Machine Learning Classifier

Kim Kirk

May 14, 2018

Supervised Machine Learning Classifier of Human Activity Recognition

Predicting quality of personal exercise activity as part of the quantified self movement

Executive Summary

The business question, "How well do members of the quantified self movement perform weight lifting exercises as part of their personal activity?" was answered. Weight Lifting Exercise Dataset was imported, cleaned, and modeled for how well subjects performed their weight lifting exercises. The outcome and predictor variables were identified. Both the outcome variable (CLASSE) and predictor variables were analyzed for missing values, outliers, correlation, etc. ; see "Data Processing" for additional details. Feature engineering was conducted to select and transform relevant variables. The classification model was created using Random Forest algorithm given the categorical nature of the outcome variable and several predictor variables, the highly non-linear relationship between outcome and predictor variables, the multiclass nature of the outcome variable, and no need for the assumption of normality of the predictor variables. The classifier had an out-of-sample accuracy rate of 99.82%.

Data Processing

The data set is imported, as are required packages. Exploratory data analysis is conducted on the data set.

There are multiple categorical predictor variables:

```r
path <- file.path(paste(getwd(), 'pml-training.csv', sep = "/"))
url <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
download.file(url, path)
trainingImport <- read.csv("pml-training.csv", header = TRUE, stringsAsFactors = FALSE)

##check environment for packages and install or not as required
##credit Matthew on StackOverflow https://stackoverflow.com/users/4125693/matthew
using<-function(...) {
    libs<-unlist(list(...))
    req<-unlist(lapply(libs,require,character.only=TRUE))
    need<-libs[req==FALSE]
    n<-length(need)
    if(n>0){
        libsmsg<-if(n>2) paste(paste(need[1:(n-1)],collapse=", "),",",sep="") else need[1]
        print(libsmsg)
        if(n>1){
            libsmsg<-paste(libsmsg," and ", need[n],sep="")
        }
        libsmsg<-paste("The following packages could not be found: ",libsmsg,"\n\r\n\rInstall missing pa
        if(winDialog(type = c("yesno"), libsmsg)=="YES"){
            install.packages(need)
            lapply(need,require,character.only=TRUE)
        }
    }
}

##install and load packages
using("dplyr")
```

```
## Loading required package: dplyr

## Warning: package 'dplyr' was built under R version 3.4.3

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
using("caret")
```

```
## Loading required package: caret

## Warning: package 'caret' was built under R version 3.4.4

## Loading required package: lattice

## Warning: package 'lattice' was built under R version 3.4.3

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 3.4.3
```

```r
using("mlbench")
```

```
## Loading required package: mlbench

## Warning: package 'mlbench' was built under R version 3.4.4
```

```r
set.seed(550)

glimpse(trainingImport[,sapply(trainingImport, typeof) == typeof("character")])
```

```
## Observations: 19,622
## Variables: 37
## $ user_name              <chr> "carlitos", "carlitos", "carlitos", "c...
## $ cvtd_timestamp         <chr> "05/12/2011 11:23", "05/12/2011 11:23"...
## $ new_window             <chr> "no", "no", "no", "no", "no", "no", "n...
## $ kurtosis_roll_belt     <chr> "", "", "", "", "", "", "", "", "", ""...
## $ kurtosis_picth_belt    <chr> "", "", "", "", "", "", "", "", "", ""...
## $ kurtosis_yaw_belt      <chr> "", "", "", "", "", "", "", "", "", ""...
## $ skewness_roll_belt     <chr> "", "", "", "", "", "", "", "", "", ""...
## $ skewness_roll_belt.1   <chr> "", "", "", "", "", "", "", "", "", ""...
## $ skewness_yaw_belt      <chr> "", "", "", "", "", "", "", "", "", ""...
## $ max_yaw_belt           <chr> "", "", "", "", "", "", "", "", "", ""...
## $ min_yaw_belt            <chr> "", "", "", "", "", "", "", "", "", ""...
## $ amplitude_yaw_belt      <chr> "", "", "", "", "", "", "", "", "", ""...
## $ kurtosis_roll_arm       <chr> "", "", "", "", "", "", "", "", "", ""...
## $ kurtosis_picth_arm      <chr> "", "", "", "", "", "", "", "", "", ""...
## $ kurtosis_yaw_arm        <chr> "", "", "", "", "", "", "", "", "", ""...
## $ skewness_roll_arm       <chr> "", "", "", "", "", "", "", "", "", ""...
## $ skewness_pitch_arm      <chr> "", "", "", "", "", "", "", "", "", ""...
## $ skewness_yaw_arm        <chr> "", "", "", "", "", "", "", "", "", ""...
## $ kurtosis_roll_dumbbell  <chr> "", "", "", "", "", "", "", "", "", ""...
## $ kurtosis_picth_dumbbell <chr> "", "", "", "", "", "", "", "", "", ""...
## $ kurtosis_yaw_dumbbell   <chr> "", "", "", "", "", "", "", "", "", ""...
## $ skewness_roll_dumbbell  <chr> "", "", "", "", "", "", "", "", "", ""...
## $ skewness_pitch_dumbbell <chr> "", "", "", "", "", "", "", "", "", ""...
## $ skewness_yaw_dumbbell   <chr> "", "", "", "", "", "", "", "", "", ""...
## $ max_yaw_dumbbell        <chr> "", "", "", "", "", "", "", "", "", ""...
## $ min_yaw_dumbbell        <chr> "", "", "", "", "", "", "", "", "", ""...
## $ amplitude_yaw_dumbbell  <chr> "", "", "", "", "", "", "", "", "", ""...
## $ kurtosis_roll_forearm   <chr> "", "", "", "", "", "", "", "", "", ""...
## $ kurtosis_picth_forearm  <chr> "", "", "", "", "", "", "", "", "", ""...
## $ kurtosis_yaw_forearm    <chr> "", "", "", "", "", "", "", "", "", ""...
## $ skewness_roll_forearm   <chr> "", "", "", "", "", "", "", "", "", ""...
## $ skewness_pitch_forearm  <chr> "", "", "", "", "", "", "", "", "", ""...
## $ skewness_yaw_forearm    <chr> "", "", "", "", "", "", "", "", "", ""...
## $ max_yaw_forearm         <chr> "", "", "", "", "", "", "", "", "", ""...
## $ min_yaw_forearm         <chr> "", "", "", "", "", "", "", "", "", ""...
## $ amplitude_yaw_forearm   <chr> "", "", "", "", "", "", "", "", "", ""...
## $ classe                  <chr> "A", "A", "A", "A", "A", "A", "A", "A"...
```

- The outcome variable is categorical, so analyzing for linearity between outcome and predictors is not appropriate.

The outcome variable is also multiclass in nature:

```r
table(trainingImport$classe)
```

```
##
##    A    B    C    D    E
## 5580 3797 3422 3216 3607
```

- Clearly, the model will need to be a classifier type.

The data set also has missing and irregular data:

```r
table(is.na(trainingImport))
```

```
##
##   FALSE    TRUE
## 1852048 1287472
```

```r
#index positions of irregular data
grep(pattern = "#DIV/0!", x = trainingImport)
```

```
##  [1]  12  13  14  15  16  17  20  23  26  69  70  71  72  73  74  87  88
## [18]  89  90  91  92  95  98 101 125 126 127 128 129 130 133 136 139
```

```r
#index positions of blank values
grep(pattern = "", x = trainingImport)
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  [18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
##  [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
##  [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
##  [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
##  [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100 101 102
## [103] 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
## [120] 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136
## [137] 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153
## [154] 154 155 156 157 158 159 160
```

- The missing and irregular data are confined to variables that hold descriptive statistics and intentional missing data. Given that these missing and irregular data are statistics that are inherent in the data set, they are removed from the data set to eliminate redundancy.

```r
indexPositions <- grep("^kurtosis|skewness|max|min|amplitude|var|avg|stddev", colnames(trainingImport),
trainingDataSet <- select(trainingImport, 1:160, -(indexPositions))
#check for NAs
sum(is.na(trainingDataSet))
```

```
## [1] 0
```

```r
#check for irregular values
grep(pattern = "#DIV/0!", x = trainingDataSet)
```

```
## integer(0)
```

The data set also has outliers for several variables; two examples of which are shown:

```r
boxplot(summary(trainingDataSet$magnet_forearm_z), main = "Magnet_Forearm_Z Variable", xlab = "magnet_fo
```

```r
boxplot(summary(trainingDataSet$accel_forearm_x), main = "Accel_Forearm_X Variable", xlab = "accel_fore
```

The data set has predictor variables that are correlated:

```r
correlationMatrix <- cor(trainingDataSet[,-c(1, 2, 5, 6, 60)])
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.5, names = TRUE)
print(highlyCorrelated)
```

```
##  [1] "accel_belt_z"         "roll_belt"            "accel_belt_y"
##  [4] "accel_arm_y"          "total_accel_belt"     "yaw_belt"
##  [7] "accel_dumbbell_z"     "accel_belt_x"         "magnet_belt_x"
## [10] "pitch_belt"           "yaw_dumbbell"         "magnet_dumbbell_x"
## [13] "accel_dumbbell_y"     "magnet_dumbbell_y"    "total_accel_dumbbell"
## [16] "magnet_dumbbell_z"    "accel_forearm_x"      "accel_arm_x"
## [19] "raw_timestamp_part_1" "accel_dumbbell_x"     "magnet_forearm_z"
## [22] "accel_arm_z"          "magnet_arm_y"         "magnet_belt_y"
## [25] "accel_forearm_y"      "magnet_arm_x"         "pitch_arm"
## [28] "gyros_dumbbell_y"     "gyros_forearm_z"      "gyros_forearm_y"
## [31] "gyros_dumbbell_z"     "gyros_arm_x"
```

```r
trainingDataSet <- select(trainingDataSet, -one_of(highlyCorrelated))
```

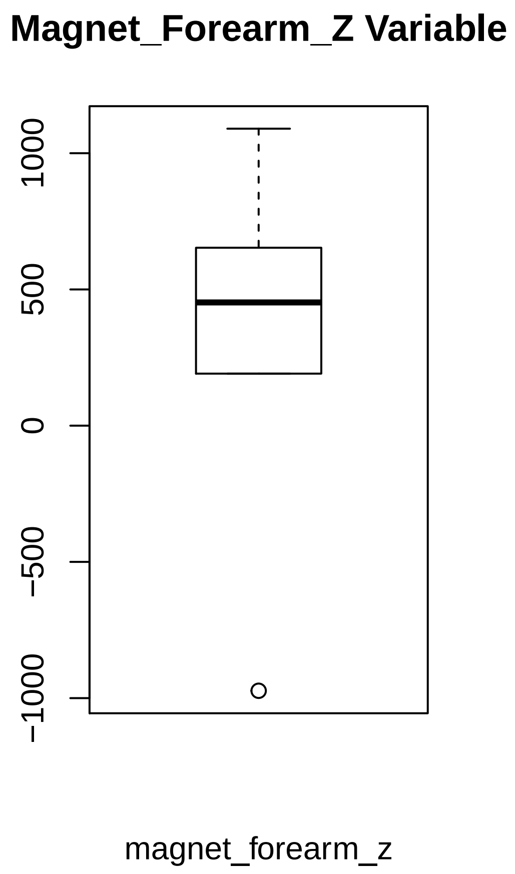


Figure 1: Outliers for two predictor variables

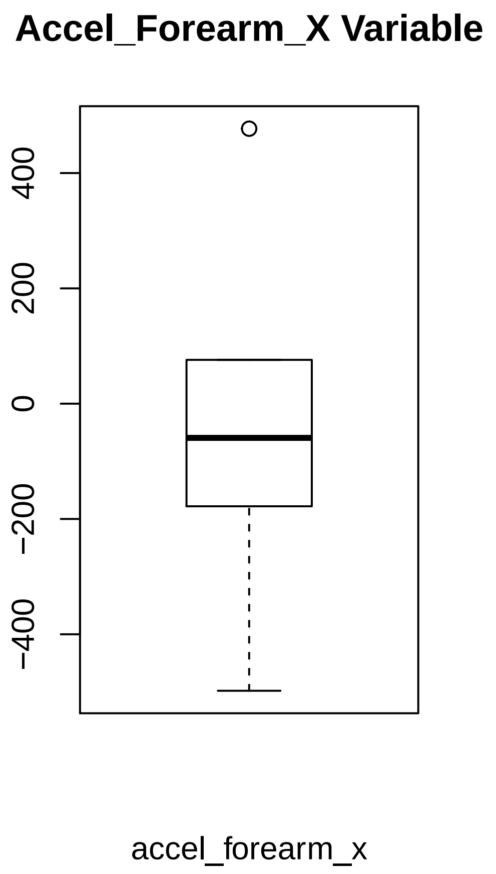


Figure 2: Outliers for two predictor variables

- Predictor variables the are correlated to each other are removed to reduce redundancy.

The CLASSE outcome variable has issues with class imbalance: See Appendix A

Feature engineering is performed. The variable X is removed as it represents a counting variable for the number of observations and is not relevant to prediction.

```
trainingDataSet <- trainingDataSet[,-(1)]
```

Additionally, feature transformation is performed on the CLASSE variable to turn it into a factor data type so that once the data is partitioned into a test set, no error message will occur about missing outcome values if the resampling doesn't pick up one of the values.

```
trainingDataSet$classe <- as.factor(trainingDataSet$classe)
```

Other continuous predictor variables have non-Gaussian distributions; two examples shown: See Appendix A

- Several of these variables cannot be transformed using a more straightforward transformation such as a BoxCox or Log due to the negative and/or zero values in the data.

Model Creation

Random Forest machine learning algorithm from the "caret" package can handle issues found in the data set during the data exploration phase such as - categorical nature of the outcome variable to create a classification model - mixed data types for predictor variables (quantitative and qualitative) without the requirement to create dummy variables or one-hot encoding - non-linear relationship between outcome and predictor variables - multiclass nature of the outcome variable - no assumption of normality of the predictor variables is required - increased interpretability for quantitative predictor variables that could not be transformed with a BoxCox or Log transformation - apply binning to imbalanced data

Parameters are tuned for resampling/validation purposes:

```
 ctrl = trainControl(method="cv", number = 3, selectionFunction = "oneSE")
```

- Cross validation with 3 folds is used with a selection function of oneSE because the tuning parameter associated with the best performance may over fit; the simplest model within one standard error of the empirically optimal model is the better choice @Breiman et al.(1984).

The training data is further partitioned into training and validation test sets:

```
 inTraining <- createDataPartition(trainingDataSet$classe, p = .75, list = FALSE)
    training <- trainingDataSet[ inTraining,]
    testing  <- trainingDataSet[-inTraining,]
```

Training data is preprocessed to transform the variables via scale and center. Binning is automatically performed by the algorithm for imbalanced data, and the data contains no missing values.

```
trained <- train(classe ~ .,
                 data = training, method = "rf",
                 trControl = ctrl, metric = "Accuracy",
                 preProcess = c("center", "scale"))
```

- Let's take a look at the model characteristics.

```
print.train(trained)
```

```
## Random Forest
##
## 14718 samples
##    26 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing: centered (48), scaled (48)
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 9811, 9812, 9813
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9424513  0.9271458
##   25    0.9979617  0.9974219
##   48    0.9957196  0.9945855
##
## Accuracy was used to select the optimal model using  the one SE rule.
## The final value used for the model was mtry = 25.
```

- The best accuracy is shown to be 99.79% using 25 variables.
- Let's plot the resampling profile of model to examine the relationship between the estimates of performance and the tuning parameters. See Appendix A.
- Let's see the most important variable in the training of the model. See Appendix A.
- The num_window (window number) seems to be the predictor with the most importance in training the model. Window number refers to the window of time (0.5s to 2.5s) in which calculations were made for Euler angles and raw accelerometer, gyroscope and magnetometer readings for the sensors used on the study subjects @Velloso, E. et al. [p. 3] . It seems that the data for a specific window of time would be of importance to classifying the subject's activity.

Now to make predictions using the trained model and the validation data set.

```
predicted <- predict(trained, newdata = testing, type = "raw")
```

Model Metrics

- Cross validation was used with 3 folds, 25 predictors, and an accuracy rate of 99.82%
- Let's create a confusion matrix to see prediction accuracy statistics and a breakdown across classes.

```
confusionMatrix <- confusionMatrix(predicted, testing[,27], mode = "everything")
confusionMatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    0    0    0    0
##          B    0  945    0    0    0
##          C    0    4  854    3    0
##          D    0    0    1  801    0
##          E    0    0    0    0  901
##
## Overall Statistics
##
##                Accuracy : 0.9984
##                  95% CI : (0.9968, 0.9993)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9979
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9958   0.9988   0.9963   1.0000
## Specificity            1.0000   1.0000   0.9983   0.9998   1.0000
## Pos Pred Value         1.0000   1.0000   0.9919   0.9988   1.0000
## Neg Pred Value         1.0000   0.9990   0.9998   0.9993   1.0000
## Precision              1.0000   1.0000   0.9919   0.9988   1.0000
## Recall                 1.0000   0.9958   0.9988   0.9963   1.0000
## F1                     1.0000   0.9979   0.9953   0.9975   1.0000
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2845   0.1927   0.1741   0.1633   0.1837
## Detection Prevalence   0.2845   0.1927   0.1756   0.1635   0.1837
## Balanced Accuracy      1.0000   0.9979   0.9986   0.9980   1.0000
```

- The accuracy rate, which is overall how often the classifier is correct, is 99.84% which is high.
- The out-of-sample error rate is 0.16% which is low. The p-value is significant.

There are a few misclassifications: - one instance of classifying a C class as a D class - three instances of classifying a D class as a C class - four instances of classifying a B class as a C class

- Sensitivity (when the class is actually the correct class, how often does the classifier predict correct class) for each class is no less than 99.58% which is high.
- Specificity (when the class is actually the incorrect class, how often does the classifier predict the incorrect class) for each class is no less than 99.83% which is high.
- Precision (when the classifier predicts a class, how often is the prediction correct) is no less than 99.19% for each class, which is high.

Conclusion

Overall, the accuracy of the classifier is very high. There are some misclassifications but they are minimal. Random Forest algorithm was a good choice to build the model as it easily handles various issues of outliers which can skew the data and create misclassified predictions, class imbalance which can bias the model as it will choose the heavily represented class during training, mixed data type predictors which would require dummy variable creation or one-hot encoding, no assumption of normality for the variables, and multiclass classification.

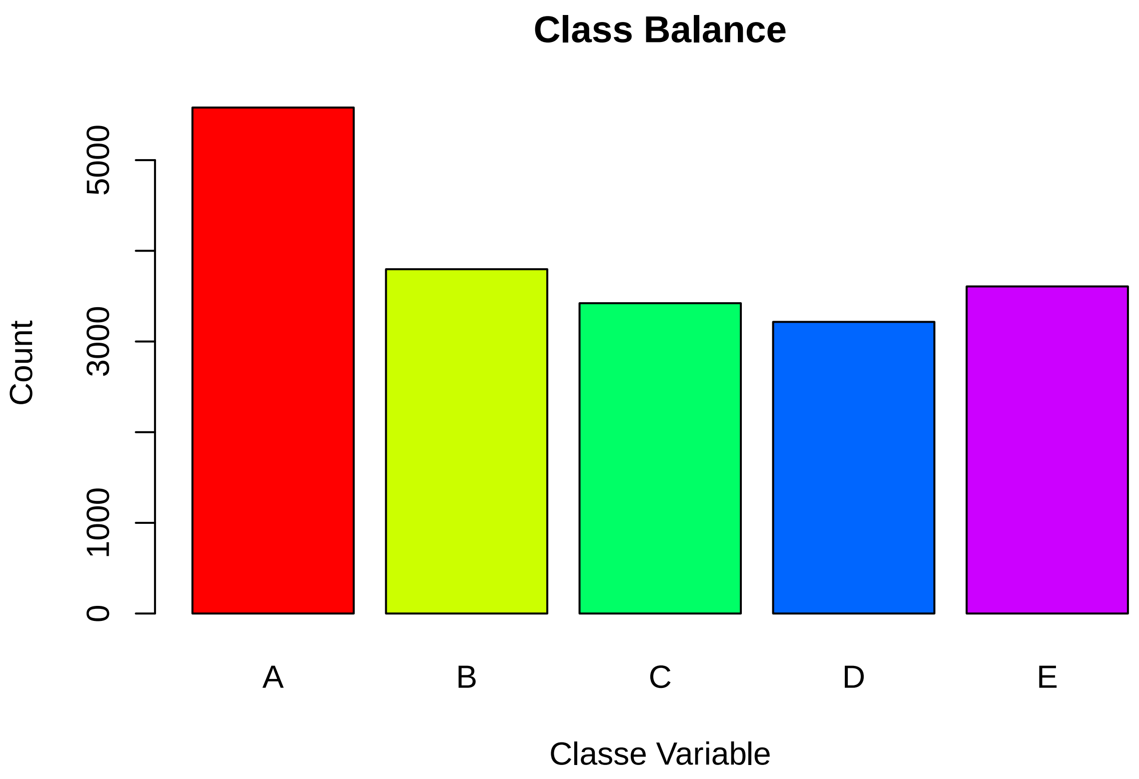
References

Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.I. (1984). Classification and regression trees. Belmont, Calif.: Wadsworth.

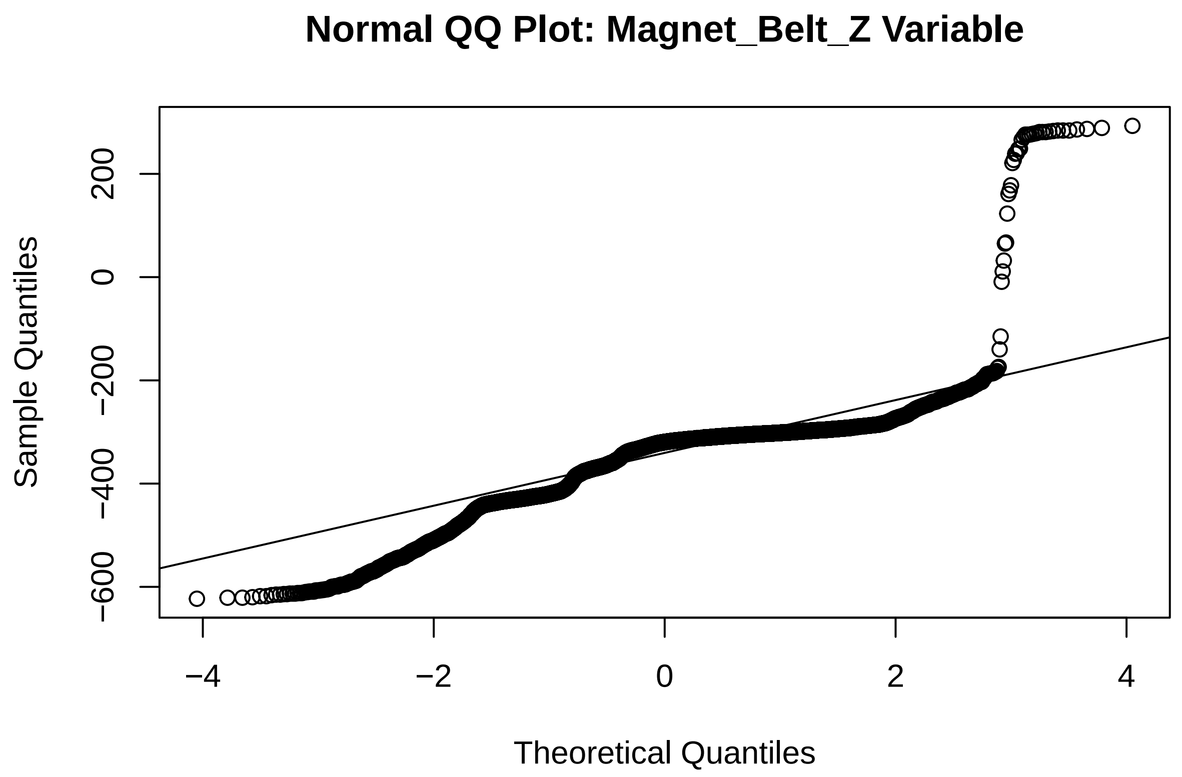
Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. Qualitative Activity Recognition of Weight Lifting Exercises. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human '13) . Stuttgart, Germany: ACM SIGCHI, 2013 http://groupware.les.inf.puc-rio.br/har#ixzz5Fc5QoJtY

Appendix A

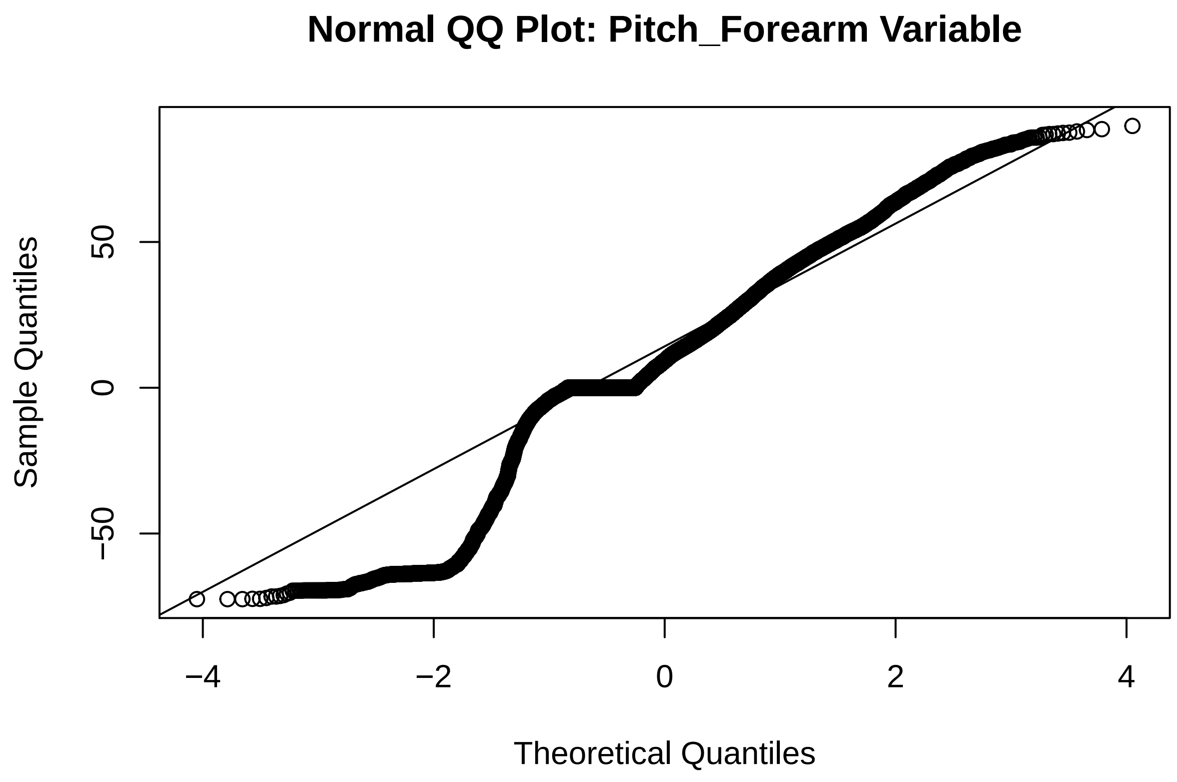
```
barplot(table(trainingDataSet$classe), main = "Class Balance", xlab = "Classe Variable", ylab = "Count"
```



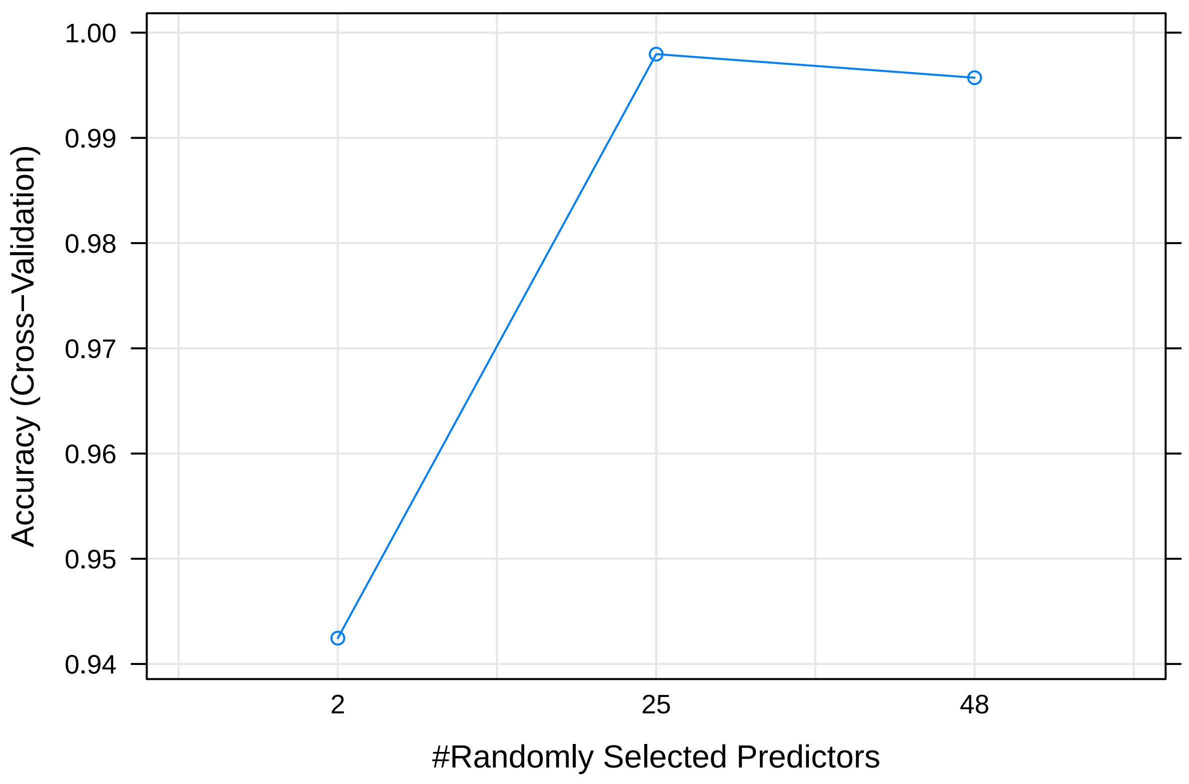
```
qqnorm(trainingDataSet$magnet_belt_z, main = "Normal QQ Plot: Magnet_Belt_Z Variable")
qqline(trainingDataSet$magnet_belt_z)
```



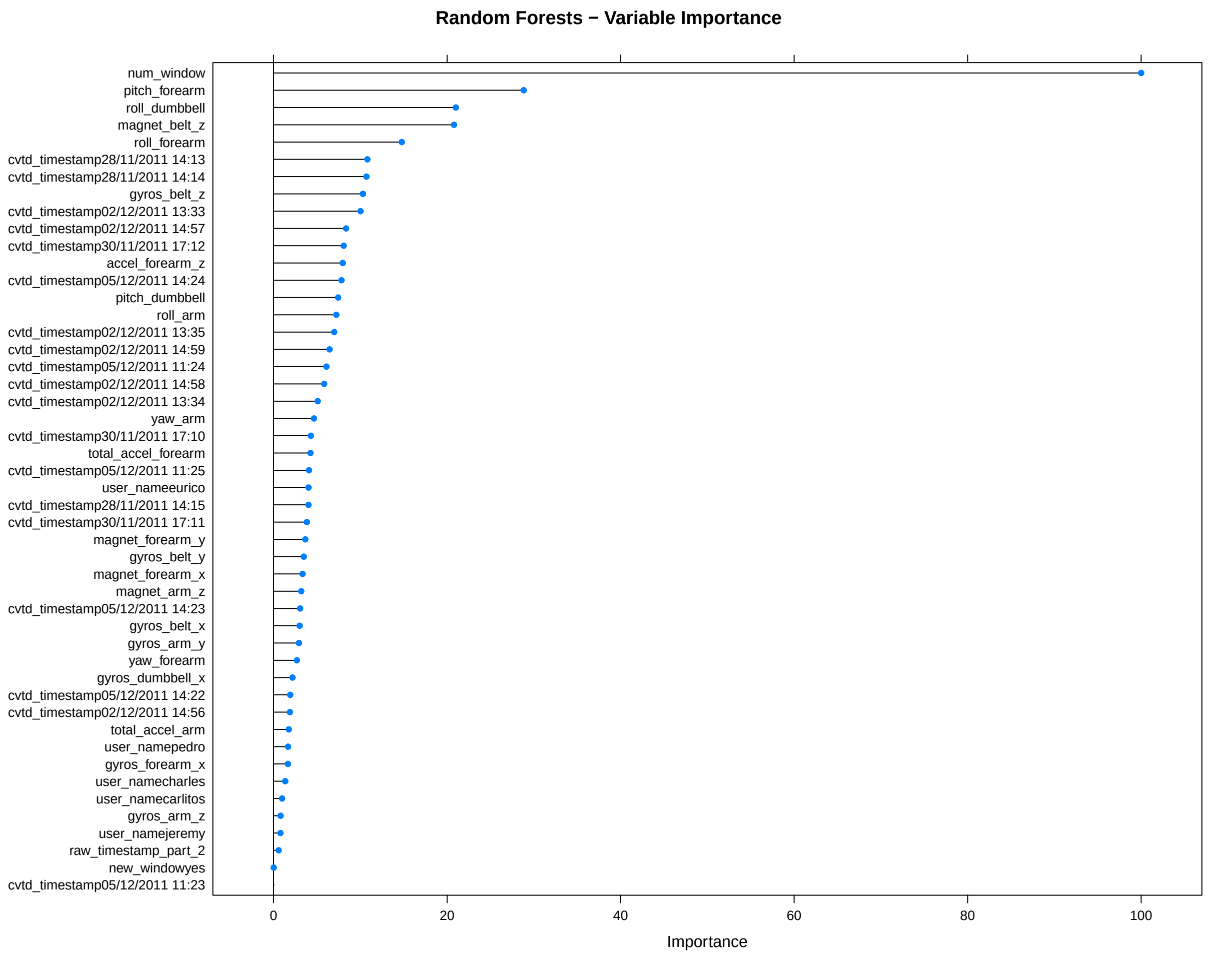
```r
qqnorm(trainingDataSet$pitch_forearm, main = "Normal QQ Plot: Pitch_Forearm Variable")
qqline(trainingDataSet$pitch_forearm)
```



```r
plot.train(trained, plotType = "line", metric = "Accuracy", nameInStrip = TRUE)
```



```r
plot(varImp(trained),main="Random Forests - Variable Importance")
```


Random Forests − Variable Importance
num_window
pitch_forearm
roll_dumbbell
magnet_belt_z
roll_forearm
cvtd_timestamp28/11/2011 14:13
cvtd_timestamp28/11/2011 14:14
gyros_belt_z
cvtd_timestamp02/12/2011 13:33
cvtd_timestamp02/12/2011 14:57
cvtd_timestamp30/11/2011 17:12
accel_forearm_z
cvtd_timestamp05/12/2011 14:24
pitch_dumbbell
roll_arm
cvtd_timestamp02/12/2011 13:35
cvtd_timestamp02/12/2011 14:59
cvtd_timestamp05/12/2011 11:24
cvtd_timestamp02/12/2011 14:58
cvtd_timestamp02/12/2011 13:34
yaw_arm
cvtd_timestamp30/11/2011 17:10
total_accel_forearm
cvtd_timestamp05/12/2011 11:25
user_nameeurico
cvtd_timestamp28/11/2011 14:15
cvtd_timestamp30/11/2011 17:11
magnet_forearm_y
gyros_belt_y
magnet_forearm_x
magnet_arm_z
cvtd_timestamp05/12/2011 14:23
gyros_belt_x
gyros_arm_y
yaw_forearm
gyros_dumbbell_x
cvtd_timestamp05/12/2011 14:22
cvtd_timestamp02/12/2011 14:56
total_accel_arm
user_namepedro
gyros_forearm_x
user_namecharles
user_namecarlitos
gyros_arm_z
user_namejeremy
raw_timestamp_part_2
new_windowyes
cvtd_timestamp05/12/2011 11:23
0
20
40
60
80
100
Importance