08 Machine Learning - Course Project

Technophobe01

2015-07-26

1 Executive Summary

The purpose of this document is to develop and answer the Coursera Peer Assessment for the Practical Machine Learning Course. This reports source code is available via **GitHub**, if you wish to review the ***code***, ***markdown formatting*** or ***html***, or ***pdf*** output in more detail.

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**.

The goal of this project is to use the referenced data captured from accelerometers on the belt, forearm, arm, and dumbell of 6 participants to *predict* how the participants performed the exercises. This report describes:

1. How a model was built to frame and analyse the data
2. How cross validation was used to confirm the results and what the estimated expected out of sample error is of the analysis
3. What choices were made to define the results and guide the analysis.

seperaralty we use the prediction data to generate

1. 20 different test case results which are to be submitted to coursera.

Note: More information is available from the website here: http://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset) on the specific exercises measured and predicted.

---

2 Data Set Description

The data used in this report was generated using a set of six male partcipants aged between 20-28 years, with little weight lifting experience. Participants were supervised by an experienced weight lifter to make sure the execution complied to the manner they were supposed to simulate. All participants were vetted to ensure they could easily simulate the mistakes in a safe and controlled manner by using a relatively light dumbbell (1.25kg).

The participants were asked to perform barbell lifts *correctly* and *incorrectly* in 5 different ways. Six young healthy participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions:

1. (Class A) - exactly according to the specification
2. (Class B) - throwing the elbows to the front
3. (Class C) - lifting the dumbbell only halfway
4. (Class D) - lowering the dumbbell only halfway

5. (Class E) - throwing the hips to the front

The data was collected from four 9 degrees of freedom Razor inertial measurement units (IMU), which provide three-axes acceleration, gyroscope and magnetometer data at a joint sampling rate of 45 Hz. Each IMU also featured a Bluetooth module to stream the recorded data to a notebook running the Context Recognition Network Toolbox. The sensors were mounted in the users' glove, armband, lumbar belt and dumbbell. The tracking system was designed to be as unobtrusive as possible, as these are all equipmentm commonly used by weight lifters.

Read more: http://groupware.les.inf.puc-rio.br/har#dataset#ixzz3gge1uKYv

2.1 Data Import / Pre-processing

The our first step was to download and load the base data from `pml-training.csv` and `pml-testing.csv` into R. Note that `trainBaseData` has **19622** objects and **160** variables, whilst `testBaseData` has **20** objects and **60** variables.

As we have access to both the train, and the test data we can deduce which columns have data in the test data set, logically those are the only ones we really want to look at in the train data set. So we strip out everything else from the train data set...

Thus, after cleaning the data note that `trainBaseData` has **19622** objects and **53** variables, whilst `testBaseData` has **20** objects and **53** variables.

2.2 Data Validation

After removal of the superfluous values, and data columns we move to validate the data in preperation for analysis... We first check for and prepare to filter out near-zero varience predictors, and validate for between-predictor correlations.

2.2.1 Near Zero Varience Predictors

To filter for near-zero variance predictors, we use the caret package function **`nearZeroVar()`** which will return the column numbers of any predictors that fulfill the conditions outlined. In this case **`nearZeroVar(trainCleanData)`** indicated that the cleaned data set contains no near-zero varience predictors.

Why check? There are potential advantages to removing predictors prior to modeling. First, fewer predictors means decreased computational time and complexity. Second, if two predictors are highly correlated, this implies that they are measuring the same underlying information. Removing one should not compromise the performance of the model and might lead to a more parsimonious and interpretable model. Third, some models can be crippled by predictors with degenerate distributions. In these cases, there can be a significant improvement in model performance and/or stability without the problematic variables.

2.2.2 Correlation Predictors

Similarly, to filter on between-predictor correlations, the `cor` function was used to calculate the correlations between predictor variables:

```
correlations <- cor(trainCleanData[,-53])
dim(correlations)
```

```
## [1] 52 52
```

To visually examine the correlation structure of the data, we used the `corrplot` package The function `corrplot` has many options including one that will reorder the variables in a way that reveals clusters of highly correlated predictors.

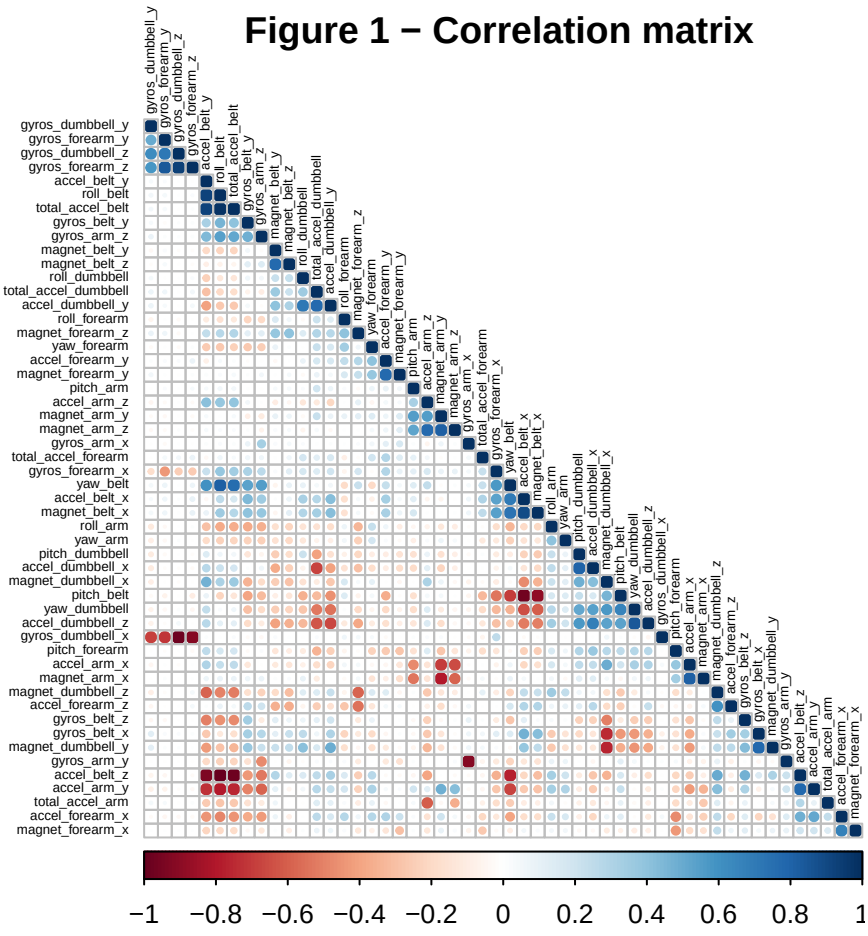
Figure 1 − Correlation matrix
gyros_dumbbell_y
gyros_forearm_y
gyros_dumbbell_z
gyros_forearm_z
accel_belt_y
roll_belt
total_accel_belt
gyros_belt_y
gyros_arm_z
magnet_belt_y
magnet_belt_z
roll_dumbbell
total_accel_dumbbell
accel_dumbbell_y
roll_forearm
magnet_forearm_z
yaw_forearm
accel_forearm_y
magnet_forearm_y
pitch_arm
accel_arm_z
magnet_arm_y
magnet_arm_z
gyros_arm_x
total_accel_forearm
gyros_forearm_x
yaw_belt
accel_belt_x
magnet_belt_x
roll_arm
yaw_arm
pitch_dumbbell
accel_dumbbell_x
magnet_dumbbell_x
pitch_belt
yaw_dumbbell
accel_dumbbell_z
gyros_dumbbell_x
pitch_forearm
accel_arm_x
magnet_arm_x
magnet_dumbbell_z
accel_forearm_z
gyros_belt_z
gyros_belt_x
magnet_dumbbell_y
gyros_arm_y
accel_belt_z
accel_arm_y
total_accel_arm
accel_forearm_x
magnet_forearm_x
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1

Figure 1:

Dark red indicates a highly negative relationship, whilst dark blue indicates highly positive relationships between variables. We observe from the plot that highly correlated predictors are not prevalent which means that all of variables ***could*** be included in the model. However, it was felt prudent to use the `findCorrelation` function to validate. The `findCorrelation` function given a threshold of pairwise correlations returns the column numbers denoting the predictors that are recommended for deletion.

```
colTitles <- colnames(trainCleanData) # Save Titles
highCorr <- findCorrelation(correlations, cutoff = .75)
length(highCorr)
sort(highCorr)
trainCleanData <- trainCleanData[, -highCorr]
```

After cleaning the data we note that `trainCleanData` has **19622** objects and **32** variables. We have removed the columns and data shown below from the `trainCleanData`, the reader can cross check visually with the corrplots in **figure 1** and **figure 2** to confirm the recommendations.

```
##  [1] "accel_belt_z"      "roll_belt"         "accel_belt_y"
##  [4] "accel_arm_y"       "total_accel_belt"  "accel_dumbbell_z"
##  [7] "accel_belt_x"      "pitch_belt"        "magnet_dumbbell_x"
## [10] "accel_dumbbell_y"  "magnet_dumbbell_y" "accel_arm_x"
## [13] "accel_dumbbell_x"  "accel_arm_z"       "magnet_arm_y"
## [16] "magnet_belt_z"     "accel_forearm_y"   "gyros_forearm_y"
## [19] "gyros_dumbbell_x"  "gyros_dumbbell_z"  "gyros_arm_x"
```

---

3 Analysis / Choice of Algorythm

Our final choice of algorthm was the **Random Forest**, as this indicated the highest **accuracy** against **system time**. The choice of algorythm was based of our analysis below. In essence, our goal was to validate the **time**, verses **accuracy** trade off of a set of machine alogorthms.

3.1 Data Split / Cross Validation

To setup the validaton work, now we have chosen our model we first split the data 60 / 40.

Our goal of cross validation is to define a dataset to "test" the model in the training phase (i.e., the validation dataset), in order to limit problems like overfitting, give an insight on how the model will generalize to an independent dataset etc. One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set). To reduce variability, multiple rounds of cross-validation can be performed using different partitions, and the validation results can be averaged over the rounds. Cross-validation is important in guarding against testing hypotheses suggested by the data (called "Type III errors"), especially where further samples are hazardous, costly or impossible to collect.

```
set.seed(12345)
inTrain <- createDataPartition( trainCleanData$classe, p = 0.70, list = F )
trainSplitData <- trainCleanData[inTrain,]  # 60%
testSplitData <- trainCleanData[-inTrain,]  # 40%
```

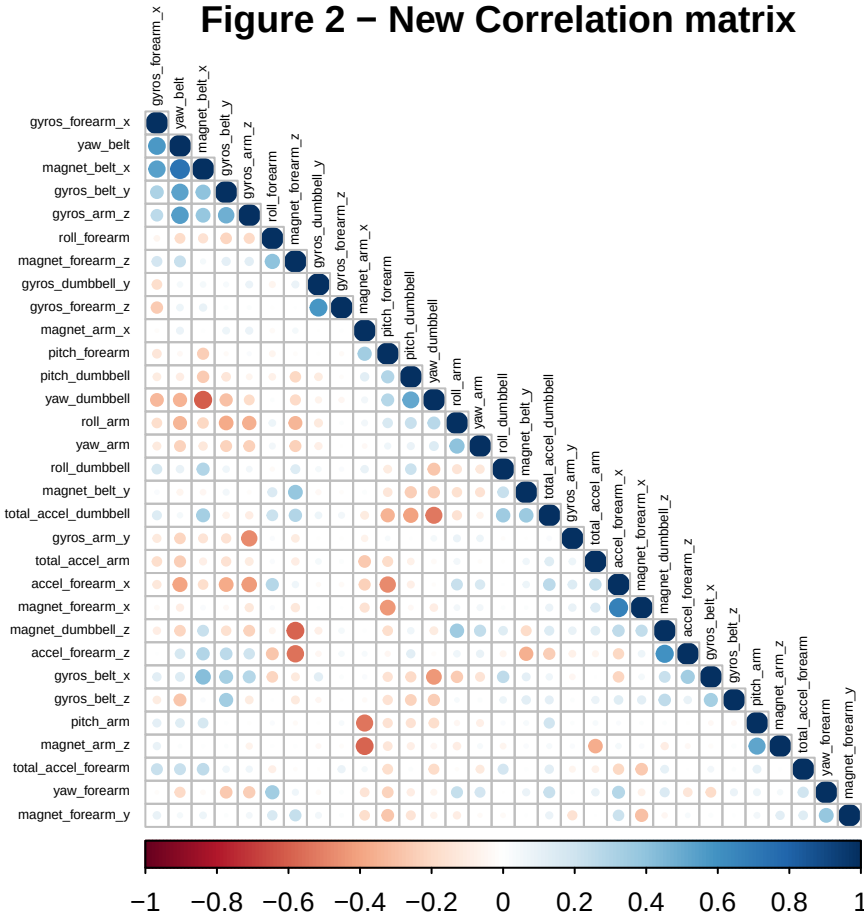
Figure 2 − New Correlation matrix
gyros_forearm_x
yaw_belt
magnet_belt_x
gyros_belt_y
gyros_arm_z
roll_forearm
magnet_forearm_z
gyros_dumbbell_y
gyros_forearm_z
magnet_arm_x
pitch_forearm
pitch_dumbbell
yaw_dumbbell
roll_arm
yaw_arm
roll_dumbbell
magnet_belt_y
total_accel_dumbbell
gyros_arm_y
total_accel_arm
accel_forearm_x
magnet_forearm_x
magnet_dumbbell_z
accel_forearm_z
gyros_belt_x
gyros_belt_z
pitch_arm
magnet_arm_z
total_accel_forearm
yaw_forearm
magnet_forearm_y
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1

Figure 2:

3.2 Model Analysis

Our approach was to run four baseline models and compare system time to accuracy. To reduce overall operation time we harness the `doParallel` package to enable paralellel multi core analysis. We test four scenarios

- Random Forest
- Stochastic Gradient Boosting
- Linear Discriminant Analysis
- Recursive Partitioning

```
#random seed
set.seed(12345)

# Training takes forever on a laptop hence the solution is to try and use as many
# cores as are available! We detect the number of cores, create a cluster and
# register the cores for use in the training process. The train function can take
# some time...
#
registerDoParallel(makeCluster(detectCores()))

timeRF  <- system.time(modelRF <- train(classe ~ ., method = "rf", data = trainSplitData))
timeGBM <- system.time(modelGBM <-train(classe ~ ., method = 'gbm', data = trainSplitData))
```

```
## Loading required package: gbm
## Loading required package: survival
##
## Attaching package: 'survival'
##
## The following object is masked from 'package:caret':
##
##     cluster
##
## Loading required package: splines
## Loaded gbm 2.1.1
```

```
timeLDA <- system.time(modelLDA <-train(classe ~ ., method = 'lda', data = trainSplitData))
```

```
## Loading required package: MASS
##
## Attaching package: 'MASS'
##
## The following object is masked from 'package:dplyr':
##
##     select
```

```
timeDT  <- system.time(modelDT <- train(classe ~ ., method = "rpart", data = trainSplitData))
```

```
## Loading required package: rpart
```

3.3 Performance and Accuracy

Haveing run the models, we are now in a position to compare them.

Model Name	Time	Accuracy
- Random Forest	588.74	1
- Stochastic Gradient Boosting	186.602	0.9640387
- Linear Discriminant Analysis	2.858	0.5853534
- Recursive Partitioning	4.394	0.5276261

The results indicate that the predicted accuracy of the Random Forest algorythm is the best (100% accuracy predicted), hence despite the time required to run the algorythm this is the preferred choice. Note: This is a ***prediction*** based on the `trainSplitData` subset we have tested against.

---

4 Final Model Evaluation and Validation

Now that we have chosen our model (Random Forest), the next step is to tune the model. Our intent is then to evaluate the tuning and identify the variables that are most important to the model. In addition to learning the general patterns in the data, the model has also learned the characteristics of each sample's unique noise. This type of model is said to be over-fit and will usually have poor accuracy when predicting a new sample.

To address this concern, we tuned the final model and performd 10 k-fold repeat operations. Essentially, the data sample is randomly partitioned into k sets of roughly equal size. A model is fit using the all samples except the first subset (called the first fold). The held-out samples are predicted by this model and used to estimate performance measures. The first subset is returned to the training set and procedure repeats with the second subset held out, and so on. The k resampled estimates of performance are summarized (usually with the mean and standard error) and used to understand the relationship between the tuning parameter(s) and model utility.

```
registerDoParallel(makeCluster(detectCores()))

modelControl <- trainControl(method = "cv",
                             number = 10,
                             verboseIter = FALSE,  # Turn on for debugging
                             repeats = 10)

finalModelRF <- train( classe ~ .,
                       method = "rf",
                       data = trainSplitData,
                       trControl = modelControl,
                       allowParallel = TRUE,
                       verbose = FALSE) # Turn on for debugging

finalModelRFAccuracy <- predict(finalModelRF , testSplitData)
finalModelRFConfusionM <- confusionMatrix(finalModelRFAccuracy, testSplitData$classe)
finalModelRFOutofSampleError <- (1 - as.numeric(finalModelRFConfusionM$overall[1]))
```

4.1 Accuracy and Out of Sample Error Rate

Now we have the `finalModelRF`, we can check its accuracy. The model acheives a **99.99%** accuracy level with less than **0.01%** *out of sample error* rate.

4.1.1 Important Variables

At this point we can move to evaluate which variables have the biggest impact on the model. The `caret` package has a unifying function called **`varImp`** that is a wrapper for variable importance functions for the following tree-model objects: ***rpart, classbagg (produced by the ipred package's bagging functions) randomForest, cforest, gbm, and cubist.***

The top five variables are calculated using the `varImp` function and displayed below:

```
##    Overall               var
## 1 764.7225          yaw_belt
## 2 615.4489 magnet_dumbbell_z
## 3 545.2149     pitch_forearm
## 4 543.1500     magnet_belt_y
## 5 492.5342     roll_dumbbell
## 6 488.6683      roll_forearm
```

5 Prediction Result

Having completed the tuning and cross validation of the `finalModelRF` we now move to use it to predict the `testCleanData` results. These results are then written to disk and subsequently uploaded to the coursera.

```
# predict(modelRF, testCleanData)
# predict(modelRF, testCleanData[-highCorr])
#
# predict(finalModelRF, testCleanData)
answers <- predict(finalModelRF, testCleanData[-highCorr])
answers
```

```
##  [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
```

```
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)
        }
}

# Write out the files
pml_write_files(answers)
```

6 Conclusions

The reader can observe that we were able to take the learning data set down from a **160 Columns** to **32 Columns**, and apply the **Random Forest Algorythm** and achieve a projected prediction accuracy of **99.99%** with less than **0.01%** out of sample error rate.

Random Forests in summary are easy to learn and use for both professionals and lay people - with little research and programming required and may be used by people without a strong statistical background. Simply put, you can safely make more accurate predictions without most basic mistakes common to other methods.

- Accuracy
- Runs efficiently on large data bases
- Handles thousands of input variables without variable deletion
- Gives estimates of what variables are important in the classification
- Generates an internal unbiased estimate of the generalization error as the forest building progresses
- Provides effective methods for estimating missing data
- Maintains accuracy when a large proportion of the data are missing
- Provides methods for balancing error in class population unbalanced data sets
- Generated forests can be saved for future use on other data
- Prototypes are computed that give information about the relation between the variables and the classification.
- Computes proximities between pairs of cases that can be used in clustering, locating outliers, or (by scaling) give interesting views of the data
- Capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection
- Offers an experimental method for detecting variable interactions

7 Appendix A

7.1 References

- *Data Classification of Body Postures and Movements.*
  - By: Ugulino, W.; Cardador, D.; Vega, K.; Velloso, E.; Milidiu, R.; Fuks, H. Wearable Computing: Accelerometers'
  - Proceedings of 21st Brazilian Symposium on Artificial Intelligence. Advances in Artificial Intelligence - SBIA 2012. In: Lecture Notes in Computer Science. , pp. 52-61. Curitiba, PR: Springer Berlin / Heidelberg, 2012. ISBN 978-3-642-34458-9. DOI: 10.1007/978-3-642-34459-6_6.
- *Applied Predictive Modeling*
  - By: Max Kuln, Kjell Johnson, Publisher: Springer; Pub Date: May 17, 2013, eISBN: 978-1-461-46848-6
- *The R Book*
  - By: Michael J. Crawley, Publisher: John Wiley & Sons Pub. Date: December 26, 2012, eISBN: 978-1-118-44896-0
- *An Introduction to Statistical Learning*
  - By: Gareth James (Author), Daniela Witten (Author), Trevor Hastie (Author), Robert Tibshirani (Author)
- *The Elements of Statistical Learning*
  - By: Trevor Hastie (Author), Robert Tibshirani (Author), Jerome Friedman (Author)
- *R in Action*
  - By: Robert Kabacoff Publisher: Manning Publications Pub. Date: August 24, 2011, ISBN-10: 1-935182-39-0
- *Mathematical Statistics with Resampling and R*
  - By: Laura Chihara; Tim Hesterberg Publisher: John Wiley & Sons Pub. Date: September 6, 2011 Print ISBN: 978-1-11-02985-5
- *OpenIntro Statistics - 2nd Editions*
  - By: by David M Diez (Author), Christopher D Barr (Author), Mine Çetinkaya-Rundel (Author)
- *Think Stats, 2nd Edition*
  - By: Allen B. Downey Publisher: O'Reilly Media, Inc. Pub. Date: October 28, 2014 Print ISBN-13: 978-1-4919-0733-7
- *Caret Training*
  - http://topepo.github.io/caret/training.html

7.2 Environment

```
##                _
## platform       x86_64-apple-darwin13.4.0
## arch           x86_64
## os             darwin13.4.0
## system         x86_64, darwin13.4.0
## status
## major          3
## minor          2.1
## year           2015
## month          06
## day            18
## svn rev        68531
## language       R
## version.string R version 3.2.1 (2015-06-18)
## nickname       World-Famous Astronaut
```