Practical Machine Learning Project

Mike Delevan

3/19/2021

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.6.3
```

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.6.3
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.6.3
```

```
library(corrplot)
```

```
## Warning: package 'corrplot' was built under R version 3.6.3
```

```
set.seed(752)
```

Overview

From the Overview section of the assignment:

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://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

Loading Data

We can start this project by first pulling in the data from their respective URL's. This data has already been split into a training and a testings sets for our convenience.

```
trainURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
testURL  <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

if(!file.exists("pml-training.csv")){
  download.file(trainURL, "pml-training.csv")
}

if(!file.exists("pml-testing.csv")){
  download.file(testURL, "pml-testing.csv")
}

train <- read.csv("pml-training.csv")
test <- read.csv("pml-testing.csv")
```

Now that we have our dataset let's get the dimensions and take a quick look at the contents

```
dim(train)
```

```
## [1] 19622   160
```

```
dim(test)
```

```
## [1]  20 160
```

So our training set and test set both have 160 features/variables. Many of these variables are probably not important for our prediction, so many will be removed before we begin to train and test our models. Also, the training set is much larger than the test set which is expected for creating a machine learning model.

Here's the a quick look at some rows in the data as well:

```
train[158:160,158:160]
```

```
##     magnet_forearm_y magnet_forearm_z classe
## 158              650              475      A
## 159              651              472      A
## 160              655              464      A
```

Cleaning Data / Feature Engineering

We could see from our sneak peek at the data that there are a lot of NA's even in the first 3 rows. Let's try and remove those variables that contain a majority of NA's. We will also remove the same columns from the test set as well. Machine learning models expect the same dataset dimensions in the test set as the training set it was trained on. Having different dimensions will throw an error.

```
trainPartition  <- createDataPartition(train$classe, p=0.8, list=FALSE)
trainSubset <- train[trainPartition,]
validSubset <- train[-trainPartition,]


AllNATrain    <- sapply(train, function(x) mean(is.na(x))) > 0.95
```

```
trainSubset <- trainSubset[, AllNATrain==FALSE]
validSubset <- validSubset[, AllNATrain==FALSE]
test <- test[, AllNATrain==FALSE]
```

We can also go ahead and remove the first 7 columns as they contain metadata that could potentially skew our model accidentally.

```
trainSubset <- trainSubset[, -(1:7)]
validSubset <- validSubset[, -(1:7)]
test <- test[, -(1:7)]
```

Let's also remove the variables that have a non-zero variance

```
nzv <- nearZeroVar(trainSubset)
trainSubset <- trainSubset[,-nzv]
validSubset <- validSubset[,-nzv]
test        <- test[,-nzv]

dim(trainSubset)
```

```
## [1] 15699    53
```

```
dim(validSubset)
```

```
## [1] 3923   53
```

```
dim(test)
```

```
## [1] 20 53
```

So after removing a lot of the unneccessary variables, we end up with only 53 variables out of the original 160. This should drastically improve model performance.

Modeling

Random Forest

We will start our modeling off with a random forest. This is a collection of decisions trees and should provide an improvement over a single decision tree.

```
tC <- trainControl(method="repeatedcv",number=3)
frFit <- train(classe~., data=trainSubset, method="rf", trControl = tC, tuneLength = 5)
```

```
print(frFit)
```

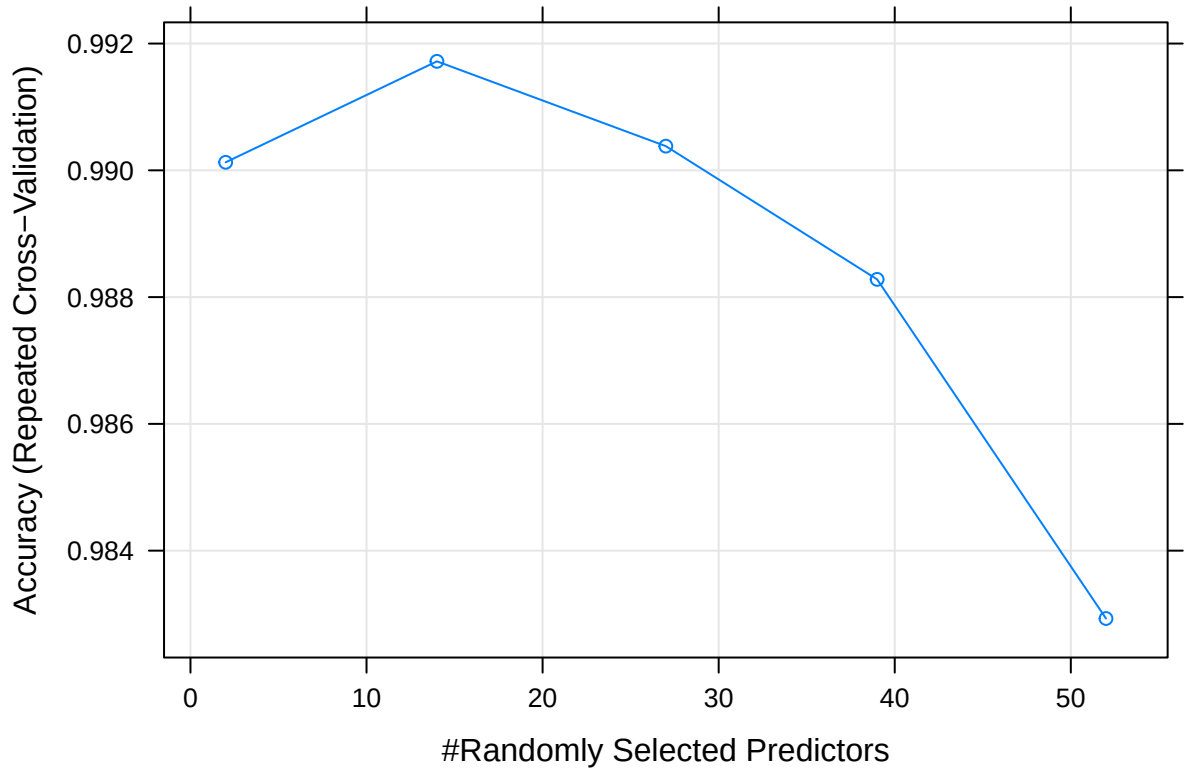
```
## Random Forest
##
## 15699 samples
```

```
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold, repeated 1 times)
## Summary of sample sizes: 10467, 10466, 10465
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9901265  0.9875097
##   14    0.9917191  0.9895250
##   27    0.9903815  0.9878331
##   39    0.9882796  0.9851735
##   52    0.9829292  0.9784028
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 14.
```

Using 3-fold cross-validation we find that the highest estimate obtained for the out-of-bag accuracy is **.9917191**. The model used to get this accuracy only used 14 of the 52 variables in our dataset. We would only see marginal improvements using more sophisticated models or even an ensemble. So we will leave it as it is and use the most accurate model to predict going forward.

Let's also plot our model to get a good idea of how it performed as more variables were introduced into the dataset:

```
plot(frFit)
```



We see from this graph that as more predictors were added into the model, the worse it performed. Imagine the performance dips we would've seen had we included all 160 variables from the original dataset!

Let's also calculate the accuracy of our model using the validation set we created in the beginning:

```
pred <- predict(frFit, validSubset)
confMatrix<- confusionMatrix(pred, factor(validSubset$classe))
confMatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1116    4    0    0    0
##          B    0  754    2    0    0
##          C    0    1  682   10    0
##          D    0    0    0  633    2
##          E    0    0    0    0  719
##
## Overall Statistics
##
##                Accuracy : 0.9952
##                  95% CI : (0.9924, 0.9971)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9939
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9934   0.9971   0.9844   0.9972
## Specificity            0.9986   0.9994   0.9966   0.9994   1.0000
## Pos Pred Value         0.9964   0.9974   0.9841   0.9969   1.0000
## Neg Pred Value         1.0000   0.9984   0.9994   0.9970   0.9994
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1922   0.1738   0.1614   0.1833
## Detection Prevalence   0.2855   0.1927   0.1767   0.1619   0.1833
## Balanced Accuracy      0.9993   0.9964   0.9968   0.9919   0.9986
```

From this result we see that the model predicts with high accuracy, even on data it has never even seen before. The accuracy acheived on the validation set is **.9952** which is higher than the accuracy predicted by our cross-validation.

Testing

Now that we have our trained model, we can predict on the actual test set and get this project done:

```
predictions <- predict(frFit, test)
predictions
```

```
##  [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
```