JeffTheDon-Jeffery B. Donaven

April 5, 2018

Prediction of Errors in the Execution of Assorted Weight-Training Exercises

Synopsis

Weight-Training is common type of strength training utilizing the force of gravity in the form of weighted bars, dumbells or weight stacks so as to oppose the force generated by muscle through concentric or eccentric contraction. The study that this project is based on uses *accelerometers* that measure and record data so as to quantify how much of a particular activity that the test subjects had done. The **main objective** of this *project* is to determine whether or not it is possible to *classify errors* during the execution of the different exercises using the data gathered by the *accelerometers* and to predict the manner in which the subjects did each exercise. I am using regressionary techniques as the tool in which will create predictive models on the HAR-Dataset. I have classified errors in and correct execution of, "**lifting barbells**" with *sensitivity*, *specificity*, and **HIGH** *accuracy*.

Introduction

This study was performed on six male participants with ages between 20 and 28 years with little or minimal experience in regars to Weight-Training. Each were asked to peform one set of Unilateral Dumbell Biceps Curl consisting of 10 repetitions in the set using a relatively light, 1.25kg dumbbell in different ways:

- **Class A** - *Exactly according to the specification of the exercise*
- **Class B** - *Throwing the elbows to the front with each repetition*
- **Class C** - *Lifting the Dumbbell only halfway up with each repetition*
- **Class D** - *Lowering the Dumbbell only halway down with each repetition*
- **Class E** - *Throwing the hips to the front with each repetition*

Mounted sensors on each of the participants were located in their respective **gloves**, on their **armbands**, in their **lumbar support belts** and in each of the **dumbbells** used. All of which collected data on the "*Euler*" angles (**pitch, roll and yaw**). Additionally, there were readings and measurements recorded via the **raw accelerator, gyroscope, and magnetometer**. All of this information can be reviewed at the aforementioned website.

Downloading of Data

As referenced earlier, the data for this course project comes from the Human Activity Recognition DataSet which is hosted by Groupware@LES:

```
train_set_site <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
test_set_site <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
download.file(train_set_site, destfile = "training_set.csv")
download.file(test_set_site, destfile = "testing_set.csv")
date_of_download <- date()
date_of_download

## [1] "Thu Apr 19 10:03:14 2018"
```

Reading of and Preprocessing the Dataset:

```
train_set <- read.csv("training_set.csv", header = TRUE, na.strings = c("NA", "#DIV/0!",
""), stringsAsFactors = FALSE)
test_set <- read.csv("testing_set.csv", header = TRUE, na.strings = c("NA", "#DIV/0!",
""), stringsAsFactors = FALSE)
```

Upon reviewing this data, the features of which *can* be classified into 3 specific Variables: **Measurement**, **Summary** and **"HouseKeeping"**. When looking at the "*summary*" variables, I can see they begin with, "**amplitude, avg, kurtosis, min, max, skewness, and stddev**". These apply summary statistics on the "*measurement*" variables which begin with, "**accel, gyros, magnet, roll, pitch and yaw**". The "*summary*" variables would obviously be the choice variable for my model because they would have drastically cut down on the number of observations and processing time and they also contain the core *measurement* variables. Unfortunately, the *"test_set"* contains only missing values, so making predictions using the *summary* variable would be NULL. Oh well.

I will be removing the **"HouseKeeping"** variables that contain the row numbers **"X"**, the time_stamps(**""raw_timestamp_part_1","raw_timestamp_part_2"", "cvtd_timestamp"**), and the measurement intervals, "**new_window and num_window**".

```
sum_variables_indexed <- grepl("^amplitude|^avg|^kurtosis|^min|^max|^skewness|^stddev"
                               , names(train_set))
sum_variables <- names(train_set)[!sum_variables_indexed]
train_df <- train_set[, sum_variables]
na_be_gone_index <- sapply(train_df, function(x)sum(is.na(x)))
fin_train_df <- train_df[, -c(1:7)]
fin_train_df <- fin_train_df[, colSums(is.na(fin_train_df))==0]
```

Furthermore, the **train_set** contains `19622` *rows* and `160` *variables*. Conversely, the **test_set** contains `20` *rows* and `160` variables. I also need to set the variables to their respective correct class so as to avoid making errors during the modeling phase.

Setting of Variables

```
fin_train_df$accel_arm_x <- as.numeric(fin_train_df$accel_arm_x)
fin_train_df$accel_arm_y <- as.numeric(fin_train_df$accel_arm_y)
fin_train_df$accel_arm_z <- as.numeric(fin_train_df$accel_arm_z)
fin_train_df$total_accel_arm <- as.numeric(fin_train_df$total_accel_arm)
fin_train_df$accel_belt_x <- as.numeric(fin_train_df$accel_belt_x)
fin_train_df$accel_belt_y <- as.numeric(fin_train_df$accel_belt_y)
fin_train_df$accel_belt_z <- as.numeric(fin_train_df$accel_belt_z)
fin_train_df$total_accel_belt <- as.numeric(fin_train_df$total_accel_belt)
fin_train_df$accel_dumbbell_x <- as.numeric(fin_train_df$accel_dumbbell_x)
fin_train_df$accel_dumbbell_y <- as.numeric(fin_train_df$accel_dumbbell_y)
fin_train_df$accel_dumbbell_z <- as.numeric(fin_train_df$accel_dumbbell_z)
fin_train_df$total_accel_dumbbell <- as.numeric(fin_train_df$total_accel_dumbbell)
fin_train_df$accel_forearm_x <- as.numeric(fin_train_df$accel_forearm_x)
fin_train_df$accel_forearm_y <- as.numeric(fin_train_df$accel_forearm_y)
fin_train_df$accel_forearm_z <- as.numeric(fin_train_df$accel_forearm_z)
fin_train_df$total_accel_forearm <- as.numeric(fin_train_df$total_accel_forearm)
fin_train_df$magnet_arm_x <- as.numeric(fin_train_df$magnet_arm_x)
fin_train_df$magnet_arm_y <- as.numeric(fin_train_df$magnet_arm_y)
fin_train_df$magnet_arm_z <- as.numeric(fin_train_df$magnet_arm_z)
fin_train_df$magnet_belt_x <- as.numeric(fin_train_df$magnet_belt_x)
fin_train_df$magnet_belt_y <- as.numeric(fin_train_df$magnet_belt_y)
fin_train_df$magnet_belt_z <- as.numeric(fin_train_df$magnet_belt_z)
fin_train_df$magnet_dumbbell_x <- as.numeric(fin_train_df$magnet_dumbbell_x)
```

```
fin_train_df$magnet_dumbbell_y <- as.numeric(fin_train_df$magnet_dumbbell_y)
fin_train_df$magnet_forearm_x <- as.numeric(fin_train_df$magnet_forearm_x)
fin_train_df$classe <- as.factor(fin_train_df$classe)
```

Checking Variables which contain zeroes

```
zed_index <- sapply(fin_train_df[,-53], sum)
zed_variables <- which(zed_index == 0)
zed_variables

## named integer(0)

fin_train_df <- fin_train_df[-c(845, 867, 899, 7058, 8468, 9029, 9264, 11989, 17508),]
```

Creation of Training, Test & Validation sets of Data

So as to truly test this data in my models-to-come, I am going to partition the dataset into 3 groups; one training set at 50%, one test set at 30%, and one validation set at 20%. The downloaded *test_set* from earlier will be the **final** validation of my models.

```
set.seed(2018)
part_1 <- createDataPartition(y = fin_train_df$classe, p = .50, list = FALSE)
train_1_set <- fin_train_df[part_1,]
validation <- fin_train_df[-part_1,]
part_2 <- createDataPartition(y = validation$classe, p = 0.70, list = FALSE)
test_1_set <- validation[part_2,]
validate_1_set <- validation[-part_2,]
```

Below you will see a table showing the final datasets and how they are comprised:

Breakdown of Training/Testing & Validation DataSets

Type_of_DataSet	Number.of.Variables	Number.of.Rows
Training DataSet	53	9808
Testing DataSet	53	6866
Validation DataSet	53	2939

Creation of Models

I will begin by generating a Random Forest Model on the Training Dataset. Per the Instructions for this Project, I will be using the variable, “classe” as my dependent variable. The “classe” variable contains the important classification on whether or not the movement through the exercise was performed correctly. Additionally, it also provides me with what error was committed as per the “Introduction” section earlier. I am also including a 5-fold cross validation to improve my model as well as repeating it three times.

```
train_model_control <- trainControl(method = "repeatedcv", number = 5, repeats = 3)
fit_all_train_model = train(classe ~ ., data = train_1_set, method = "rf", trControl =
train_model_control)
```

Assessment of Model 1 - Training DataSet

Upon examination of the results, I find it to be pretty accurate with a value of approximately 98.5%.

mtry	Accuracy	Kappa	AccuracySD	KappaSD
2	0.9854540	0.9815973	0.0024442	0.0030934

27	0.9855898	0.9817701	0.0034984	0.0044280
52	0.9747821	0.9681002	0.0039412	0.0049889

The table below shows which predictions of the *Training DataSet* were correct, and which were errors via the **error_in_sample**. The errors are represented by the numbers which are not on the diagonal from top left to bottom right. The sum of errors on the Training DataSet numbered **96** out of a total **9790** which has a misclassifaction rate of **0.98%.** The Accuracy of correct classifications in my model is **99.02%**.

	A	B	C	D	E	class.error
A	2783	4	0	0	2	0.0021513
B	20	1866	10	1	0	0.0163416
C	0	14	1687	10	0	0.0140269
D	0	4	35	1566	3	0.0261194
E	0	3	4	7	1789	0.0077648

I expect the error out of sample to be slightly less than the error in sample that was measured above. Now lets see how the predictions go against the Testing DataSet:

```
pred_test_set <- predict(fit_all_train_model, newdata = test_1_set)
err_out_of_sample <- table(pred_test_set, test_1_set$classe)
accuracy_of_model <- confusionMatrix(pred_test_set, test_1_set$classe)
```

	A	B	C	D	E
A	1944	15	0	0	0
B	5	1307	15	0	3
C	3	5	1174	20	4
D	0	1	8	1105	6
E	0	0	0	1	1250

As predicted, the **error out of sample** for the *Test DataSet* via the table produced above, shows a misclassification rate of **1.22%** with an Accuracy Rate for this DataSet of **98.78%** with **84** misclassifacations out of a total possible **6866**. This confirmed my hypothesis of it being less accurate than the **error in sample**.

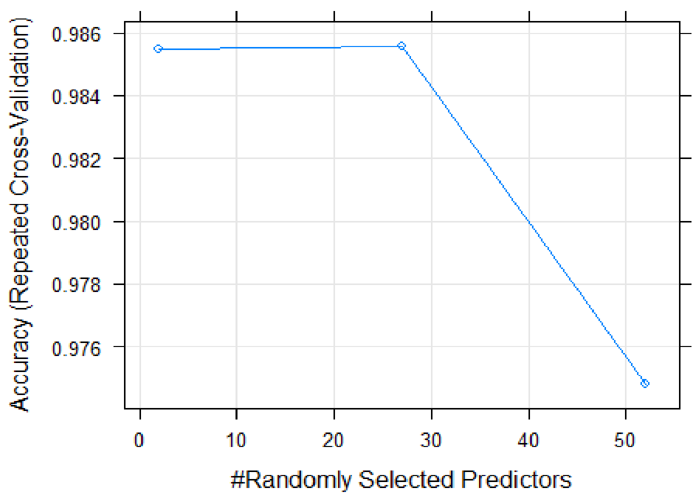
The **confusionMatrix** function summarizes in detail the **accuracy, sensitivity, specficty** as well as other *parameters* of my model's prediction by *class*. Let's take a look at this:

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: A	0.9959016	0.9969475	0.9923430	0.9983697	0.9923430	0.9959016	0.9941192	0.2842994	0.2831343	0.2853190	0.9964246
Class: B	0.9841867	0.9958469	0.9827068	0.9962066	0.9827068	0.9841867	0.9834462	0.1934168	0.1903583	0.1937081	0.9900168
Cla	0.9807	0.9943	0.9734	0.9959	0.9734	0.9807	0.9771	0.1743	0.1709	0.1756	0.9875

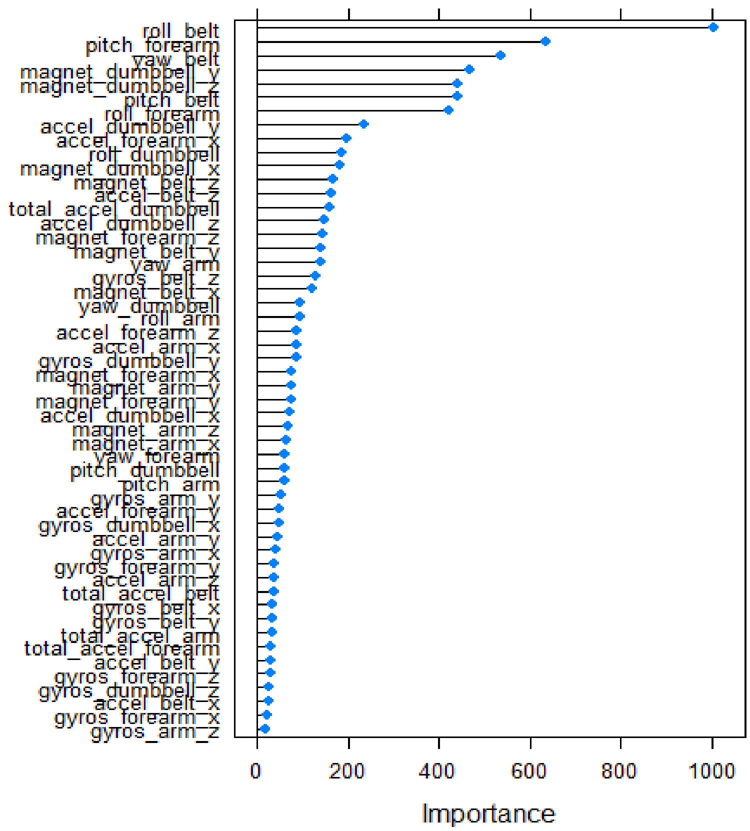
ss: C	853	553	660	364	660	853	119	373	875	481	703
Class: D	0.9813499	0.9973868	0.9866071	0.9963453	0.9866071	0.9813499	0.9839715	0.1639965	0.1609380	0.1631226	0.9893683
Class: E	0.9897070	0.9998215	0.9992006	0.9976848	0.9992006	0.9897070	0.9944312	0.1839499	0.1820565	0.1822022	0.9947643

I have created a plot to showcase the relationship that exists between the *number of randomly selected predictors* and the *accuracy*. Accuracy of the model is at its highest point, when, **mtry**(the tuning parameter for caret) has the number of variables available for splitting at each tree node is at 27. “mtry” is defined as, ‘The number of Variables randomly sampled as canditates for each split.’ Keep this in mind whilst viewing the plot *below*:

```
plot(fit_all_train_model)
```



Next I want to check which features are highly **correlated** so as to decide which features to keep for the next model run. I want to hone the model so as to improve *processing time*, *scalability* as well as *interpretability*. Therefore I am going to take a look at the Features and see just how important each one



is:

names(train_1_set)[higher_cor]
accel_belt_z
roll_belt
accel_belt_y
accel_dumbbell_z
accel_belt_x
pitch_belt
accel_dumbbell_x
accel_arm_x
magnet_arm_y
gyros_forearm_y
gyros_dumbbell_x
gyros_dumbbell_z
gyros_arm_x

So, can I gain anymore precision in my model with the above found correlated variables? I shall run another model to see if it is possible with the above listed 13 variables which is the top 20% of variables to find out:

```
final_model <- trainControl(method = "repeatedcv", number = 5, repeats = 3)
fit_train_13_model= train(classe ~ accel_belt_z + roll_belt + accel_belt_y +
accel_dumbbell_x + accel_belt_x + pitch_belt + accel_dumbbell_x + accel_arm_x +
magnet_arm_y + gyros_forearm_y + gyros_dumbbell_x + gyros_dumbbell_z + gyros_arm_x, data
= train_1_set, method = "rf", trControl = final_model)

pred_model_13 <- predict(fit_train_13_model, newdata = test_1_set)
accuracy_model_13 <- confusionMatrix(pred_model_13, test_1_set$classe)
```

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: A	0.9713115	0.9886040	0.9713115	0.9886040	0.9713115	0.9713115	0.9713115	0.2842994	0.2761433	0.2842994	0.9799577
Class: B	0.9593373	0.9909715	0.9622356	0.9902562	0.9622356	0.9593373	0.9607843	0.1934168	0.1855520	0.1928343	0.9751544
Class: C	0.9440267	0.9818310	0.9164639	0.9881058	0.9164639	0.9440267	0.9300412	0.1743373	0.1645791	0.1795805	0.9629289
Class: D	0.9333925	0.9885017	0.9409132	0.9869543	0.9409132	0.9333925	0.9371378	0.1639965	0.1530731	0.1626857	0.9609471
Class: E	0.9730800	0.9980368	0.9911290	0.9939566	0.9911290	0.9730800	0.9820216	0.1839499	0.1789980	0.1806001	0.9855584

Well, running the model with the 13 correlated variables showed a slight decrease in accuracy, specificity and sensitivity. Although the numbers went down, I want to run one more model, this time with only 6 variables to see if I can get a rise with the numbers. I think the numbers will continue to go down a bit though.

```
fit_train_6_model = train(classe ~ accel_belt_z + roll_belt + accel_belt_y +
accel_dumbbell_x + accel_belt_x + pitch_belt, data = train_1_set, method = "rf",
trControl = final_model)

pred_model_6 <- predict(fit_train_6_model, newdata = test_1_set)
accuracy_model_6 <- confusionMatrix(pred_model_6, test_1_set$classe)
```

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: A	0.8580943	0.9163614	0.8029722	0.9420502	0.8029722	0.8580943	0.8296186	0.2842994	0.2439557	0.3038159	0.8872278
Class: B	0.8305723	0.9712893	0.8740095	0.9598501	0.8740095	0.8305723	0.8517375	0.1934168	0.1606467	0.1838043	0.9009308
Class: C	0.7719298	0.9585465	0.7972390	0.9521640	0.7972390	0.7719298	0.7843803	0.1743373	0.1345762	0.1688028	0.8652382
Class: D	0.7904085	0.9573171	0.7841410	0.9588205	0.7841410	0.7904085	0.7872623	0.1639965	0.1296242	0.1653073	0.8738628
Class: E	0.9326999	0.9917901	0.9624183	0.9849344	0.9624183	0.9326999	0.9473261	0.1839499	0.1715701	0.1782697	0.9622450

As predicted, sensitivity, specificity and accuracy all suffered from a decrease in the numbers with 6 variables. However, this model has better interpreability, scalability and faster processing time even though there is an increase in bias which reduced my capacity to predict accurately.

Validation Set Model Fitting

Now I shall run the model on the validation test set to see how accurate it is. I will be running the model with all the variables and the one with only 6 to see the disparity between the two.

```
pred_val_test_all <- predict(fit_all_train_model, newdata = validate_1_set)
accuracy_val_all <-confusionMatrix(pred_val_test_all, validate_1_set$classe)
pred_model_6b <- predict(fit_train_6_model, newdata = validate_1_set)
accuracy_model_6b <- confusionMatrix(pred_model_6b, validate_1_set$classe)
```

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: A	0.9928230	0.9990490	0.9975962	0.9971523	0.9975962	0.9928230	0.9952038	0.2844505	0.2824090	0.2830895	0.9959360
Class: B	0.9929577	0.9945171	0.9774697	0.9983065	0.9774697	0.9929577	0.9851528	0.1932630	0.1919020	0.1963253	0.9937374
Class: C	0.9844055	0.9929926	0.9674330	0.9966901	0.9674330	0.9844055	0.9758454	0.1745492	0.1718272	0.1776114	0.9886990
Class:	0.9792531	0.9987790	0.9936842	0.9959416	0.9936842	0.9792531	0.9864159	0.1640014	0.1605988	0.1616196	0.9890161

D

Class: E	0.9870370	1.0000000	1.0000000	0.9970906	1.0000000	0.9870370	0.9934762	0.1837360	0.1813542	0.1813542	0.9935185

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: A	0.8648325	0.9196386	0.8105381	0.9447973	0.8105381	0.8648325	0.8368056	0.2844505	0.2460020	0.3035046	0.8922356
Class: B	0.8204225	0.9637284	0.8442029	0.9572685	0.8442029	0.8204225	0.8321429	0.1932630	0.1585573	0.1878190	0.8920755
Class: C	0.7524366	0.9571311	0.7877551	0.9481421	0.7877551	0.7524366	0.7696909	0.1745492	0.1313372	0.1667234	0.8547839
Class: D	0.8070539	0.9601140	0.7987680	0.9620718	0.7987680	0.8070539	0.8028896	0.1640014	0.1323579	0.1657026	0.8835840
Class: E	0.9333333	0.9941642	0.9729730	0.9851301	0.9729730	0.9333333	0.9527410	0.1837360	0.1714869	0.1762504	0.9637488

Conclusions

Classification of Errors using predictive modeling on the **HAR Weightlifting Dataset** showed high **sensitivity**, **specificity** and **accuracy** with the correct execution of lifting the dumbbells. It needds to be pointed out though, that the **errors** in the movement of the excercises were performed *purposefully*. Because of this fact, different results could be retained when the errors in the movements are committed *without* intent to commit the error in the first place.

Please see the appendix for different plots which will show the misclassification of errors by the the models on the Validation Test Set.

Predictions on the Test DataSet

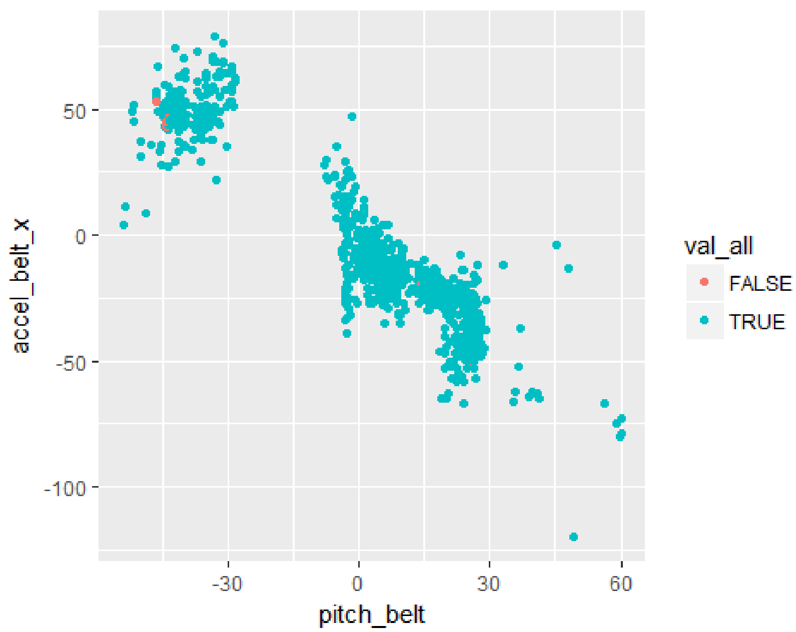
I will now use the models created herein to Predict on the Downloaded Test DataSet:

```
test_data <- test_set[ , which(names(test_set) %in% names(train_1_set))]
pred_test_all <- predict(fit_all_train_model, newdata = test_set)
print(pred_test_all)

##  [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

pred_test_13 <- predict(fit_train_13_model, newdata = test_set)
print(pred_test_13)
```

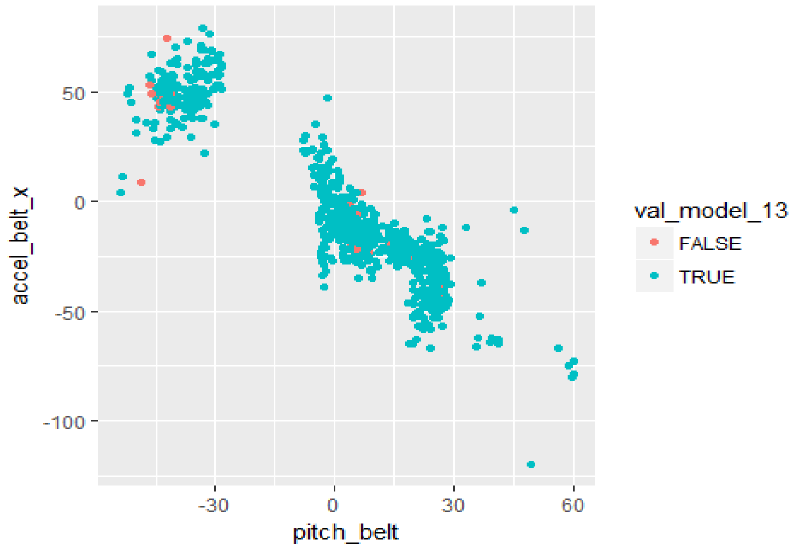
```
##  [1] B A B A A E D C A A A C B A E E A B B B
## Levels: A B C D E

pred_test_6 <- predict(fit_train_6_model, newdata = test_set)
print(pred_test_6)

##  [1] B C B D C E C D B A C C B D E E A B B B
## Levels: A B C D E
```

Appendix

```
val_all<- pred_val_test_all == validate_1_set$classe
qplot(pitch_belt, accel_belt_x, color = val_all, data = validate_1_set)
```




```
pred_val_13 <- predict(fit_train_13_model, newdata = validate_1_set)
accuracy_val_13 <- confusionMatrix(pred_val_13, validate_1_set$classe)
val_model_13 <- pred_val_13 == validate_1_set$classe
qplot(pitch_belt, accel_belt_x, color = val_model_13, data = validate_1_set)
```




```r
pred_val_6 <- predict(fit_train_6_model, newdata = validate_1_set)
accuracy_val_6 <- confusionMatrix(pred_val_6, validate_1_set$classe)
val_model_6 <- pred_val_6 == validate_1_set$classe
qplot(pitch_belt, accel_belt_x, color = val_model_6, data = validate_1_set)
```