Coursera Machine Learning Assignment

Ralph Schlosser

Introduction

It is conventional wisdom that regular exercise and physical activity lead to a positive effect on health and overall well-being. This "wisdom" is supported and further quantified by a large body of scientific research, some of which already dates back from several decades.

Unlike previously, however, where costly long-term studies had to be set-up to record and analyze data relating to physical activity, we now live in an era where data is readily available and plentiful.

This was made possible by the advent of inexpensive self-monitoring devices such as *Fit Bit*, *Jawbone Up*, or *Nike FuelBand* which enable the carrier of this device to collect various physical parameters through e.g. gyroscopes, accelerometers, pulse meters and in some cases even GPS which are built into the device.

Goal

In this study we are considering a data set produced in the above fashion, which records parameters of 6 participants while they were performing a specific type of weight lifting exercises known as *Unilateral Dumbbell Biceps Curl*.

Participants were asked to perform the exercise in 5 different ways, only one of which being the correct way.

The intention here is to use machine-learning techniques to perform *qualitative activity recognition*, i.e. to not only be able to tell what type of exercise a participant was doing but also how well they were doing it.

The original study and data set can be found at: http://groupware.les.inf.puc-rio.br/ha

Initial exploration

The data was already broken up into training and test set data and can simply be loaded using the `read.csv2` function:

```
training.raw <- read.csv2('~/Dropbox/coursera/data_science/machine_learning/assignment/data/pml-training
test.raw <- read.csv2('~/Dropbox/coursera/data_science/machine_learning/assignment/data/pml-testing.csv
```

The above commands already perform a small amount of data cleaning in that they replace `#DIV/0!` strings with `NA`.

In total there are 19622 rows of data in the training set of 158 variables; the test set contains 20 rows.

Of particular interest for this study is the `classe` variable in the training set. This factor variable records how well the exercise was performed using levels `A` to `E`, where only `A` corresponds to a correctly performed lift exercise.

This is what we are aiming to predict.

We can look at a table for the `classe` variable.

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

Unfortunately, due to the sheer amount of covariates there is not much more exploration / plotting that would make sense at this stage.

Reducing the amount of covariates

While glancing over the data it becomes apparent that there are quite a few columns with either all `NA`, or nearly all `NA` values. As the amount of actual data values is much smaller than the amount of `NA` entries, techniques like e.g. *imputing* using means etc. cannot be applied, so I chose to drop these columns altogether and exclude them from the subsequent steps.

Interestingly, in the test data set, those same columns consist of exclusively `NA` values, so the prediction result cannot rely on the respective covariates from those columns:

```
preprocess_data <- function(in.df, drop.cols) {
  out.df <- in.df[, -drop.cols]
  t <- apply(out.df[, 1:52], 2, as.numeric)
  out.df <- data.frame(t, classe=out.df[, 53])
  return(out.df)
}

# Drop cols 1-6 and all other colls with only NA values in the test set.
drop.cols <- c(1:6,  which(apply(is.na(test.raw), 2, all)==TRUE))
training <- preprocess_data(training.raw, drop.cols)
```

Model building

For the prediction model I have decided to use the **Random Forest** algorithm. This is because I would expect to see a high **out-of-sample** prediction accuracy. Two different approaches were implemented to compare and verify prediction results. However other methods are possible too.

In order to assess the out-of-sample error rate, **cross validation** was employed in the second approach, whereas in the first approach a validation set was split from the original training set.

1. Using the `randomForest` package / Validation set

When I started experimentation with `caret`, I found that running `train` with default options seemingly "never"" terminates.

So I left a closer look at what is going on for later and turned my attention to the `randomForest` package:

```
library(caret)
library(randomForest)
in.train <- createDataPartition(y=training$classe, p=0.75, list=FALSE)
training.train <- training[in.train, ]
training.verify <- training[-in.train, ]

rf.model <- randomForest(classe ~ ., data = training, subset = in.train)
```

As can be seen from the code, a *validation* set is split from the full training set. This set contains 25% of the data.

Because the prediction accuracy on the training set is overly optimistic, this validation set is used to get an estimate of the *out of sample* estimation error:

```r
# Prediction on verification set.
predictions.verif <- predict(rf.model, newdata = training.verify)

# Print confusion matrix.
confusionMatrix(predictions.verif, training.verify$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1394    5    0    0    0
##          B    0  944    7    0    0
##          C    0    0  848    7    0
##          D    0    0    0  797    2
##          E    1    0    0    0  899
##
## Overall Statistics
##
##                Accuracy : 0.9955
##                  95% CI : (0.9932, 0.9972)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9943
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9993   0.9947   0.9918   0.9913   0.9978
## Specificity            0.9986   0.9982   0.9983   0.9995   0.9998
## Pos Pred Value         0.9964   0.9926   0.9918   0.9975   0.9989
## Neg Pred Value         0.9997   0.9987   0.9983   0.9983   0.9995
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2843   0.1925   0.1729   0.1625   0.1833
## Detection Prevalence   0.2853   0.1939   0.1743   0.1629   0.1835
## Balanced Accuracy      0.9989   0.9965   0.9950   0.9954   0.9988
```

Out-of-sample error rate estimate

The above confusion matrix yields an accuracy of $\approx 99.5\%$ with a small p-value on the held-out data. This is already quite good but only an estimate.

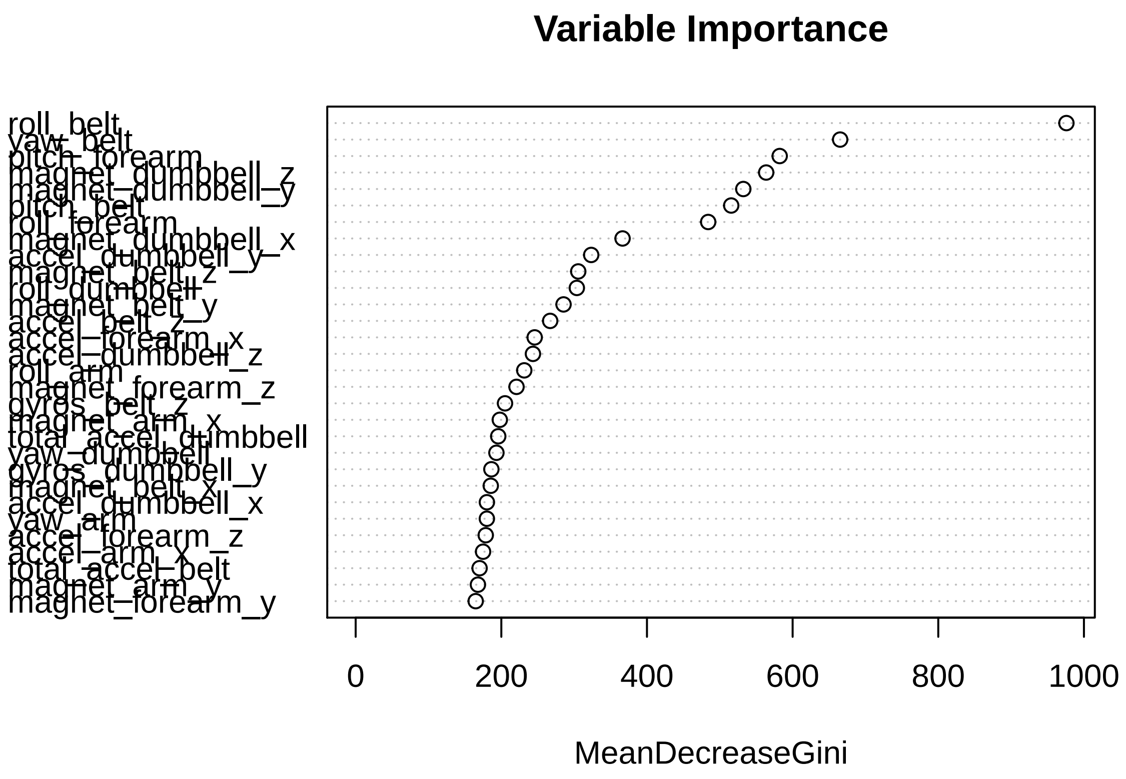
Predicting on the test set Before predicting on the test set we need to ensure that the same transformations are applied to it as to the training set.

After doing this, we can then use our model to predict how well the weight exercise was performed for the given test set:

```r
test <- preprocess_data(test.raw, drop.cols)
test <- test[, -53]
predict(rf.model, newdata = test)
```

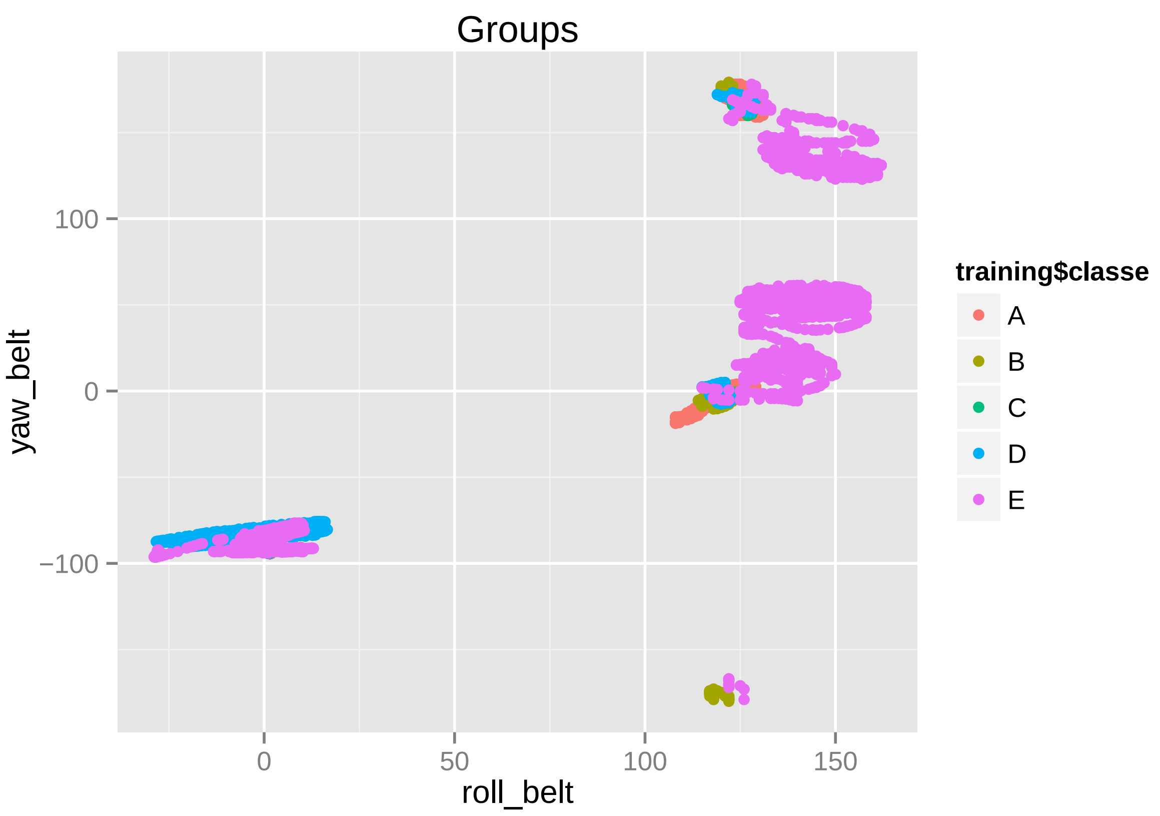
```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  B  A  B  A  A  E  D  B  A  A  B  C  B  A  E  E  A  B  B  B
## Levels: A B C D E
```

Most important variables The following plot visualizes the variable importance in decreasing order:



From this is can be seen that `roll_belt` and `yaw_belt` appear to have the biggest influence on the prediction outcome.

With this information we can look at a plot of these and see how well they separate the data:



It turns out that class is `E` pretty well separated by these two variables alone.

2. Using caret's `train` function / Cross validation

Running `train` with the `rf` method on the data using default values takes a very long time. This is because it performs **10-fold cross-validation** and for each fold, models are build for each one of 3 different tuning parameters.

However, the execution time can be significantly reduced by performing several optimizations:

- Modify default settings for `trainControl`.
- Use the `doMC` package to leverage multi-core execution.

```
library(doMC)
registerDoMC(4)
fitControl <- trainControl(method = "cv",
                           number = 10,
                           repeats = 1,
                           verboseIter = TRUE,
                           allowParallel = TRUE)
rf.model.new <- train(classe ~ ., data = training, method="rf", trControl = fitControl,
                      tuneLength = 1)

predictions.new <- predict(rf.model.new, newdata = test)
```

For 10-fold cross-validation, the final model yields an only marginally higher prediction accuracy of 99.6%. This does, however, come at the cost of a significantly longer execution time.

Predicting on the test set The results are as before:

```
predictions.new
 [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
```

Conclusions

- Using the Random Forest algorithm we were able to predict how well a participant performed their weight lifting exercise for the given data set with high accuracy.
- The high prediction accuracy seems unusual. Further study would be needed.
- But both approaches implemented concur in their prediction results and come in very close in their accuracy results.
- However, in general I would put higher trust the `train` results, as it offers more flexibility and internal validation.
- The `train` package requires a significant amount of tuning to run faster. More options could have been tested.