Fitness Quality Prediction

Lance Dooley

2/07/2021

Executive Summary: Machine Learning Algorithms for Fitness Quality Prediciton

This presentation will show how an appropriate Machine Learning Algorithm is trained on physical fitness data to predict the quality of the physical exercises. Three Machine Learning Algorithms were used to predict the outcome of classe for our data: Recursive Partitioning and Regression Trees, Random Forest, and Support Vector Machine.
Ultimately, I chose model created by **Random Forest** which produced an accurracy of **99.7%** and an out of sample error rate of: 1 **-** 0.997 = **.003** which is 0.3%

Background

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit, it is now possible to collect a large amount of data about personal activity relatively inexpensively.
This report uses data from accelerometers on the belt, forearm, arm, and dumbbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways and the classe of quality is the output we want to predict. Below, I walk through my process of acquiring the data, cleaning the data, and training models, and then show proof of my final choice, followed by plots of each type of fitted model.

Load the Required Libraries

```
# Load the required libraries
require(tidyverse); require(caret); require(rpart); require(randomForest); require(e1071); req
uire(rattle)
```

Read the Raw Data and Clean It

```
# Get the raw data
pml_training <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.cs
v")
pml_testing  <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
)

# Convert classe into factor
pml_training$classe <- as.factor(pml_training$classe)
levels(pml_training$classe)
```

```
## [1] "A" "B" "C" "D" "E"
```

```
# Exploration of columns in pml_training, shows many NA/empty or DIV/0! NA values.
# Remove all columns with  NA or empty or DIV/0! values.
not_any_na_empty_div0 <- function(x) { !any( { is.na(x) | x=="" | x== "#DIV/0!"} ) }
pml_training = pml_training[, apply( pml_training, 2, not_any_na_empty_div0 )]
pml_testing  = pml_testing[,  apply( pml_testing,  2, not_any_na_empty_div0 )]

# Remove these 5 columns in each data frame because they are not suitable predictors
not_needed <- c("X",  "user_name", "raw_timestamp_part_1", "raw_timestamp_part_2",
                "cvtd_timestamp", "new_window", "num_window")
pml_training <- pml_training %>% dplyr::select(-not_needed )
pml_testing  <- pml_testing  %>% dplyr::select(-not_needed )
```

Split the Training Set into Pure Training and Validation Training

```
set.seed(33833)
inTrain <-   caret::createDataPartition(y=pml_training$classe, p=0.75, list = FALSE)
pml_training_train    <- pml_training[inTrain,]
pml_training_validate <- pml_training[-inTrain,]
# Double check that the length of train and validate datasets = 19622
nrow(pml_training_train); nrow(pml_training_validate)
```

```
## [1] 14718
```

```
## [1] 4904
```

Train Fitted Models with 3 Different Machine Learning Algorithms

```
# Three Machine Algorithms were used to train on the data
modFit_rpart <- rpart::rpart(classe ~ ., method="class",  data=pml_training_train)
modFit_rf    <- randomForest::randomForest(classe ~ ., method="class",  data=pml_training_trai
n)
modFit_svm   <- e1071::svm(  classe ~ ., kernel="radial", data=pml_training_train )
```

Make Predictions on Validation Training Data

Use each of the 3 fitted models to make predictions on the Validation Training set, which we have carefully reserved for this purpose. This will be our last chance to do any kind of verification before making predictions on the test data.

```
# Now we use each of the 3 models to carry out predictions on the pml_training_validate data
pred_rpart <- stats::predict(modFit_rpart, newdata=pml_training_validate, type = "class" )
pred_rf    <- stats::predict(modFit_rf,    newdata=pml_training_validate, type = "class" )
pred_svm   <- stats::predict(modFit_svm,   newdata=pml_training_validate, type = "class" )
```

Cross Validation with Confusion Matrix

```
# Use Cross Validation to compare each prediction with the known values in validation training
set
confusion_matrix_rpart <- caret::confusionMatrix(pred_rpart, as.factor(pml_training_validate$c
lasse))
confusion_matrix_rf   <- caret::confusionMatrix(pred_rf,     as.factor(pml_training_validate$c
lasse))
confusion_matrix_svm  <- caret::confusionMatrix(pred_svm,    as.factor(pml_training_validate$c
lasse))

# Show just the tables of each confusion matrix below, and look for the one with the most agre
ement between Prediction and Reference (along the diagonal).
confusion_matrix_rpart$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1204  128   16   41   10
##          B   55  619   79   70   78
##          C   34   85  682  126   99
##          D   54   69   59  506   48
##          E   48   48   19   61  666
```

```
confusion_matrix_rf$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    4    0    0    0
##          B    0  943    2    0    0
##          C    0    2  852    4    0
##          D    0    0    1  800    1
##          E    0    0    0    0  900
```

```
confusion_matrix_svm$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1391   75    3    1    0
##          B    1  842   27    0    5
##          C    3   29  819   68   19
##          D    0    1    6  734   19
##          E    0    2    0    1  858
```

The Best Fitted Model was trained with Random Forest!
From the output of each confusion matrix, the clear winner is the fitted model that was created by the Random Forest, a confusion_matrix_rf$table above. However, to analyze and verify this one step deeper, let’s look at the overall statistics for each model so we know the true accuracy and confidence internal of the accuracy.

```
confusion_matrix_rpart$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull 
##   7.497961e-01   6.835538e-01   7.374251e-01   7.618690e-01   2.844617e-01 
## AccuracyPValue  McnemarPValue 
##   0.000000e+00   4.300939e-27
```

```
confusion_matrix_rf$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull 
##      0.9971452      0.9963887      0.9952147      0.9984384      0.2844617 
## AccuracyPValue  McnemarPValue 
##      0.0000000            NaN
```

```
confusion_matrix_svm$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull 
##      0.9469821      0.9328198      0.9403378      0.9530876      0.2844617 
## AccuracyPValue  McnemarPValue 
##      0.0000000            NaN
```

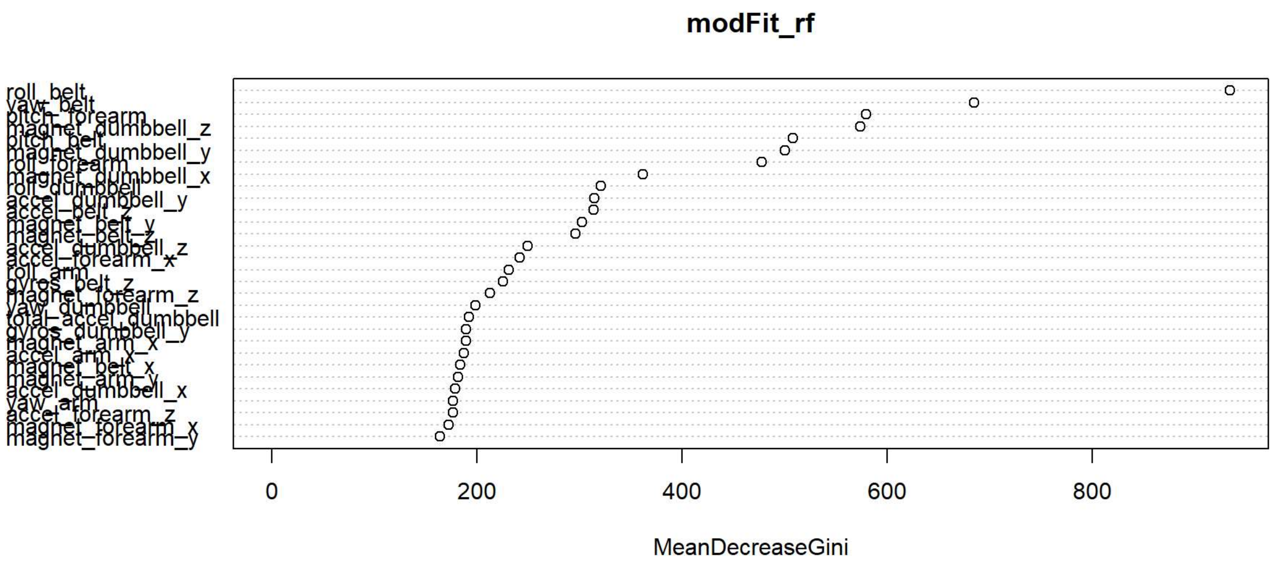
Statistical Accuracy of Random Forest on the Validation Data
Examining the confusion_matrix_rf$overall, overall statistics, we can see that the Random Forest machine learning algorithm is **99.7%** accurate, and we can say with **95% confidence that our accuracy is between 99.5% and 99.8%**
Also, we should expect **Out of Sample Error Rate = .003, or 0.03%**. The Support Vector Machine model wasn't far behind, and the rpart model was too low to consider.

Prediction on Test Data
Now we can use the fitted model for Random Forest, to predict the classe for each of the 20 samples in the test data set.

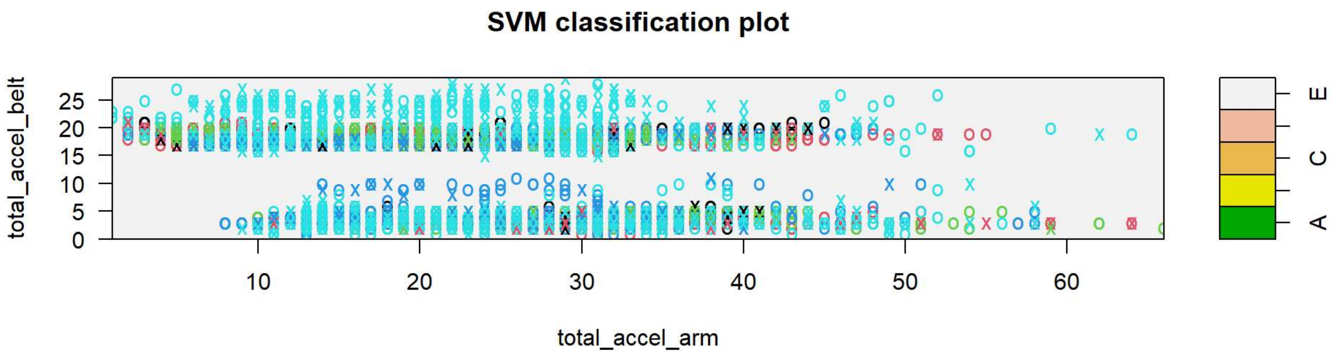
```
# Make final predction on testing data set
stats::predict( modFit_rf, newdata = pml_testing, type="class")
```

```
##  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
```

Plotting the Winning Model: Random Forest!
This section gives us a visual of our winning model, which used Random Forest. It is easy to see the most important predictors on the left. **The most important predictor is roll_belt**.

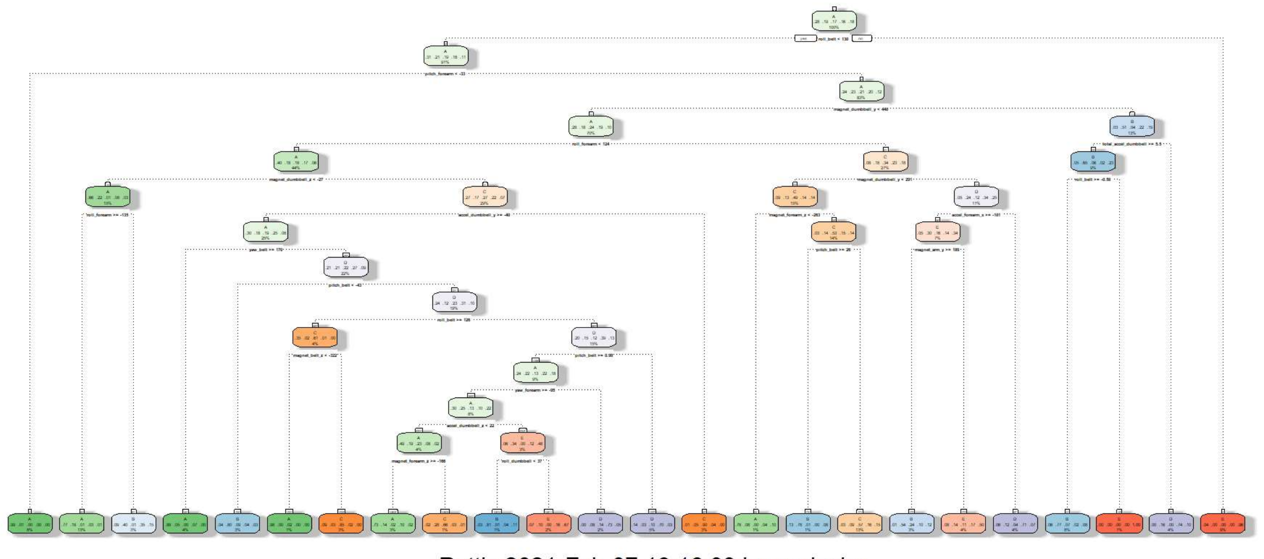


Plotting the complicated of Support Vector Machine SVM Classification Plot
This section is to demonstrate the complexities of the Support Vector Machine SVM Classification Plot. Even though we did not choose the svm fitted model, it is worth seeing. Because of the many factors used in our model, we would need many different plots to be of any use, and that is another reason not to use SVM for this project. This is only one plot, with two predictors on x and y axis, respectively.



Plotting the Worst Model: Recursive Partitioning and Regression Trees
This section gives us a visual of the worst model, Recursive Partitioning and Regression Trees, which had an accuracy of only approx 75%. If you enlarged this model, it would be possible to follow the tree downard, until

arriving at the nodes on the bottom, which represent the classe(s)



Rattle 2021-Feb-07 19:16:09 lance.dooley