Practical Machine Learning Project Report

Ioannis Nennes

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. These type of devices are part of the quantified self movement – a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks. 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. In this project, your goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. More information is available from the website here. (see the section on the Weight Lifting Exercise Dataset).

Loading Data

The training data for this project are available here. The test data are available here. The data for this project come from this source. If you use the document you create for this class for any purpose please cite them as they have been very generous in allowing their data to be used for this kind of assignment.

The above data has been downloaded to the 'data' directory. The following code loads the data and the necessary libraries.

```r
suppressMessages(library(knitr, warn.conflicts = FALSE, quietly=TRUE))

opts_chunk$set(
  cache=TRUE
  , dev='png'
  , dev.args=list(type="cairo")
  , fig.height=4
  ,dpi=300
  , warning=FALSE
  , message=FALSE
  , results="hide"
  , echo=TRUE
  , eval=TRUE
  , xtable.type="html")
knit_hooks$set(inline = function(x) {
  prettyNum(x, big.mark=",")
})

# Library the required dependencies
suppressMessages(library(caret, warn.conflicts = FALSE, quietly=TRUE))
suppressMessages(library(plyr, warn.conflicts = FALSE, quietly=TRUE))
suppressMessages(library(dplyr, warn.conflicts = FALSE, quietly=TRUE))
suppressMessages(library(randomForest, warn.conflicts = FALSE, quietly=TRUE))
suppressMessages(library(doParallel, warn.conflicts = FALSE, quietly=TRUE))

#load raw data
setwd("C:/Users/John/Dropbox/Code/GitHub/ML-coursera")
training <- read.csv("data/pml-training.csv", header = TRUE, na.strings=c("NA","","#DIV/0!"))
```

```r
testing  <- read.csv('data/pml-testing.csv', header = TRUE, na.strings=c("NA","","#DIV/0!"))

#set the classe feature type
training$classe <- as.factor(training$classe)
```

Cleaning Data

After examining the data, we have identified a number of actions required to clean the dataset:

- There are a few columns where the majority of the values are NA. We will remove columns where there are more than 50% NA values.
- We will also remove the first 7 columns, as they do not contain measurements.
- Near zero variance features will be removed.
- Due to the sheer amount of featured, we will perform PCA to reduce the dimensionality and cosider the effectiveness of the predictions.

```r
# We will remove columns where there are more than 50% NA values.
na.rich.cols<- apply(training,2,function(x) {sum(is.na(x))});
training <- training[,which(na.rich.cols <  nrow(training)*0.5)];

# We wil also remove the first 7 columns, as they do not contain measurements.
training<-training[,8:ncol(training)]

# Near zero variance features will be removed.
nz.cols <- nearZeroVar(training, saveMetrics = TRUE)
training <- training[, nz.cols$nzv==FALSE]
```

Creating the Cross Validation set

We will use 75% of the training data set for training our model and the remaining 25% for cross validating it.

```r
set.seed(221)

training.idx <- createDataPartition(training$classe, p = 3/4, list=FALSE)
training.part <- training[training.idx,];
testing.part <- training[-training.idx,];
```

```r
# Due to the sheer amount of featured, we will perform PCA to reduce the dimensionality.
principal.components <- preProcess(training.part[, -ncol(training.part)], method=c("BoxCox", "center",
training.pc <- predict(principal.components, training.part[, -ncol(training.part)])
```

Model Comparison

We will try two models (with and without PCA): Random forest ("rf") and linear discriminant analysis ("lda"). Based on the accuracy of the predictions, we will choose the optimal.

```r
set.seed(222)

train.rf <- train(classe ~ ., method = 'rf' , data = training.part,
             # add tuning par. mtry to avoid long lasting computations for the
             # ideal mtry; take the randomforest dft: sqrt(number of features)
             tuneGrid = data.frame(mtry = floor(sqrt(ncol(training.part)-1)))
            )
```

```
train.rf.pc <- train(training.part$classe ~ ., method = 'rf' , data = training.pc)
```

```
train.lda <-train(classe ~ ., method = 'lda', data = training.part)
```

```
train.lda.pc <-train(training.part$classe ~ ., method = 'lda', data = training.pc)
```

```
predict.rf     <- predict(train.rf              , testing.part)
predict.rf.pc  <- predict(principal.components , testing.part[, -ncol(training.part)])
predict.lda    <- predict(train.lda             , testing.part)
predict.lda.pc <- predict(principal.components , testing.part[, -ncol(training.part)])
```

Accuracies:

- Random forest: 0.994
- Random forest (PCA): 0.974
- Linear discriminant analysis: 0.697
- Linear discriminant analysis (PCA): 0.521

Based on the above, we will use the Random Forest model.

Tuning with Cross Validation

To avoid over-fitting, we will use cross-validation on the test set created inside the original training set.

```
set.seed(223)

fit.control <- trainControl(method = "repeatedcv", number = 5, repeats = 1)
model.rf.cv <- train(classe ~ ., method="rf",  data=training.part, trControl = fit.control,
                 # add tuning par. mtry to avoid long lasting computations for the
                 # ideal mtry; take the randomforest dft: sqrt(number of features)
                 tuneGrid = data.frame(mtry = floor(sqrt(ncol(training.part)-1)))
                )

predict.rf.cv <<- predict(model.rf.cv , testing.part)
```

Tuning proved (marginally) useful, as the accuracy was improved from 0.99368 to 0.99409.

Predicting on the test set

```
predict.final <<- predict(model.rf.cv , testing)
```

We will store the predictions in the working directory, using the code below:

```
answers <- as.vector(predict.final)

pml_write_files = function(x) {
    n = length(x)
    for (i in 1:n) {
        filename = paste0("problem_id_", i, ".txt")
        write.table(x[i], file = filename, quote = FALSE, row.names = FALSE,
            col.names = FALSE)
    }
}

pml_write_files(answers)
```