosting

```
# GBM
gbm <- train(classe ~., data=trainPart, method = "gbm", trControl=trctrl, verbose=FALSE)
gbm
```

```
## Stochastic Gradient Boosting
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 12362, 12362, 12363, 12362, 12362, 12363, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7512497  0.6844415
##   1                  100      0.8212802  0.7738219
##   1                  150      0.8535282  0.8146153
##   2                   50      0.8567316  0.8184523
##   2                  100      0.9062333  0.8813508
##   2                  150      0.9324414  0.9145165
##   3                   50      0.8969157  0.8694620
```

```
##   3                  100      0.9443098  0.9295428
##   3                  150      0.9624370  0.9524815
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150, interaction.depth =
##  3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
plot(gbm, main="Stochastic Gradient Boosting")
```



We see that a depth of 3 gives higher accuracy than a depth < 3.

Analysing the results

We now make a prediction with the out-of-sample part of the data and create a confusion matrix from that prediction and the reference values of that part.

Confusion matrix SVM

```
predsvm <- predict(svm, testPart)
confusionMatrix(predsvm, as.factor(testPart$classe))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
```

```
##          A 1511  153   92   68   50
##          B   33  786  111   36  121
##          C   58   80  781  113   79
##          D   66   29   23  694   51
##          E    6   91   19   53  781
##
## Overall Statistics
##
##                Accuracy : 0.7737
##                  95% CI : (0.7628, 0.7843)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7124
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9026   0.6901   0.7612   0.7199   0.7218
## Specificity            0.9138   0.9366   0.9321   0.9657   0.9648
## Pos Pred Value         0.8063   0.7231   0.7030   0.8042   0.8221
## Neg Pred Value         0.9594   0.9264   0.9487   0.9462   0.9390
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2568   0.1336   0.1327   0.1179   0.1327
## Detection Prevalence   0.3184   0.1847   0.1888   0.1466   0.1614
## Balanced Accuracy      0.9082   0.8133   0.8466   0.8428   0.8433
```

Confusion matrix RF

```
predrf <- predict(rf, testPart)
confusionMatrix(predrf, as.factor(testPart$classe))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1671   10    0    0    0
##          B    2 1121    0    0    0
##          C    1    8 1021   13    0
##          D    0    0    5  951    1
##          E    0    0    0    0 1081
##
## Overall Statistics
##
##                Accuracy : 0.9932
##                  95% CI : (0.9908, 0.9951)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9914
##
##  Mcnemar's Test P-Value : NA
```

```
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9982   0.9842   0.9951   0.9865   0.9991
## Specificity            0.9976   0.9996   0.9955   0.9988   1.0000
## Pos Pred Value         0.9941   0.9982   0.9789   0.9937   1.0000
## Neg Pred Value         0.9993   0.9962   0.9990   0.9974   0.9998
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2839   0.1905   0.1735   0.1616   0.1837
## Detection Prevalence   0.2856   0.1908   0.1772   0.1626   0.1837
## Balanced Accuracy      0.9979   0.9919   0.9953   0.9926   0.9995
```

Confusion matrix GBM

```
predgbm <- predict(gbm, testPart)
confusionMatrix(predgbm, as.factor(testPart$classe))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1640   41    0    0    2
##          B   24 1052   23    6    2
##          C    8   43  979   24    6
##          D    0    0   23  929    6
##          E    2    3    1    5 1066
##
## Overall Statistics
##
##                Accuracy : 0.9628
##                  95% CI : (0.9576, 0.9675)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9529
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9797   0.9236   0.9542   0.9637   0.9852
## Specificity            0.9898   0.9884   0.9833   0.9941   0.9977
## Pos Pred Value         0.9745   0.9503   0.9236   0.9697   0.9898
## Neg Pred Value         0.9919   0.9818   0.9903   0.9929   0.9967
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2787   0.1788   0.1664   0.1579   0.1811
## Detection Prevalence   0.2860   0.1881   0.1801   0.1628   0.1830
## Balanced Accuracy      0.9847   0.9560   0.9688   0.9789   0.9915
```

Conclusion

Best model and out-of-sample error

The Random Forest model gave the highest accuracy so we select that for the actual prediction.

With an accuracy on the out-of-sample data of 0.993 we have an out-of-sample error of 1 - 0.993 = 0.007 or 0.7% with a 95% confidence interval.

Predicting values for the test set

Finally, generate a result for the test set. We use the *Random Forest* algorithm as that gave the best outcome.

```
test_values <- predict(rf,testing)
test_values
```

```
##  [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
```

References

Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. Qualitative Activity Recognition of Weight Lifting Exercises. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human '13) . Stuttgart, Germany: ACM SIGCHI, 2013.