Predicting Exercise Activity Quality

Ozan Sayin

July 24, 2014

The goal of this project is to produce a machine learning algorithm that predicts the quality of a barbell lifting activity. "Quality" of activitiy was defined at 5 discrete levels regarding the manner the activity was carried out:

- ***Class A:*** Exactly according to the specification
- ***Class B:*** Throwing the elbows to the front
- ***Class C:*** Lifting the dumbbell only halfway
- ***Class D:*** Lowering the dumbbell only halfway
- ***Class E:*** Throwing the hips to the front

The training data where 6 participants were asked to perform the activity in all 5 ways included measurements from accelerometers on the belt, forearm, arm, and dumbell. The source of all the data for this project is at http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har).

The class prediction algorithm is the obtained with a Random Forest model after pre-processing the relevant predictors with PCA.

Building the Algorithm:

We start off by identifying variables in the data that have NAs and remove those predictors.

```
vars_train <- names(train_data)
vars_test <- names(test_data)

# Only choose predictors that don't have NAs for both train and test data
nonNA_vars_train <- vars_train[colSums(is.na(train_data)) == 0]
nonNA_vars_test <- vars_test[colSums(is.na(test_data)) == 0]
nonNA_vars <- intersect(nonNA_vars_test,nonNA_vars_train)

train_data <- subset(train_data, select = c(nonNA_vars,"classe"))
test_data <- subset(test_data, select = c(nonNA_vars,"problem_id"))
```

We further remove variables that contain time stamps and activity windows. These should not be related to how one performs the activity. After this step, we end up with 53 predictors.

```
train_data <- subset(train_data, select = c(-X,-raw_timestamp_part_1,-raw_timestamp_part_2,
-cvtd_timestamp,-new_window,-num_window))

test_data <- subset(test_data, select = c(-X,-raw_timestamp_part_1,-raw_timestamp_part_2,-c
vtd_timestamp,-new_window,-num_window))

train_data$user_name <- as.numeric(train_data$user_name)
test_data$user_name <- as.numeric(test_data$user_name)
```

Then we divide the train_data into the training set (where we build the model) and testing set (where we test our final model)

```
## Divide the training observations to training and testing data
inTrain <- createDataPartition(train_data$classe, p=0.6, list = FALSE)
training <- train_data[inTrain,]
testing <- train_data[-inTrain,]
```

It is likely that there is a sizeable amount of correletaion between the 53 accelerometer readings, so we first perform a PCA and keep the principal components that explain 98% of the variance in the training set. For the accuracy and efficiency of the random forest fit, it is important to have predictors with little-to-no correlation with a reduced dimensionality while preserving information.

```
## First, center and scale the data. Necessary for PCA!!
preObj <- preProcess(training[,-54],method=c("center","scale"))
train_preVars <- predict(preObj,training[,-54])
## Compute the principal components
pcaObj <- preProcess(train_preVars,method="pca",thresh=0.98)
train_PC <- predict(pcaObj,train_preVars)
```

After the PCA, we end up with 31 principal components that we can use as predictors.

```
train_PC <- cbind(train_PC,training$classe)
colnames(train_PC)[ncol(train_PC)] <- "classe"

## Fit a random forest model with cross validation
rfFit <- train(classe ~.,data=train_PC, method="rf",trControl=trainControl(method="cv"), pr
ox=T)

rfFit
```

```
## Random Forest
##
## 11776 samples
##    31 predictors
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
##
## Summary of sample sizes: 10599, 10598, 10597, 10599, 10597, 10598, ...
##
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa  Accuracy SD  Kappa SD
##   2     1         1      0.005        0.006
##   20    1         1      0.004        0.005
##   30    1         0.9    0.007        0.008
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

10-fold cross validation is used during the fitting of the random forest model.

```
rfFit$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry, proximity = ..1)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 2.56%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3325    6    8    6    3     0.00687
## B   44 2205   24    2    4     0.03247
## C    4   34 1991   19    6     0.03067
## D    8    2   86 1830    4     0.05181
## E    2   10   18   11 2124     0.01894
```

The expected ***out-of-sample error rate*** is ***2.556%***, which is estimated from the average of the classification error on the 10 different hold-out testing sample sets.

Let us now apply our model to the testing set to see how it performs and whether the out-of-sample error estimate we previously obtained from the 10 fold cross-validation seems realistic.

```
## Preprocess the test data in the same way as prior to building the model on the train set
test_preVars <- predict(preObj,testing[,-54])
test_PC <- predict(pcaObj,test_preVars)

## Predict with random forest
pred_class <- predict(rfFit,test_PC)

pred_result <- confusionMatrix(pred_class, testing$classe)
pred_result$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 2219   33    6    3    0
##          B    5 1459   27    1    1
##          C    4   18 1321   53   19
##          D    4    8   12 1224    9
##          E    0    0    2    5 1413
```

```
error_on_test <- 1 - pred_result$overall[1]
names(error_on_test) <- "Error Rate(%)"
100 * error_on_test
```

```
## Error Rate(%)
##         2.677
```

The error we obtained on the test set is very similar to our out-of-sample error estimate, and the learning algorithm seems to be working quite well!

Furthermore, one can get a more detailed insight by investigating the performance of the prediction algorithm on each seperate class.

```
pred_result$byClass
```

```
##          Sensitivity Specificity Pos Pred Value Neg Pred Value Prevalence
## Class: A      0.9942      0.9925         0.9814         0.9977     0.2845
## Class: B      0.9611      0.9946         0.9772         0.9907     0.1935
## Class: C      0.9656      0.9855         0.9336         0.9927     0.1744
## Class: D      0.9518      0.9950         0.9737         0.9906     0.1639
## Class: E      0.9799      0.9989         0.9951         0.9955     0.1838
##          Detection Rate Detection Prevalence Balanced Accuracy
## Class: A         0.2828               0.2882            0.9933
## Class: B         0.1860               0.1903            0.9779
## Class: C         0.1684               0.1803            0.9756
## Class: D         0.1560               0.1602            0.9734
## Class: E         0.1801               0.1810            0.9894
```