Machine Learning Project 1

T. G.

June 20, 2017

```
knitr::opts_chunk$set(echo = TRUE, cache= TRUE)

require(caret)
```

```
## Loading required package: caret
```

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

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

```
require(rpart)
```

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

```
require(rattle)
```

```
## Loading required package: rattle
```

```
## Rattle: A free graphical interface for data mining with R.
## Version 4.1.0 Copyright (c) 2006-2015 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
require(kernlab)
```

```
## Loading required package: kernlab
```

```
##
## Attaching package: 'kernlab'
```

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

Introduction and Background

Introduction

From the project introduction:

> 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: http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har) (see the section on the Weight Lifting Exercise Dataset).

> The goal of your project is to predict the manner in which they did the exercise. This is the "classe" variable in the training set. You may use any of the other variables to predict with. You should create a report describing how you built your model, how you used cross validation, what you think the expected out of sample error is, and why you made the choices you did. You will also use your prediction model to predict 20 different test cases.

Data

> The training data for this project are available here: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv)

> The test data are available here: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv)

The data come from this publication:

Ugulino, W.; Cardador, D.; Vega, K.; Velloso, E.; Milidiu, R.; Fuks, H. Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements. 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.

Goal

After performing initial data cleaning, three different classification models will be trained on a subset of the data from pml-training.csv and tested on the rest of the training data from that dataset. The goal is to maximize accuracy without a need for interpretation of the results, as that was not a specificed requirement.

Data Import and Cleaning

Import

Import the data using the basic R command, while also doing preliminary cleaning by labeling specific values as N/A.

```
rawTrainingData <- read.csv("Data/pml-training.csv", na.strings = c("NA", "",
"#DIV/0!"))
rawTestData <- read.csv("Data/pml-testing.csv", na.strings = c("NA", "", "#DIV/
0!"))
```

Data Cleaning

Any data cleaning performed mut be performed identially to both the testing and training data.

Remove PII & Non-contributing data

The first seven columns are row number, username, datetime, and diagnostic data. Removing these will not impact predictive power, increases data security by removing what may be considered PII, and lessens the amount of computations the system will have to perform.

```
noPIITraining <- rawTrainingData[-c(1:7)]
noPIITest <- rawTestData[-c(1:7)]
```

Remove columns where all data is missing

With any sort of collected data, there is the possibility of data corruption, mis-entry, or other flaws in the data. When examining the training data, there is a non-negligible number of columns with missing data. These must be removed in order to properly train the machine models.

```
noBadCTraining <- noPIITraining[,sapply(noPIITraining, function(x)any(is.na
(x))) == FALSE]
noBadCTesting <- noPIITest[,sapply(noPIITraining, function(x)any(is.na(x))) ==
FALSE]
```

Data Splitting

In order to properly create models based on machine learning algorithms, one must split the input data into a “training” set and a “testing” set. This testing set is distinct from the imported testing set. A random sample of 70% is taken from the noBadCTraining set and made into a “finalTraining” set. The other 30% is assigned to a “finalTesting” set.

R generated the random number 28563 that will be used to seed the rest of the random number generators to ensure repeatable results.

```
set.seed(28563)

trainPart <- createDataPartition(noBadCTraining$classe, p = 0.7, list = FALSE)

finalTraining <- noBadCTraining[trainPart,]
finalTesting <- noBadCTraining[-trainPart,]
```

Model Creation

Models were selected from the broader library of classification algorithms that were covered within the course. Random Trees was chosen specifically so that the researcher could investigate SVM in different basis. K-fold cross-validation is left to the parameter default of 10. Model 3 has parallel processing enabled for better time efficiency, which may cause issues with reproducibility due to the interaction of setting the seed with parallelization.

1. CART Model - RPart in Caret

First examined is the typical classification tree implemented through RPart in caret.

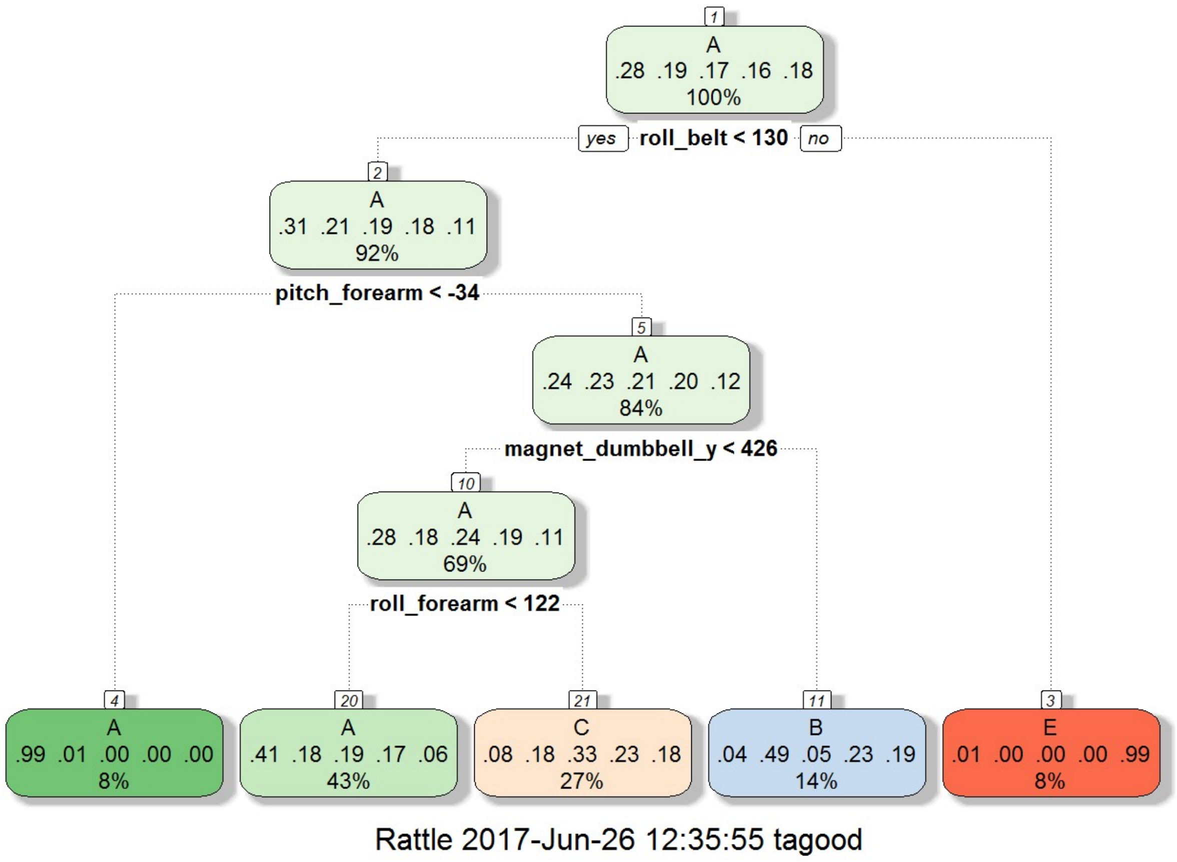
```
treeModel <- train(classe ~., data=finalTraining, method = "rpart")

treePredict <- predict(treeModel, finalTesting)
confTree <- confusionMatrix(finalTesting$classe, treePredict)
print(confTree)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1495   46  129    0    4
##          B  459  408  272    0    0
##          C  455   46  525    0    0
##          D  413  183  368    0    0
##          E  145  162  279    0  496
## 
## Overall Statistics
## 
##                Accuracy : 0.4969
##                  95% CI : (0.484, 0.5097)
##     No Information Rate : 0.5042
##     P-Value [Acc > NIR] : 0.8716
## 
##                   Kappa : 0.3437
##  Mcnemar's Test P-Value : NA
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.5039  0.48284  0.33376       NA  0.99200
## Specificity            0.9387  0.85496  0.88381   0.8362  0.89118
## Pos Pred Value         0.8931  0.35821  0.51170       NA  0.45841
## Neg Pred Value         0.6504  0.90792  0.78432       NA  0.99917
## Prevalence             0.5042  0.14359  0.26729   0.0000  0.08496
## Detection Rate         0.2540  0.06933  0.08921   0.0000  0.08428
## Detection Prevalence   0.2845  0.19354  0.17434   0.1638  0.18386
## Balanced Accuracy      0.7213  0.66890  0.60878       NA  0.94159
```

Accuracy is `as.numeric(confTree$overall[1]`, which results in a `1 - as.numeric(confTree$overall[1])` out-of-sample error. This is not ideal. This does make a pretty picture, though.

```
fancyRpartPlot(treeModel$finalModel)
```



2. GBM Model - gbm in Caret

A gradient boosting machine model is explored as a second option. This is implemented through the gbm package.

```
gbmModel <- train(classe ~., data=finalTraining, method = "gbm", verbose = FALS
E)
```

```
## Loading required package: gbm
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survival'
```

```
## The following object is masked from 'package:caret':
##
##     cluster
## Loading required package: splines
```

```
## Loading required package: parallel
```

```
## Loaded gbm 2.1.3
```

```
## Loading required package: plyr
```

```
print(gbmModel)
```

```
## Stochastic Gradient Boosting
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 13737, 13737, 13737, 13737, 13737, 13737, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7505130  0.6837604
##   1                  100      0.8173600  0.7688937
##   1                  150      0.8503038  0.8106304
##   2                   50      0.8510936  0.8113625
##   2                  100      0.9039999  0.8785344
##   2                  150      0.9288615  0.9099927
##   3                   50      0.8930219  0.8646025
##   3                  100      0.9386818  0.9224222
##   3                  150      0.9574166  0.9461288
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were n.trees = 150,
##  interaction.depth = 3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
gbmPredict <- predict(gbmModel, finalTesting)
confGBM <- confusionMatrix(finalTesting$classe, gbmPredict)
print(confGBM)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1655   10    5    1    3
##          B   35 1079   23    0    2
##          C    0   36  975   14    1
##          D    1    4   32  919    8
##          E    2   12   11   19 1038
## 
## Overall Statistics
## 
##                Accuracy : 0.9628          
##                  95% CI : (0.9576, 0.9675)
##     No Information Rate : 0.2877          
##     P-Value [Acc > NIR] : < 2.2e-16       
## 
##                   Kappa : 0.9529          
##  Mcnemar's Test P-Value : 7.592e-08       
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9776   0.9457   0.9321   0.9643   0.9867
## Specificity            0.9955   0.9874   0.9895   0.9909   0.9909
## Pos Pred Value         0.9886   0.9473   0.9503   0.9533   0.9593
## Neg Pred Value         0.9910   0.9869   0.9854   0.9931   0.9971
## Prevalence             0.2877   0.1939   0.1777   0.1619   0.1788
## Detection Rate         0.2812   0.1833   0.1657   0.1562   0.1764
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9865   0.9665   0.9608   0.9776   0.9888
```

Accuracy is `as.numeric(confGBM$overall[1]` , which results in a `1 - as.numeric(confGBM$overall[1])` out-of-sample error. While this is high, a third model could achieve higher accuracy.

3. Support Vector Machine Model with Polynomial Basis - kernlab in caret

Support Vector Machine Model using a polynomial basis is explored as a third model. A linear basis model was tested but performed worse than this model, and is not suitable for inclusion. The known drawback to this model is that the computational power required to run this model is significant. Runtime was around 8 hours. This is due to the fact that the polynomial basis is computationally more demanding.

```
polysvmModel <-train(classe ~., data=finalTraining, method = "svmPoly", allowPa
rallel = TRUE)
print(polysvmModel)
```

```
## Support Vector Machines with Polynomial Kernel
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 13737, 13737, 13737, 13737, 13737, 13737, ...
## Resampling results across tuning parameters:
##
##   degree  scale  C     Accuracy   Kappa
##   1       0.001  0.25  0.5024424  0.3514119
##   1       0.001  0.50  0.5717586  0.4509959
##   1       0.001  1.00  0.6038993  0.4946571
##   1       0.010  0.25  0.6484232  0.5530538
##   1       0.010  0.50  0.6775955  0.5901440
##   1       0.010  1.00  0.7033099  0.6228248
##   1       0.100  0.25  0.7309556  0.6580470
##   1       0.100  0.50  0.7451197  0.6761203
##   1       0.100  1.00  0.7565457  0.6906504
##   2       0.001  0.25  0.5782652  0.4594902
##   2       0.001  0.50  0.6154073  0.5094919
##   2       0.001  1.00  0.6596012  0.5671998
##   2       0.010  0.25  0.8307352  0.7855177
##   2       0.010  0.50  0.8664643  0.8307747
##   2       0.010  1.00  0.8957586  0.8679172
##   2       0.100  0.25  0.9668981  0.9581076
##   2       0.100  0.50  0.9735445  0.9665298
##   2       0.100  1.00  0.9776627  0.9717452
##   3       0.001  0.25  0.6095114  0.5011878
##   3       0.001  0.50  0.6575016  0.5641035
##   3       0.001  1.00  0.7054719  0.6258347
##   3       0.010  0.25  0.8906797  0.8614649
##   3       0.010  0.50  0.9168590  0.8946698
##   3       0.010  1.00  0.9394154  0.9232759
##   3       0.100  0.25  0.9856754  0.9818826
##   3       0.100  0.50  0.9862444  0.9826029
##   3       0.100  1.00  0.9859441  0.9822227
##
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were degree = 3, scale = 0.1 and C
##  = 0.5.
```

```
svmPredict <- predict(polysvmModel, finalTesting)
confSVM <- confusionMatrix(finalTesting$classe, svmPredict)
print(confSVM)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1672    2    0    0    0
##          B    6 1130    3    0    0
##          C    0    5 1016    5    0
##          D    0    0   13  947    4
##          E    0    0    1    0 1081
##
## Overall Statistics
##
##                Accuracy : 0.9934
##                  95% CI : (0.991, 0.9953)
##     No Information Rate : 0.2851
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9916
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9964   0.9938   0.9835   0.9947   0.9963
## Specificity            0.9995   0.9981   0.9979   0.9966   0.9998
## Pos Pred Value         0.9988   0.9921   0.9903   0.9824   0.9991
## Neg Pred Value         0.9986   0.9985   0.9965   0.9990   0.9992
## Prevalence             0.2851   0.1932   0.1755   0.1618   0.1844
## Detection Rate         0.2841   0.1920   0.1726   0.1609   0.1837
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9980   0.9960   0.9907   0.9957   0.9981
```

Accuracy is `as.numeric(confSVM$overall[1]`, which results in a `1 - as.numeric(confSVM$overall[1])` out-of-sample error. This is the most accurate model, but it is unclear whether the additional computational time required justifies the use of this model.

Prediction

```
answers <- predict(polysvmModel, noBadCTesting)
print(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
```