Predicting the Correctness of Barbell Lifting

Rodrigo Suzuki Okada

12/2/2016

Introduction

Background

The advent of wearable devices has made it possible to collect large amounts of data from personal activity, which is often used to evaluate personal health and how it could be improved. However, most analysis evaluate how much of a particular activity a person does, but they rarely quantify how well they do it.

The objective of this analysis is to use data from accelerometers on the belt, forearm, arm, and dumbbell of 6 participants, and evaluate if they perform barbell lifting in a correct manner. The data itself is available here.

This project is part of the Coursera's *Practical Machine Learning* course, available here.

Data

The data for this analysis can be found below:

```
downloadIfDoesNotExist <- function(filename, url) {
    if (!file.exists(filename)) { download.file(url, filename, quiet = T) }
}
downloadIfDoesNotExist("pml-training.csv", "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-trai
downloadIfDoesNotExist("pml-testing.csv", "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testi
```

The data can be loaded using the script below. The *na.strings* option is there to remove the invalid values present on the data - particularly, blank spaces and **#DIV/0**.

```
training <- read.csv("pml-training.csv", na.strings = c('#DIV/0!', '', 'NA'), stringsAsFactors = T)
testing  <- read.csv("pml-testing.csv", na.strings = c('#DIV/0!', '', 'NA'), stringsAsFactors = T)
```

The training data has `ncol(training)` columns and `nrow(training)` samples, while the test data has `nrow(testing)` samples. A summary of the training set is given below.

```
str(training, list.len = 10)
```

```
## 'data.frame':    19622 obs. of  160 variables:
##  $ X                       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ user_name               : Factor w/ 6 levels "adelmo","carlitos",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ raw_timestamp_part_1    : int  1323084231 1323084231 1323084231 1323084232 1323084232 1323084232 
##  $ raw_timestamp_part_2    : int  788290 808298 820366 120339 196328 304277 368296 440390 484323 4844
##  $ cvtd_timestamp          : Factor w/ 20 levels "02/12/2011 13:32",..: 9 9 9 9 9 9 9 9 9 9 ...
##  $ new_window              : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ num_window              : int  11 11 11 12 12 12 12 12 12 12 ...
##  $ roll_belt               : num  1.41 1.41 1.42 1.48 1.48 1.45 1.42 1.42 1.43 1.45 ...
##  $ pitch_belt              : num  8.07 8.07 8.07 8.05 8.07 8.06 8.09 8.13 8.16 8.17 ...
##  $ yaw_belt                : num  -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 ...
##   [list output truncated]
```

The correctness of the exercise is given by the *classe* variable. Class A corresponds to the correct movement, while the other 4 classes correspond to common mistakes while exercising. A quick analysis shows that those classes are well-balanced, even across test subjects.

```
summary(training$classe)
```

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

```
prop.table(table(training$user_name, training$classe), 1)
```

```
## 
##                    A         B         C         D         E
##   adelmo   0.2993320 0.1993834 0.1927030 0.1323227 0.1762590
##   carlitos 0.2679949 0.2217224 0.1584190 0.1561697 0.1956941
##   charles  0.2542421 0.2106900 0.1524321 0.1815611 0.2010747
##   eurico   0.2817590 0.1928339 0.1592834 0.1895765 0.1765472
##   jeremy   0.3459730 0.1437390 0.1916520 0.1534392 0.1651969
##   pedro    0.2452107 0.1934866 0.1911877 0.1796935 0.1904215
```

Data Manipulation

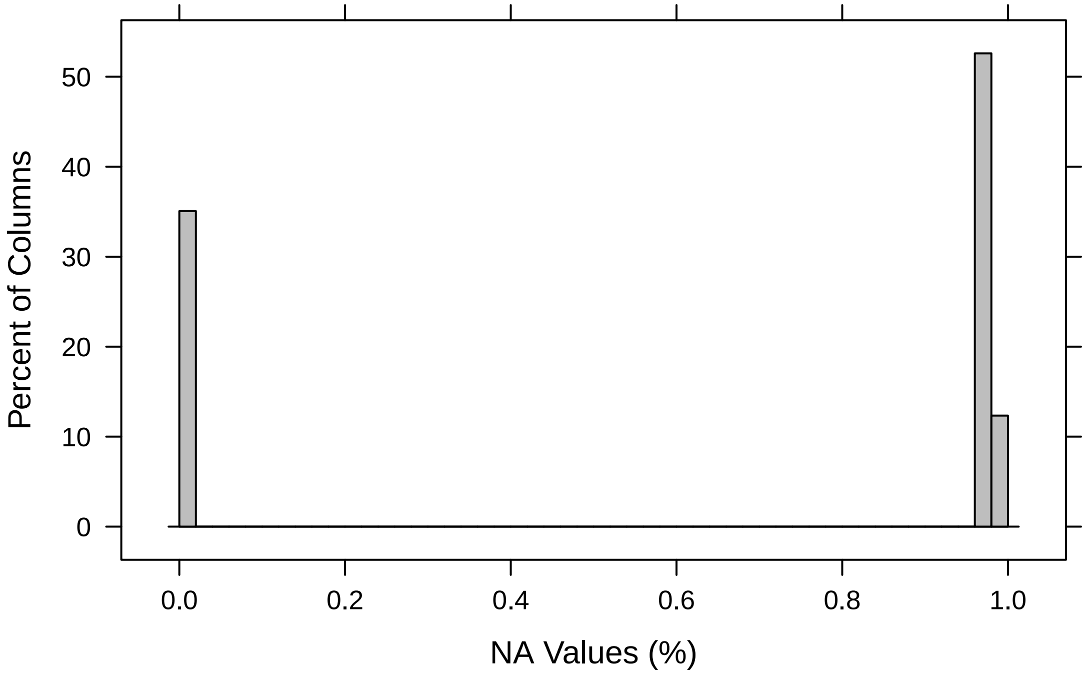
Cleaning

Based on the data description, we notice that the first six columns are there for reference purposes, and does not help explaining the classes. We can safely remove them from the data.

```
training <- training[,7:160]
testing  <- testing[,7:160]
```

Furthermore, we check how much valid data each column has.

```
library(lattice)
na.count <- function(x) { sum(is.na(x)) / nrow(training) }
training_na_count <- sapply(training, na.count)
histogram(training_na_count, breaks = 50, xlab = "NA Values (%)", ylab = "Percent of Columns", col = "g
```



It is pretty noticeable that more than half of the columns are primarily made of NA's, making them rather questionable about their usefulness. We'll be removing them for this analysis.

```
training <- training[,training_na_count < 0.9]
testing  <- testing[,training_na_count < 0.9]
dim(training)
```

```
## [1] 19622    54
```

```
dim(testing)
```

```
## [1] 20 54
```

Partitioning

We use the *caret* library to split the training data into two to perform cross-validation, sampling 60% of the data to train our classifier and using the remaining 40% to evaluate its accuracy.

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.2.5

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 3.2.4
```

```
set.seed(12345)
inTrain <- createDataPartition(y = training$classe, p = 0.6, list = F)
training_real <- training[inTrain,]  # Training
training_cv   <- training[-inTrain,] # Cross Validation
```

Near-Zero Variance Predictors

Before training the classifiers, we will check if there are any predictors with very low variances. This usually happens for predictors that have few unique values and a large gap between the first and second most frequent values.

```
nzv <- nearZeroVar(training_real)
nzv
```

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

This output has shown that there are no near-zero variance predictors, therefore, no column needs to be removed from the training set.

Column Selection

Before training our model, we can check which variables seem to have higher influence over the output *classe*. We can use Random Forest algorithm to evaluate their importance and select the most influential ones.

```
library(randomForest)
```

```
## randomForest 4.6-12
```

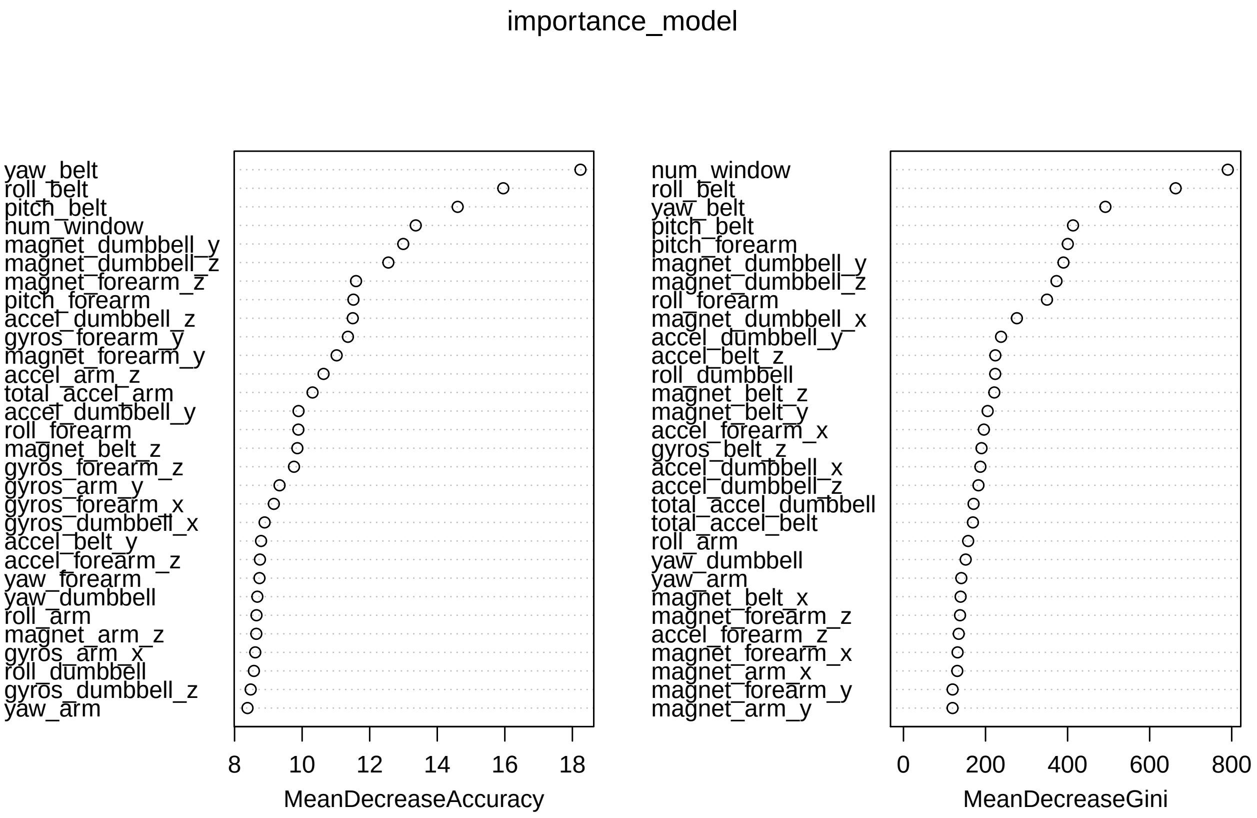
```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
set.seed(31415)

importance_model <- randomForest(classe ~ ., data = training_real, importance = T, ntree = 50)
varImpPlot(importance_model)
```



```
varImp(importance_model)
```

```
##                              A         B         C         D         E
## num_window            8.506513 12.478511 15.072276 12.537799 12.850516
## roll_belt            12.322822 14.288402 13.860658 13.178874 11.088677
## pitch_belt            8.759003 15.546158 10.848444 10.426269 10.264981
## yaw_belt             11.305432 12.940081 11.703043 13.846507  8.238146
## total_accel_belt      5.617896  5.510344  5.025322  5.750641  5.444048
## gyros_belt_x          5.592000  4.237510  4.690866  4.163251  4.165610
## gyros_belt_y          4.064902  5.007972  5.721609  5.776675  5.087273
## gyros_belt_z          5.642839  6.890177  5.488009  5.247560  6.496329
## accel_belt_x          4.305310  5.838110  4.492446  4.971489  4.788559
## accel_belt_y          4.530977  4.856817  4.826197  5.460508  4.536894
## accel_belt_z          5.040195  7.593112  8.059260  6.995766  6.745618
## magnet_belt_x         6.510470  6.021210  6.041211  5.899016  7.981670
## magnet_belt_y         5.816590  8.809511  7.532216  7.317972  6.255713
## magnet_belt_z         6.768080  8.232513  8.017305  9.229607  7.900909
## roll_arm              5.165784  8.792443  7.891720  7.147944  5.504490
## pitch_arm             4.692148  8.292594  7.436554  6.553575  5.409820
## yaw_arm               6.565185  7.182251  7.744596  7.006779  5.529317
## total_accel_arm       5.276995  7.653858  9.332542  7.022213  5.796652
## gyros_arm_x           5.434353  7.549688  6.255283  5.785414  8.328331
## gyros_arm_y           5.778666  7.385144  6.289846  8.262626  5.226462
## gyros_arm_z           2.521945  4.590474  4.743460  3.469682  5.019231
## accel_arm_x           4.816497  4.732318  5.085137  7.069308  5.336037
## accel_arm_y           4.507908  6.232780  4.935647  5.199972  5.015455
## accel_arm_z           6.677476  6.948416  7.129739  7.768955  6.073459
## magnet_arm_x          4.867052  5.200086  5.489852  5.298844  4.673022
## magnet_arm_y          4.045987  6.204823  6.805912  6.922113  4.685969
## magnet_arm_z          5.047550  5.983027  7.011416  5.974544  5.024035
## roll_dumbbell         6.433864  8.785277  7.936737  8.377523  7.626021
## pitch_dumbbell        3.351885  6.832187  5.348362  3.770767  3.807018
## yaw_dumbbell          5.895885  7.507033  7.172843  6.089946  7.210426
## total_accel_dumbbell  5.088256  8.070629  5.527155  7.388601  6.310472
## gyros_dumbbell_x      4.817706  6.974059  6.063647  6.871186  4.825881
## gyros_dumbbell_y      5.536074  5.708886  8.022547  5.881578  6.155279
## gyros_dumbbell_z      4.665271  5.997897  4.361781  5.179033  4.416961
## accel_dumbbell_x      5.148810  7.358790  5.769730  5.843072  5.848550
## accel_dumbbell_y      7.443008  8.332757  8.163134  8.848924  8.637056
## accel_dumbbell_z      6.383625  8.823507  8.330769  6.519280  9.851789
## magnet_dumbbell_x     6.505169  7.583394  8.574476  7.673444  6.402133
## magnet_dumbbell_y    10.649447 12.619648 13.621259  9.278145  9.991395
## magnet_dumbbell_z    11.353252 10.410265 10.324725  9.490620  9.819020
## roll_forearm          9.815913  8.538230 10.675585  8.015292  8.121008
## pitch_forearm         9.006985  8.406576 10.602332 11.588059 10.168749
## yaw_forearm           6.203887  6.813968  6.648448  5.452517  6.044098
## total_accel_forearm   5.237648  7.237108  5.899839  5.033773  4.791802
## gyros_forearm_x       3.598897  5.064726  6.326276  5.630954  4.100437
## gyros_forearm_y       4.862276  6.948296  7.981179  5.710216  6.020178
## gyros_forearm_z       4.559854  7.546548  7.592516  5.429648  6.331120
## accel_forearm_x       5.365184  7.624133  7.207399  8.125403  6.506567
## accel_forearm_y       5.460611  5.794866  6.841307  4.660770  5.459344
## accel_forearm_z       4.820317  6.893075  6.656496  8.201387  8.243119
## magnet_forearm_x      5.109221  6.520685  6.882901  6.111624  6.834699
## magnet_forearm_y      8.529038  7.997600  7.559826  6.941048  7.926269
## magnet_forearm_z      6.753437 10.043742  8.153076  7.469919  7.724833
```

If we limit our classifier to use 10 variables, we can select *yaw_belt*, *roll_belt*, *pitch_belt*, *num_window*, *magnetic_dumbbell_y*, *magnetic_dumbbell_z*, *magnetic_forearm_z*, *pitch_forearm*, *accel_dumbbell_z* and *gyros_forearm_y*, based on the chart above.

```
cols <- c("yaw_belt", "roll_belt", "pitch_belt", "num_window", "magnet_dumbbell_y", "magnet_dumbbell_z"
training_real <- training_real[, cols]
training_cv   <- training_cv[, cols]
```

Variable Correlations

We can also check if there seems to be a correlation among those 10 variables.

```
var_correlations <- cor(training_real[,-11])
diag(var_correlations) <- 0
which(abs(var_correlations) > 0.7, arr.ind = T)
```

```
##           row col
## roll_belt   2   1
## yaw_belt    1   2
```

Particularly, *roll_belt* and *yaw_belt* seem to be highly correlated (`cor(training_real$roll_belt, training_real$yaw_belt)`), which may bring harm to our model. As countermeasure, we will be removing the *yaw_belt* variable, which is the first column of the datasets.

```
training_real <- training_real[,-1]
training_cv <- training_cv[,-1]
```

Training

In this section we will model our classifier using Random Forest with 100 trees and measure its accuracy using the cross-validation set. We will be testing two approaches:

- Random Forest using the cleaned data as is.
- Random Forest using Principal Component Analysis.

Using the Cleaned Data as is

In this test we will build a classifier for the *classe* using the nine variables we have filtered on previous sections. This model will be evaluated using the cross-validation set and compared with the expected results using a confusion matrix.

```
set.seed(237)
model <- randomForest(classe ~ ., training_real, ntree = 100)
cv_result <- predict(model, training_cv)
model_matrix <- confusionMatrix(training_cv$classe, cv_result)
model_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2232    0    0    0    0
##          B    1 1517    0    0    0
##          C    0    2 1366    0    0
##          D    0    0    1 1285    0
##          E    0    0    1    0 1441
##
## Overall Statistics
##
##                Accuracy : 0.9994
##                  95% CI : (0.9985, 0.9998)
##     No Information Rate : 0.2846
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9992
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9996   0.9987   0.9985   1.0000   1.0000
## Specificity            1.0000   0.9998   0.9997   0.9998   0.9998
## Pos Pred Value         1.0000   0.9993   0.9985   0.9992   0.9993
## Neg Pred Value         0.9998   0.9997   0.9997   1.0000   1.0000
## Prevalence             0.2846   0.1936   0.1744   0.1638   0.1837
## Detection Rate         0.2845   0.1933   0.1741   0.1638   0.1837
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9998   0.9993   0.9991   0.9999   0.9999
```

The accuracy of the model has been estimated to be `as.numeric(model_matrix$overall[1]*100)`%. It is safe to say that the model's accuracy is much better than the proportion of the most frequent class (*No Information Rate*), as its p-value is very low.

Using Principal Component Analysis

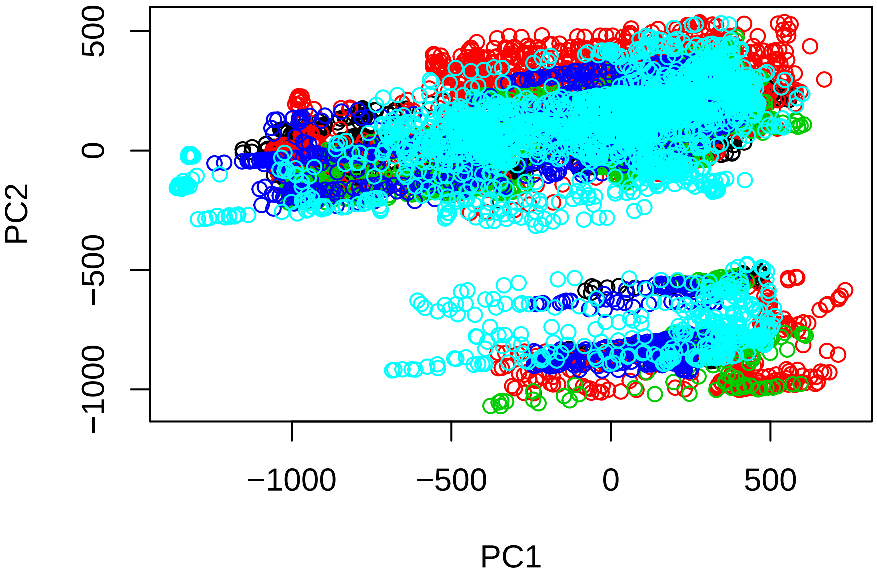
In this test we will build a classifier for the *classe* using Principal Component Analysis (PCA) to create new variables based on the variables we have filtered.

```
prComp <- prcomp(training_real[,-10])
prComp
```

```
## Standard deviations:
## [1] 382.707628 341.238046 231.663702 110.663996  86.436452  52.889703
## [7]  26.503820  13.144654   3.577561
##
## Rotation:
##                            PC1           PC2           PC3          PC4
## roll_belt          0.0542643261 -0.0442386830  4.850207e-02 -0.227635571
## pitch_belt        -0.0015852838 -0.0252289953  9.447540e-03 -0.062460836
## num_window         0.0936234056  0.3459674709  9.323177e-01  0.035447404
## magnet_dumbbell_y -0.1217802056  0.9172133219 -3.189075e-01 -0.178399203
## magnet_dumbbell_z -0.2362340814  0.0835654016 -2.738890e-02  0.868695170
## magnet_forearm_z   0.9529103352  0.0915696095 -1.325302e-01  0.165632870
## pitch_forearm      0.0007086310 -0.0141122653  6.116468e-03 -0.065637451
## accel_dumbbell_z  -0.0980395260 -0.1444482731  9.103981e-02 -0.353304407
## gyros_forearm_y    0.0004585762  0.0001964059  2.581670e-06  0.000267332
##                           PC5          PC6           PC7           PC8
## roll_belt         -0.055021136  0.942490880  0.0117625185  0.2224227846
## pitch_belt         0.083220922 -0.237602680 -0.0871629514  0.9613092561
## num_window        -0.015235829 -0.028574988 -0.0025985024 -0.0036034386
## magnet_dumbbell_y  0.097501756  0.025870379  0.0015979725  0.0135131890
## magnet_dumbbell_z  0.346981025  0.230267921 -0.0418584953  0.0815936306
## magnet_forearm_z   0.196263555  0.007532586 -0.0009876989  0.0008342533
## pitch_forearm      0.057272802  0.021107684 -0.9913768694 -0.0943399840
## accel_dumbbell_z   0.904494776 -0.015386696  0.0875613023 -0.1019177450
## gyros_forearm_y    0.001428585  0.002785893  0.0026017791 -0.0164788403
##                            PC9
## roll_belt         -1.132286e-03
## pitch_belt        -1.663584e-02
## num_window         7.402456e-05
## magnet_dumbbell_y  6.863294e-05
## magnet_dumbbell_z -1.760657e-04
## magnet_forearm_z   7.840637e-04
## pitch_forearm     -9.042156e-04
## accel_dumbbell_z   2.989471e-03
## gyros_forearm_y   -9.998558e-01
```

Using the first two components, we can plot a chart, to visually check if we can actually explain the output from those two alone.

```
typeColor <- training_real$classe
plot(prComp$x[,1], prComp$x[,2], col=typeColor, xlab = "PC1", ylab = "PC2")
```



This chart has shown that the first two components can explain part of the data, but the amount of overlap hints that the other components will be necessary to build our model.

```
set.seed(237)
preproc_pca <- preProcess(training_real[,-10], method = "pca", pcaComp = 10)
train_pca <- predict(preproc_pca, training_real[,-10])
model_pca <- randomForest(training_real$classe ~ ., train_pca, ntree = 100)

cv_pca <- predict(preproc_pca, training_cv)
cv_pca_result <- predict(model_pca, cv_pca)
confusionMatrix(training_cv$classe, cv_pca_result)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2196   28    4    4    0
##          B   30 1421   48   11    8
##          C    2   29 1312   18    7
##          D    5    3   48 1220   10
##          E    1   17    7   16 1401
##
## Overall Statistics
##
##                Accuracy : 0.9623
##                  95% CI : (0.9578, 0.9664)
##     No Information Rate : 0.2847
##     P-Value [Acc > NIR] : < 2.2e-16
##
```

```
##                   Kappa : 0.9523
##  Mcnemar's Test P-Value : 0.001086
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9830   0.9486   0.9246   0.9614   0.9825
## Specificity            0.9936   0.9847   0.9913   0.9900   0.9936
## Pos Pred Value         0.9839   0.9361   0.9591   0.9487   0.9716
## Neg Pred Value         0.9932   0.9878   0.9835   0.9925   0.9961
## Prevalence             0.2847   0.1909   0.1809   0.1617   0.1817
## Detection Rate         0.2799   0.1811   0.1672   0.1555   0.1786
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9883   0.9667   0.9579   0.9757   0.9880
```

Interestingly, the model's accuracy is lower than the Random Forest using the filtered data as is.

Prediction

Using the first model, the predictions of the 20 test samples can be made using the following script.

```
predictions <- predict(model, testing)
```

Comparing those results to the expected values available on the course's prediction quiz, we can confirm that all 20 have been correctly predicted. We have not disclosed the actual predictions, as it is part of the course's grading process.