Predicting the quality of dumbbell curl exercises

Eusebio Rufian-Zilbermann

Weight Lifting Activity Recognition

Introduction

The goal of this project is to obtain an algorithm that can evaluate the manner in which a person has performed a dumbbell curl, whether correctly or falling into one of 4 common mistakes, using data from accelerometers on the belt, forearm, arm, and on the dumbbell itself. The source of the data is http://groupware.les.inf.puc-rio.br/har#weight_lifting_exercises and it was presented in:
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.
(a link to the document is available at the end of the indicated page)

Dataset exploration and analysis

The dataset contains the following types of information:

- Participant identifier (there were 6 different participants)
- Timestamps for beginning and end of each observation
- Statistical analysis data (observations grouped into windows, each with a window identifier and summary information that describes the distribution of values over that window)
- Sensor data These are all the variables that start with "pitch_", "roll_", "total_accel_", "yaw_" (spherical magnitudes) or end with "_x","_y" or "_z" (cartesian magtnitudes).
- Identifier for how the exercise was performed:
  A-Exactly according to the specification
  B-Throwing the elbows to the front
  C-lifting the dumbbell only halfway
  D-lowering the dumbbell only halfway
  E-throwing the hips to the front

Each observation corresponds to one 'rep' (a repetition of the curl exercise).

The sensor data will be used as predictors.

3 different sets of data were analyzed:

- Spherical magnitudes only. Having both Tait-Bryan angles and magnitude (spherical coordinates) and 3-Dimensional euclidean space x,y,z (cartesian magnitudes} is actually two ways of expressing the same information, and the information is redundant. Using just one set reduces the amount of data in the model, making it faster to train. The Spherical coordinates are easier to interpret intuitively and that is why I have chosen this set. The resulting dataset uses 16 predictors. Note: Another possibility for obtaining a smaller dataset would have been to apply principal component analysis but the results are likely less intuitive, that is why I didn't choose them.
- Full spherical and cartesian coordinates. Even though the information is redundant, keeping both spherical and cartesian coordinates can be an effective way to have covariates and solve underfit problems. The resulting dataset has 52 predictors.
- Cross-product cartesian features. One possibility for additional covariates are the cartesian xy, xz and yz cross-products. For models where underfit is a problem these can improve the resulting accuracy. The resulting dataset has 88 predictors

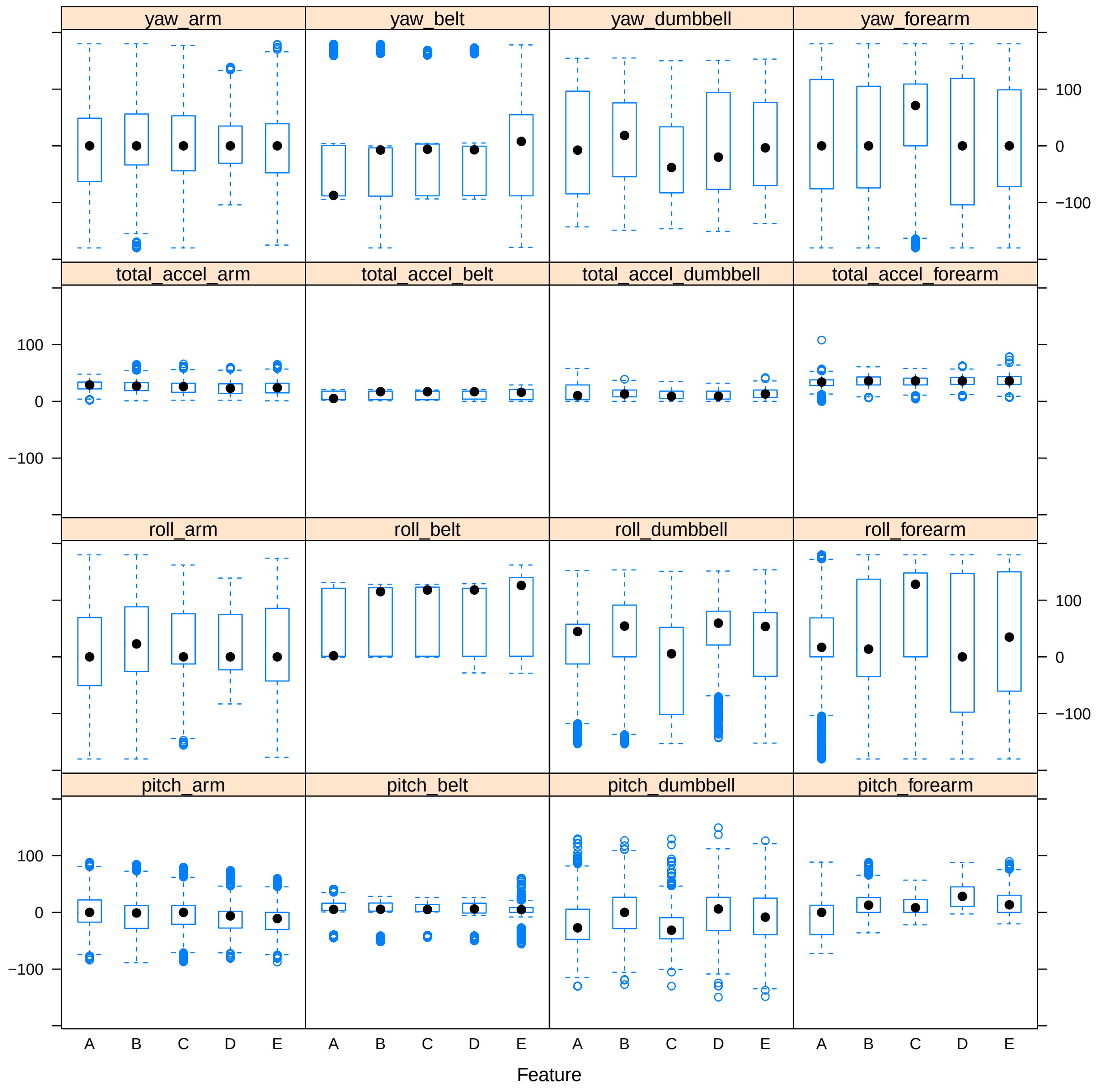
The Identifier for how the exercise was performed, classe, will be the outcome of the algorithm and it needs to be included in the datasets as well.

We want to get partitions that are varied and not skewed towards specific outcomes or participants so we will use classe and user_name for partitioning. We will use 60% of the data for training and 40% for testing. Testing data will not be used for refining the model so it is not necessary to further add a validation set. Cross-validation and Out of sample error estimation will be performed on the testing set.

The participant identifier is not used as a predictor in order to obtain a general algorithm that is not specific to an individual participant. The statistical analysis data must not be used as a predictor as it is specific to this dataset and not generalizable to future data.

To get an idea of what the data looks like, we can see the plot for the Tait-Bryan angles and magnitude features

```
require(caret)
plt<-featurePlot(x=pml[grepl("^roll_|^yaw_|^pitch_|^total_accel_",names(pml))],
                 y=pml$classe,plot="boxplot",layout=c(4,4))
print(plt)
```



Algorithm and results

We are looking for a categorical result therefore we need a classifier. The outcome is available therefore supervised learning techniques will be more effective. Several modeling techniques were explored: Logistic regression, neural networks and Random forests. In terms of computational requirements, random forests are the slowest, requiring a very significant amount of training time as the number of features grows. Neural networks were the fastest to train but the accuracy with these feature sets was lower than the other methods. Accuracy improved with the addition of features, suggesting that it may be possible to improve the model by adding new features (e.g., we could add squared cartesian magnitudes). Logistic regression was slightly slower than neural networks but they provided perfect accuracy even on the small feature set. Training the logistic regression on the larger feature set took less time than the random forests on the smaller set. We use cross-validation as a training option.

The percentual Out of sample error estimates obtained from cross-validation on the different models and feature sets are the following:

```r
round(100*(1-accuracies),2)
```

```
##              RandomForest Multinom LogisticRegression
## 16Predictors         1.22    51.64              52.27
## 52Predictors         0.90    26.20              27.05
## 88Predictors         1.03    16.92              17.93
```

Based on these results our recommendation is to apply Random Forest using the medium feature set (rfMod52f), or possibly the small feature set (rfMod16f) if training performance is a concern. Their confusion matrices are:

```r
rfConfMat16f
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2239    9    1    0    0
##          B    6 1440    9    3    1
##          C    0   18 1345   18   10
##          D    1    1   10 1325    5
##          E    0    1    0    3 1403
##
## Overall Statistics
##
##                Accuracy : 0.9878
##                  95% CI : (0.9851, 0.9901)
##     No Information Rate : 0.2862
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9845
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9969   0.9803   0.9853   0.9822   0.9887
## Specificity            0.9982   0.9970   0.9929   0.9974   0.9994
## Pos Pred Value         0.9956   0.9870   0.9669   0.9873   0.9972
## Neg Pred Value         0.9987   0.9955   0.9969   0.9963   0.9975
## Prevalence             0.2862   0.1872   0.1739   0.1719   0.1808
## Detection Rate         0.2853   0.1835   0.1714   0.1688   0.1788
## Detection Prevalence   0.2866   0.1859   0.1772   0.1710   0.1793
## Balanced Accuracy      0.9975   0.9886   0.9891   0.9898   0.9941
```

```
rfConfMat52f
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2242   16    0    0    0
##          B    3 1446   13    0    0
##          C    0    7 1351   21    2
##          D    0    0    1 1326    5
##          E    1    0    0    2 1412
##
## Overall Statistics
##
##                Accuracy : 0.991
##                  95% CI : (0.9886, 0.9929)
##     No Information Rate : 0.2862
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9886
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9982   0.9843   0.9897   0.9830   0.9951
## Specificity            0.9971   0.9975   0.9954   0.9991   0.9995
## Pos Pred Value         0.9929   0.9891   0.9783   0.9955   0.9979
## Neg Pred Value         0.9993   0.9964   0.9978   0.9965   0.9989
## Prevalence             0.2862   0.1872   0.1739   0.1719   0.1808
## Detection Rate         0.2857   0.1843   0.1721   0.1690   0.1799
## Detection Prevalence   0.2877   0.1863   0.1760   0.1697   0.1803
## Balanced Accuracy      0.9977   0.9909   0.9926   0.9910   0.9973
```

Applying these 2 algorithms to the provided Test Data Set result in:

```
TestSet<-read.csv2("pml-testing.csv", header=TRUE, sep = ",", dec=".", na.strings="NA")
data.frame(rfMod16f=predict(rfMod16f,TestSet),rfMod52f=predict(rfMod52f,TestSet))
```

```
##    rfMod16f rfMod52f
## 1         B        B
## 2         A        A
## 3         B        B
## 4         A        A
## 5         A        A
## 6         E        E
## 7         D        D
## 8         B        B
## 9         A        A
## 10        A        A
## 11        B        B
## 12          C         C
## 13          B         B
## 14          A         A
## 15          E         E
## 16          E         E
## 17          A         A
## 18          B         B
## 19          B         B
## 20          B         B
```

Both sets of predicted outcomes are identical.

Appendix: Reference Code

```
require(doParallel)
require(nnet)
require(caret)
require(dplyr)
require(ggplot2)
require(randomForest)
require(e1071)
cl <- makeCluster(detectCores())
registerDoParallel(cl)
set.seed(13234)
pml<-read.csv2("pml-training.csv", header=TRUE, sep = ",", dec=".", na.strings="NA")
partitionFactor<-transmute(pml,paste(toString(user_name[1]),toString(classe[1]),sep=""))
inTrain = createDataPartition(partitionFactor[,1],p=0.6,list=FALSE)
# Apologies for the ugly use of copy..paste programming
# Some refactoring would be appropriate (if I have the time to do it...)
pml16f = pml[grepl("^classe$|^roll_|^yaw_|^pitch_|^total_accel_",names(pml))]
training16f = pml16f[ inTrain,]
testing16f = pml16f[-inTrain,]
pml52f = pml[grepl("^classe$|^roll_|^yaw_|^pitch_|^total_accel_|_x$|_y$|_z$",names(pml))]
training52f = pml52f[ inTrain,]
testing52f = pml52f[-inTrain,]
pml88f = pml52f %>%
  mutate(gyros_belt_xy=gyros_belt_x*gyros_belt_y) %>%
    mutate(gyros_belt_xz=gyros_belt_x*gyros_belt_z) %>%
    mutate(gyros_belt_yz=gyros_belt_y*gyros_belt_z) %>%
    mutate(accel_belt_xy=accel_belt_x*accel_belt_y) %>%
    mutate(accel_belt_xz=accel_belt_x*accel_belt_z) %>%
    mutate(accel_belt_yz=accel_belt_y*accel_belt_z) %>%
    mutate(magnet_belt_xy=magnet_belt_x*magnet_belt_y) %>%
    mutate(magnet_belt_xz=magnet_belt_x*magnet_belt_z) %>%
    mutate(magnet_belt_yz=magnet_belt_y*magnet_belt_z) %>%
    mutate(gyros_arm_xy=gyros_arm_x*gyros_arm_y) %>%
    mutate(gyros_arm_xz=gyros_arm_x*gyros_arm_z) %>%
    mutate(gyros_arm_yz=gyros_arm_y*gyros_arm_z) %>%
    mutate(accel_arm_xy=accel_arm_x*accel_arm_y) %>%
    mutate(accel_arm_xz=accel_arm_x*accel_arm_z) %>%
    mutate(accel_arm_yz=accel_arm_y*accel_arm_z) %>%
    mutate(magnet_arm_xy=magnet_arm_x*magnet_arm_y) %>%
    mutate(magnet_arm_xz=magnet_arm_x*magnet_arm_z) %>%
    mutate(magnet_arm_yz=magnet_arm_y*magnet_arm_z) %>%
    mutate(gyros_dumbbell_xy=gyros_dumbbell_x*gyros_dumbbell_y) %>%
    mutate(gyros_dumbbell_xz=gyros_dumbbell_x*gyros_dumbbell_z) %>%
    mutate(gyros_dumbbell_yz=gyros_dumbbell_y*gyros_dumbbell_z) %>%
    mutate(accel_dumbbell_xy=accel_dumbbell_x*accel_dumbbell_y) %>%
    mutate(accel_dumbbell_xz=accel_dumbbell_x*accel_dumbbell_z) %>%
    mutate(accel_dumbbell_yz=accel_dumbbell_y*accel_dumbbell_z) %>%
    mutate(magnet_dumbbell_xy=magnet_dumbbell_x*magnet_dumbbell_y) %>%
    mutate(magnet_dumbbell_xz=magnet_dumbbell_x*magnet_dumbbell_z) %>%
    mutate(magnet_dumbbell_yz=magnet_dumbbell_y*magnet_dumbbell_z) %>%
    mutate(gyros_forearm_xy=gyros_forearm_x*gyros_forearm_y) %>%
    mutate(gyros_forearm_xz=gyros_forearm_x*gyros_forearm_z) %>%
    mutate(gyros_forearm_yz=gyros_forearm_y*gyros_forearm_z) %>%
    mutate(accel_forearm_xy=accel_forearm_x*accel_forearm_y) %>%
    mutate(accel_forearm_xz=accel_forearm_x*accel_forearm_z) %>%
    mutate(accel_forearm_yz=accel_forearm_y*accel_forearm_z) %>%
    mutate(magnet_forearm_xy=magnet_forearm_x*magnet_forearm_y) %>%
    mutate(magnet_forearm_xz=magnet_forearm_x*magnet_forearm_z) %>%
    mutate(magnet_forearm_yz=magnet_forearm_y*magnet_forearm_z)
training88f = pml88f[ inTrain,]
testing88f = pml88f[-inTrain,]
rfMod16f = train(classe~.,data=training16f,method="rf",
                 trControl=trainControl(method="cv"))
rfPred16f = predict(rfMod16f,testing16f)
rfMod52f = train(classe~.,data=training52f,method="rf",
                 trControl=trainControl(method="cv"))
rfPred52f = predict(rfMod52f,testing52f)
rfMod88f = train(classe~.,data=training88f,method="rf",
                 trControl=trainControl(method="cv"))
rfPred88f = predict(rfMod88f,testing88f)
nnMod16f = multinom(classe~.,data=training16f)
nnPred16f<-predict(nnMod16f,testing16f)
nnMod52f = multinom(classe~.,data=training52f,maxit=250)
nnPred52f<-predict(nnMod52f,testing52f)
nnMod88f = multinom(classe~.,data=training88f,maxit=500)
nnPred88f<-predict(nnMod88f,testing88f)
logModA16f<-glm(classe~.,mutate(training16f,classe=ifelse(classe=="A",TRUE,FALSE)),
                family="binomial",maxit=100)
predictA16f<-predict(logModA16f,testing16f,type="response")
logModB16f<-glm(classe~.,mutate(training16f,classe=ifelse(classe=="B",TRUE,FALSE)),
                family="binomial",maxit=100)
predictB16f<-predict(logModB16f,testing16f,type="response")
logModC16f<-glm(classe~.,mutate(training16f,classe=ifelse(classe=="C",TRUE,FALSE)),
                family="binomial",maxit=100)
predictC16f<-predict(logModC16f,testing16f,type="response")
logModD16f<-glm(classe~.,mutate(training16f,classe=ifelse(classe=="D",TRUE,FALSE)),
                family="binomial",maxit=100)
predictD16f<-predict(logModD16f,testing16f,type="response")
logModE16f<-glm(classe~.,mutate(training16f,classe=ifelse(classe=="E",TRUE,FALSE)),
                family="binomial",maxit=100)
predictE16f<-predict(logModE16f,testing16f,type="response")
logPredRaw16f<-data.frame(predictA16f,predictB16f,predictC16f,predictD16f,predictE16f)
logPred16f<-factor(apply(logPredRaw16f,1,which.max),labels=c("A","B","C","D","E"))
logModA52f<-glm(classe~.,mutate(training52f,classe=ifelse(classe=="A",TRUE,FALSE)),
                family="binomial",maxit=100)
predictA52f<-predict(logModA52f,testing52f,type="response")
logModB52f<-glm(classe~.,mutate(training52f,classe=ifelse(classe=="B",TRUE,FALSE)),
                family="binomial",maxit=100)
```

```r
predictB52f<-predict(logModB52f,testing52f,type="response")
logModC52f<-glm(classe~.,mutate(training52f,classe=ifelse(classe=="C",TRUE,FALSE)),
                family="binomial",maxit=100)
predictC52f<-predict(logModC52f,testing52f,type="response")
logModD52f<-glm(classe~.,mutate(training52f,classe=ifelse(classe=="D",TRUE,FALSE)),
                family="binomial",maxit=100)
predictD52f<-predict(logModD52f,testing52f,type="response")
logModE52f<-glm(classe~.,mutate(training52f,classe=ifelse(classe=="E",TRUE,FALSE)),
                family="binomial",maxit=100)
predictE52f<-predict(logModE52f,testing52f,type="response")
logPredRaw52f<-data.frame(predictA52f,predictB52f,predictC52f,predictD52f,predictE52f)
logPred52f<-factor(apply(logPredRaw52f,1,which.max),labels=c("A","B","C","D","E"))
logModA88f<-glm(classe~.,mutate(training88f,classe=ifelse(classe=="A",TRUE,FALSE)),
                family="binomial",maxit=100)
predictA88f<-predict(logModA88f,testing88f,type="response")
logModB88f<-glm(classe~.,mutate(training88f,classe=ifelse(classe=="B",TRUE,FALSE)),
                family="binomial",maxit=100)
predictB88f<-predict(logModB88f,testing88f,type="response")
logModC88f<-glm(classe~.,mutate(training88f,classe=ifelse(classe=="C",TRUE,FALSE)),
                family="binomial",maxit=100)
predictC88f<-predict(logModC88f,testing88f,type="response")
logModD88f<-glm(classe~.,mutate(training88f,classe=ifelse(classe=="D",TRUE,FALSE)),
                family="binomial",maxit=100)
predictD88f<-predict(logModD88f,testing88f,type="response")
logModE88f<-glm(classe~.,mutate(training88f,classe=ifelse(classe=="E",TRUE,FALSE)),
                family="binomial",maxit=100)
predictE88f<-predict(logModE88f,testing88f,type="response")
logPredRaw88f<-data.frame(predictA88f,predictB88f,predictC88f,predictD88f,predictE88f)
logPred88f<-factor(apply(logPredRaw88f,1,which.max),labels=c("A","B","C","D","E"))
rfConfMat16f<-confusionMatrix(rfPred16f,testing16f$classe)
rfAccuracy16f<-rfConfMat16f[[3]][1]
rfConfMat52f<-confusionMatrix(rfPred52f,testing52f$classe)
rfAccuracy52f<-rfConfMat52f[[3]][1]
rfConfMat88f<-confusionMatrix(rfPred88f,testing88f$classe)
rfAccuracy88f<-rfConfMat88f[[3]][1]
nnConfMat16f<-confusionMatrix(nnPred16f,testing16f$classe)
nnAccuracy16f<-nnConfMat16f[[3]][1]
nnConfMat52f<-confusionMatrix(nnPred52f,testing52f$classe)
nnAccuracy52f<-nnConfMat52f[[3]][1]
nnConfMat88f<-confusionMatrix(nnPred88f,testing88f$classe)
nnAccuracy88f<-nnConfMat88f[[3]][1]
logConfMat16f<-confusionMatrix(logPred16f,testing16f$classe)
logAccuracy16f<-logConfMat16f[[3]][1]
logConfMat52f<-confusionMatrix(logPred52f,testing52f$classe)
logAccuracy52f<-logConfMat52f[[3]][1]
logConfMat88f<-confusionMatrix(logPred88f,testing88f$classe)
logAccuracy88f<-logConfMat88f[[3]][1]
accuracies<-data.frame(RandomForest=c(rfAccuracy16f,rfAccuracy52f,rfAccuracy88f),
                       Multinom=c(nnAccuracy16f,nnAccuracy52f,nnAccuracy88f),
                       LogisticRegression=c(logAccuracy16f,logAccuracy52f,logAccuracy88f),
                       row.names=c("16Predictors","52Predictors","88Predictors"))
```