Quality of Personal Activity Project

Introduction

With the increase of the widespread use of personal activity tracking devices, it is now possible to collect a large amount of data about personal activity relatively inexpensive. One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways.

In this project, our goal is to build a machine learning model and correctly identify 20 test cases available in the test dataset.

Project Overview

Since the goal is to correctly identify 20 test cases we need not bother with the interpretability and can choose the best performing model. To replicate the real world application we will partition train dataset into training (80% of the training data) and validation dataset (20% of the training data), and save the true test dataset (with 20 subjects) untill the very end.

Feature selection. Again, since the goal of this project isto correctly identify the 20 subjects we will ignore columns with more than 19 NAs in the final testing dataset. Thus we train our model on the columns that are present and no empty in the final test dataset.

We are going to consider the most accurate models: random forsts, gradient boosting with trees, and bagging.

Data Set Up and Exploration

Data for this project comes from Groupware: [http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har)]

```
if(!file.exists('pers_act_train.csv')){
        download.file('https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.
csv', 'pers_act_train.csv')
}
if(!file.exists('pers_act_test.csv')){
        download.file('https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.c
sv', 'pers_act_test.csv')
}
training <- read.csv('pers_act_train.csv')
testing <- read.csv('pers_act_test.csv')
```

Coding Housekeepng

```
set.seed(111)
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library(dplyr)
```

```
##
## 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
```

```
library(ggplot2)
library(parallel)
library(doParallel)
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

Variables of Interest

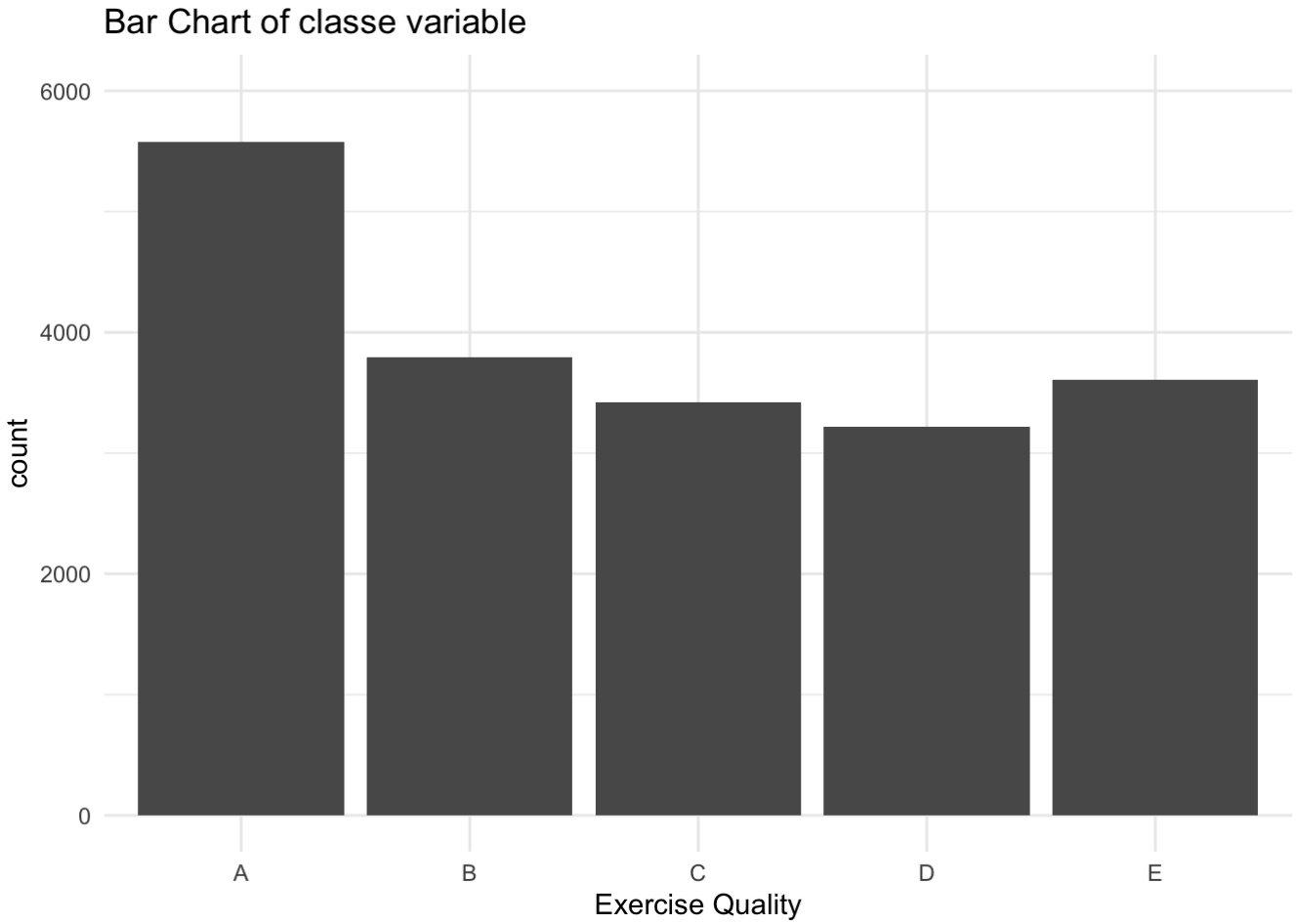
Want to ensure we have the same variables in both datasets and we will set up training and validation sets. We are going to train and validate our models on to_train dataset split 80% to 20% and save the to_test dataset for the final prediction.

```
to_test <- testing[,colSums(is.na(testing))<20 ]
to_test <- subset(to_test, select = -c(X,user_name,raw_timestamp_part_1,raw_timestamp_pa
rt_2,cvtd_timestamp, new_window,num_window, problem_id))
cols_to_keep <- names(to_test)
to_train <- select(training, append(cols_to_keep,'classe'))
```

Exploratory Data Anaysis

Before we begin with model fitting we need to understand what we are predicting and what out dataset looks like:

```
ggplot(data = to_train, aes(x=classe)) + geom_bar(position='dodge') + xlab('Exercise Qua
lity') + ggtitle('Bar Chart of classe variable') + theme_minimal() + ylim(0, 6000)
```



The most highly occuring class is A ~5500 times. Apart form A being overrepresented there are no other issues with predicted variable representation.

Model Fitting

Principal Component Analysis

We have 59 variables which is a lot and can potentially introduce noise. We do not care about interpretability in this case, so we will use PCA to reduce number of features:

```
train_index <- createDataPartition(to_train$classe, p = 0.8, list = FALSE, )
to_train_s <- to_train[train_index,]
to_validate_s <- to_train[-train_index,]

pca_process <- preProcess(subset(to_train_s, select = -c(classe)), method = 'pca', thres
h = 0.95)
pca_train <- predict(pca_process, subset(to_train_s, select = -c(classe)))
pca_validation <- predict(pca_process, subset(to_validate_s, select = -c(classe)))
pca_test <- predict(pca_process, to_test)

print(dim(pca_train))
```

```
## [1] 15699    25
```

We now have 25 variables to work with that explain 95% of variance of the original feature set.

Building models

We are going to focus on two best performing models: gradient boosted trees and random forest. Let's train these models separately.

```
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)

fitControl <- trainControl(method = "repeatedcv", number = 5, allowParallel = TRUE)
mtry <- floor(sqrt(ncol(pca_train)))+2
tunegrid <- expand.grid(.mtry=mtry, .ntree = 700)
rf_model <- train(classe ~ ., method = 'rf', metric = "Accuracy", data = data.frame(pca_
train, classe=to_train_s$classe) , trControl = fitControl)

stopCluster(cluster)
registerDoSEQ()
```

```
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)

fitControl <- trainControl(method = "repeatedcv", number = 5, allowParallel = TRUE)
gbm_model <- train(classe ~ ., method = 'gbm', metric = "Accuracy", data = data.frame(pc
a_train, classe=to_train_s$classe), trControl = fitControl, tuneGrid = expand.grid(inter
action.depth = 7, n.trees = 140, shrinkage = 0.23, n.minobsinnode = 10), verbose = FALSE
)

stopCluster(cluster)
registerDoSEQ()
```

Evaluation of the Random Forest and GBM:

```
confusionMatrix(predict(rf_model, pca_validation), to_validate_s$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1111   17    4    0    0
##          B    0  733    7    0    4
##          C    1    8  664   31   11
##          D    1    0    7  610    6
##          E    3    1    2    2  700
##
## Overall Statistics
##
##                Accuracy : 0.9732
##                  95% CI : (0.9677, 0.9781)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9661
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9955   0.9657   0.9708   0.9487   0.9709
## Specificity            0.9925   0.9965   0.9843   0.9957   0.9975
## Pos Pred Value         0.9814   0.9852   0.9287   0.9776   0.9887
## Neg Pred Value         0.9982   0.9918   0.9938   0.9900   0.9935
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2832   0.1868   0.1693   0.1555   0.1784
## Detection Prevalence   0.2886   0.1897   0.1823   0.1591   0.1805
## Balanced Accuracy      0.9940   0.9811   0.9775   0.9722   0.9842
```

```
confusionMatrix(predict(gbm_model, pca_validation), to_validate_s$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1091   43   11    6    3
##          B   13  678   21    6   16
##          C    5   27  634   31   23
##          D    6    4   15  596   19
##          E    1    7    3    4  660
##
## Overall Statistics
##
##                Accuracy : 0.9327
##                  95% CI : (0.9244, 0.9403)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9148
##
##  Mcnemar's Test P-Value : 3.553e-08
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9776   0.8933   0.9269   0.9269   0.9154
## Specificity            0.9776   0.9823   0.9734   0.9866   0.9953
## Pos Pred Value         0.9454   0.9237   0.8806   0.9312   0.9778
## Neg Pred Value         0.9910   0.9746   0.9844   0.9857   0.9812
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2781   0.1728   0.1616   0.1519   0.1682
## Detection Prevalence   0.2942   0.1871   0.1835   0.1631   0.1721
## Balanced Accuracy      0.9776   0.9378   0.9502   0.9567   0.9554
```

Random forest performs much better than the gradient boosting. Let's now tune random forest's paramters, mtry and ntree

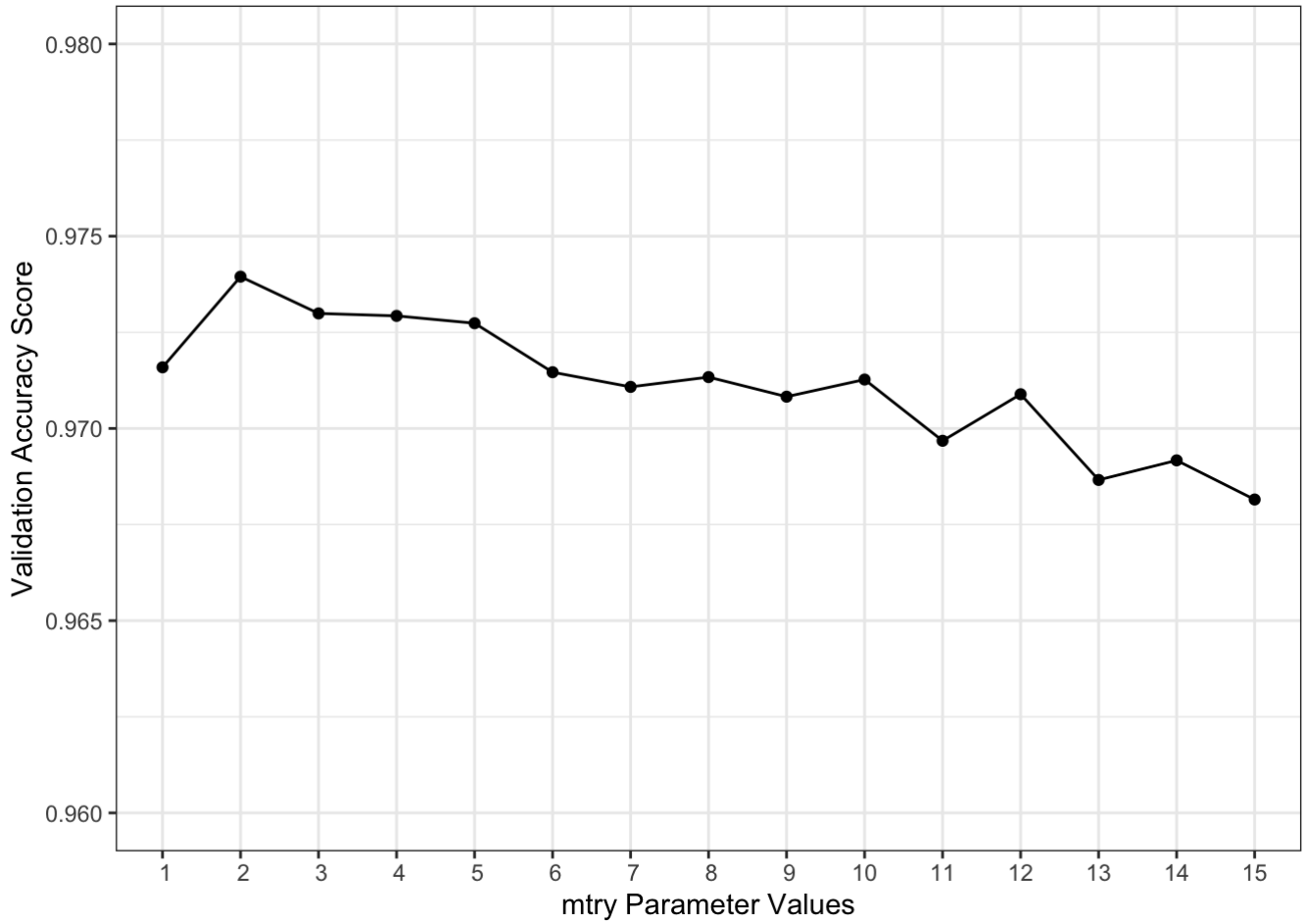
Tuning rf_model:

```
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)

control <- trainControl(method = 'repeatedcv', number = 5, search = 'grid', allowParalle
l = TRUE)
tunegrid <- expand.grid(.mtry=c(1:15))
rf_gridsearch <- train(classe ~ ., method = 'rf', metric = "Accuracy", data = data.frame
(pca_train, classe=to_train_s$classe) , trControl = control, tuneGrid=tunegrid, .ntree =
700)

stopCluster(cluster)
registerDoSEQ()

qplot(rf_gridsearch$results$mtry, rf_gridsearch$results$Accuracy) + geom_line() +
        ylab("Validation Accuracy Score") +
        theme_bw()+
        scale_x_discrete(name = "mtry Parameter Values", limits= c(1:15))+
        ylim(0.96, 0.98)
```



Best paramter for mtry is 2

Evaluation of the Random Forest:

```
confusionMatrix(predict(rf_gridsearch, pca_validation), to_validate_s$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1111   17    1    0    0
##          B    0  733    9    0    3
##          C    1    8  667   29   12
##          D    2    0    4  612    7
##          E    2    1    3    2  699
##
## Overall Statistics
##
##                Accuracy : 0.9743
##                  95% CI : (0.9688, 0.979)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9674
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9955   0.9657   0.9751   0.9518   0.9695
## Specificity            0.9936   0.9962   0.9846   0.9960   0.9975
## Pos Pred Value         0.9841   0.9839   0.9303   0.9792   0.9887
## Neg Pred Value         0.9982   0.9918   0.9947   0.9906   0.9932
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2832   0.1868   0.1700   0.1560   0.1782
## Detection Prevalence   0.2878   0.1899   0.1828   0.1593   0.1802
## Balanced Accuracy      0.9946   0.9810   0.9799   0.9739   0.9835
```

Prediction on the Test Set

After perfomring cross validation we now predict classes of the test set at the very end:

```
predict(rf_gridsearch, pca_test)
```

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