Machine Learning Assignment

Brian Thomas

February 12, 2016

Contents

Thanks!

First, let us thank the contributors of this dataset for their conscientious work in compiling the data, and their generosity in making it available to the public: Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. Qualitative Activity Recognition of Weight Lifting Exercises. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human '13) . Stuttgart, Germany: ACM SIGCHI, 2013.

Now, let's get this analysis going…

Libraries

Load some useful libraries

```
library(caret)
library(dplyr)
library(data.table)
library(ggplot2)
library(rpart)
library(doParallel)
library(knitr)
```

Get the Data

Open a connection to the data, download the data, and close the connection.

```
urltr<-"https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
contr<-url(urltr, open = "rb" )
trdat<-fread(urltr, na.strings="NA")
trdat<-tbl_df(trdat)
close(contr)
set.seed(1991)


urlte<-"https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
conte<-url(urlte, open="rb")
tedat<-fread(urlte, na.strings="NA")
tedat<-tbl_df(tedat)
close(conte)
rm(contr,conte, urlte, urltr)
```

Create a data partition to separate training and cross validation data sets. Once these are created, we'll get a feel for the dimensions.

```r
inTrain<-createDataPartition(y=trdat$classe, p=.70, list=FALSE)
trdat1<-trdat[inTrain,]
cvdat<-trdat[-inTrain,]
rm(inTrain)
dim(trdat1)
```

[1] 13737 160

```r
dim(cvdat)
```

[1] 5885 160

```r
dim(tedat)
```

[1] 20 160

Clean the data

Let's look at the class of each variable

```r
sapply(trdat1, class)
```

```
                   V1             user_name  raw_timestamp_part_1 
          "character"           "character"             "integer" 
 raw_timestamp_part_2        cvtd_timestamp            new_window 
            "integer"           "character"           "character" 
           num_window             roll_belt            pitch_belt 
            "integer"             "numeric"             "numeric" 
             yaw_belt      total_accel_belt    kurtosis_roll_belt 
            "numeric"             "integer"           "character" 
  kurtosis_picth_belt     kurtosis_yaw_belt    skewness_roll_belt 
          "character"           "character"           "character" 
skewness_roll_belt.1     skewness_yaw_belt         max_roll_belt 
          "character"           "character"             "numeric" 
       max_picth_belt          max_yaw_belt         min_roll_belt 
            "integer"           "character"             "numeric" 
       min_pitch_belt          min_yaw_belt   amplitude_roll_belt 
            "integer"           "character"             "numeric" 
 amplitude_pitch_belt    amplitude_yaw_belt  var_total_accel_belt 
            "integer"           "character"             "numeric" 
        avg_roll_belt      stddev_roll_belt         var_roll_belt 
            "numeric"             "numeric"             "numeric" 
       avg_pitch_belt     stddev_pitch_belt        var_pitch_belt 
            "numeric"             "numeric"             "numeric" 
         avg_yaw_belt       stddev_yaw_belt          var_yaw_belt 
            "numeric"             "numeric"             "numeric" 
         gyros_belt_x          gyros_belt_y          gyros_belt_z 
            "numeric"             "numeric"             "numeric" 
         accel_belt_x          accel_belt_y          accel_belt_z 
            "integer"             "integer"             "integer" 
        magnet_belt_x         magnet_belt_y         magnet_belt_z 
```

```
            "integer"                "integer"                "integer"
             roll_arm                pitch_arm                  yaw_arm
            "numeric"                "numeric"                "numeric"
      total_accel_arm            var_accel_arm              avg_roll_arm
            "integer"                "numeric"                "numeric"
      stddev_roll_arm             var_roll_arm             avg_pitch_arm
            "numeric"                "numeric"                "numeric"
     stddev_pitch_arm            var_pitch_arm               avg_yaw_arm
            "numeric"                "numeric"                "numeric"
       stddev_yaw_arm              var_yaw_arm               gyros_arm_x
            "numeric"                "numeric"                "numeric"
          gyros_arm_y              gyros_arm_z               accel_arm_x
            "numeric"                "numeric"                "integer"
          accel_arm_y              accel_arm_z              magnet_arm_x
            "integer"                "integer"                "integer"
         magnet_arm_y             magnet_arm_z         kurtosis_roll_arm
            "integer"                "integer"              "character"
   kurtosis_picth_arm        kurtosis_yaw_arm          skewness_roll_arm
          "character"              "character"              "character"
   skewness_pitch_arm        skewness_yaw_arm               max_roll_arm
          "character"              "character"                "numeric"
        max_picth_arm              max_yaw_arm              min_roll_arm
            "numeric"                "integer"                "numeric"
        min_pitch_arm              min_yaw_arm        amplitude_roll_arm
            "numeric"                "integer"                "numeric"
  amplitude_pitch_arm       amplitude_yaw_arm              roll_dumbbell
            "numeric"                "integer"                "numeric"
       pitch_dumbbell             yaw_dumbbell    kurtosis_roll_dumbbell
            "numeric"                "numeric"              "character"
```

kurtosis_picth_dumbbell kurtosis_yaw_dumbbell skewness_roll_dumbbell “character” “character” “character” skewness_pitch_dumbbell skewness_yaw_dumbbell max_roll_dumbbell “character” “character” “numeric” max_picth_dumbbell max_yaw_dumbbell min_roll_dumbbell “numeric” “character” “numeric” min_pitch_dumbbell min_yaw_dumbbell amplitude_roll_dumbbell “numeric” “character” “numeric” amplitude_pitch_dumbbell amplitude_yaw_dumbbell total_accel_dumbbell “numeric” “character” “integer” var_accel_dumbbell avg_roll_dumbbell stddev_roll_dumbbell “numeric” “numeric” “numeric” var_roll_dumbbell avg_pitch_dumbbell stddev_pitch_dumbbell “numeric” “numeric” “numeric” var_pitch_dumbbell avg_yaw_dumbbell stddev_yaw_dumbbell “numeric” “numeric” “numeric” var_yaw_dumbbell gyros_dumbbell_x gyros_dumbbell_y “numeric” “numeric” “numeric” gyros_dumbbell_z accel_dumbbell_x accel_dumbbell_y “numeric” “integer” “integer” accel_dumbbell_z magnet_dumbbell_x magnet_dumbbell_y “integer” “integer” “integer” magnet_dumbbell_z roll_forearm pitch_forearm “numeric” “numeric” “numeric” yaw_forearm kurtosis_roll_forearm kurtosis_picth_forearm “numeric” “character” “character” kurtosis_yaw_forearm skewness_roll_forearm skewness_pitch_forearm “character” “character” “character” skewness_yaw_forearm max_roll_forearm max_picth_forearm “character” “numeric” “numeric” max_yaw_forearm min_roll_forearm min_pitch_forearm “character” “numeric” “numeric” min_yaw_forearm amplitude_roll_forearm amplitude_pitch_forearm “character” “numeric” “numeric” amplitude_yaw_forearm total_accel_forearm var_accel_forearm “character” “integer” “numeric” avg_roll_forearm stddev_roll_forearm var_roll_forearm “numeric” “numeric” “numeric” avg_pitch_forearm stddev_pitch_forearm var_pitch_forearm “numeric” “numeric” “numeric” avg_yaw_forearm stddev_yaw_forearm var_yaw_forearm “numeric” “numeric” “numeric” gyros_forearm_x gyros_forearm_y gyros_forearm_z “numeric” “numeric” “numeric” accel_forearm_x accel_forearm_y accel_forearm_z “integer” “integer” “integer” magnet_forearm_x magnet_forearm_y magnet_forearm_z “integer” “numeric” “numeric” classe “character”

We find numerous "Character" classes that should be numeric for our analysis. Let's change these

```
trdat1<-cbind(trdat1[,1:6], sapply(trdat1[,7:159], as.numeric), trdat1[,160])
```

There are NA values dispersed throughout the dataset. Let's change these to zero

```
trdat1[is.na(trdat1)]<-0
```

There are also #DIV/0! values. Let's try to change these to zero as well.

```
trdat1[which(trdat1=="#DIV/0!")]<-0
```

See how many zeroes exist in each variable after transformation. If the number is a strong majority, the variable may not be very useful. We will return a vector of column indices.

```
zero_df<-data.frame()
for(i in 1:160){
  zero_df[i,1]<-i
  zero_df[i,2]<-sum(trdat1[,i]==0)
}
zero_df
```

V1 V2

1 1 0 2 2 0 3 3 0 4 4 0 5 5 0 6 6 0 7 7 0 8 8 3 9 9 1 10 10 1 11 11 2 12 12 13459 13 13 13472 14 14 13737 15 15 13462 16 16 13473 17 17 13737 18 18 13452 19 19 13452 20 20 13463 21 21 13452 22 22 13453 23 23 13463 24 24 13461 25 25 13472 26 26 13737 27 27 13491 28 28 13453 29 29 13479 30 30 13536 31 31 13452 32 32 13454 33 33 13536 34 34 13452 35 35 13481 36 36 13497 37 37 736 38 38 3305 39 39 1217 40 40 20 41 41 380 42 42 14 43 43 243 44 44 0 45 45 0 46 46 2364 47 47 2364 48 48 2364 49 49 0 50 50 13460 51 51 13504 52 52 13504 53 53 13504 54 54 13504 55 55 13504 56 56 13504 57 57 13504 58 58 13505 59 59 13505 60 60 346 61 61 364 62 62 373 63 63 39 64 64 45 65 65 40 66 66 6 67 67 13 68 68 4 69 69 13504 70 70 13505 71 71 13460 72 72 13504 73 73 13506 74 74 13460 75 75 13504 76 76 13504 77 77 13452 78 78 13504 79 79 13504 80 80 13452 81 81 13504 82 82 13505 83 83 13460 84 84 96 85 85 234 86 86 75 87 87 13456 88 88 13453 89 89 13737 90 90 13456 91 91 13453 92 92 13737 93 93 13453 94 94 13452 95 95 13465 96 96 13456 97 97 13456 98 98 13465 99 99 13465 100 100 13465 101 101 13737 102 102 20 103 103 13466 104 104 13452 105 105 13465 106 106 13465 107 107 13452 108 108 13465 109 109 13465 110 110 13452 111 111 13465 112 112 13465 113 113 423 114 114 376 115 115 421 116 116 234 117 117 93 118 118 72 119 119 2 120 120 0 121 121 62 122 122 2716 123 123 2716 124 124 2715 125 125 13517 126 126 13518 127 127 13737 128 128 13517 129 129 13519 130 130 13737 131 131 13517 132 132 13517 133 133 13520 134 134 13517 135 135 13517 136 136 13520 137 137 13517 138 138 13518 139 139 13737 140 140 5 141 141 13457 142 142 13517 143 143 13519 144 144 13519 145 145 13517 146 146 13517 147 147 13517 148 148 13517 149 149 13518 150 150 13518 151 151 373 152 152 264 153 153 294 154 154 22 155 155 21 156 156 53 157 157 7 158 158 2 159 159 7 160 160 0

There are a lot of variables containing very little data.
Let's isolate those that are mostly zero and remove them from each dataset Also, it's time to remove column "V1" since it is going to introduce problems later.

```
zero_cols<-zero_df[which(zero_df[,2]>5000),1]; zero_cols;
```

[1] 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 [18] 29 30 31 32 33 34 35 36 50 51 52 53 54 55 56 57 58 [35] 59 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 87 [52] 88 89 90 91 92 93 94 95 96 97 98 99 100 101 103 104 105 [69] 106 107 108 109 110 111 112 125 126 127 128 129 130 131 132 133 134 [86] 135 136 137 138 139 141 142 143 144 145 146 147 148 149 150

```r
trdat2<-trdat1[,-c(1,zero_cols)]; dim(trdat2)
```

[1] 13737 59

```r
cvdat2<-cvdat[,-c(1,zero_cols)]; dim(cvdat2)
```

[1] 5885 59

```r
tedat2<-tedat[,-c(1,zero_cols)]; dim(tedat2)
```

[1] 20 59

Choose a Model

My initial plan is to run an "rf" model. I have set up the following trainControl parameters.

```r
tc<-trainControl(method = "oob",
                 number = 10,
                 repeats = 10,
                 p = 0.75,
                 search = "grid",
                 initialWindow = NULL,
                 horizon = 1,
                 fixedWindow = TRUE,
                 verboseIter = FALSE,
                 returnData = TRUE,
                 returnResamp = "final",
                 savePredictions = FALSE,
                 classProbs = TRUE,
                 summaryFunction = defaultSummary,
                 selectionFunction = "best",
                 preProcOptions = list(thresh = 0.95, ICAcomp = 3, k = 5),
                 sampling = NULL,
                 index = NULL,
                 indexOut = NULL,
                 timingSamps = 0,
                 predictionBounds = rep(FALSE, 2),
                 #seeds = seeds,
                 adaptive = list(min = 5, alpha = 0.05,
                                 method = "gls", complete = TRUE),
                 trim = FALSE,
                 allowParallel = TRUE)
```

After numerous failed attempts at training models due to memory limits, I decided to randomly split my data into several models.

```r
dat_size<-dim(trdat2)[1]/3
dat_size2<-dat_size+1
dat_size3<-dat_size*2
dat_size4<-dat_size3+1
dat_size5<-dat_size*3
samp_vect<-sample(dim(trdat2)[1],dim(trdat2)[1],replace = FALSE)
trdat2_1<-trdat2[samp_vect[1:dat_size],]; dim(trdat2_1)
```

[1] 4579 59

```
trdat2_2<-trdat2[samp_vect[(dat_size2):(dat_size3)],]; dim(trdat2_2)
```

[1] 4579 59

```
trdat2_3<-trdat2[samp_vect[(dat_size4):(dat_size5)],]; dim(trdat2_3)
```

[1] 4579 59

```
tgrid<- expand.grid(mtry = 50)
memory.limit(size=10000000000000)
```

[1] 1e+13

```
registerDoParallel(4)
memory.limit(size=10000000000000)
```

[1] 1e+13

```
t1<-Sys.time()
modFit_1<-train(classe~.,
              trdat2_1,
              method = "rf",
              metric = "Accuracy",
              maximize = TRUE,
              trControl = tc,
              tuneGrid = tgrid,
              tuneLength=50,
              prox=TRUE
)
t2<-Sys.time()
Mod1_time<-t2-t1

registerDoParallel(4)
memory.limit(size=10000000000000)
```

[1] 1e+13

```
t1<-Sys.time()
modFit_2<-train(classe~.,
              trdat2_2,
              method = "rf",
              metric = "Accuracy",
              maximize = TRUE,
              trControl = tc,
              tuneGrid = tgrid,
              tuneLength=50,
              prox=TRUE
)
t2<-Sys.time()
```

```r
Mod2_time<-t2-t1

registerDoParallel(4)
memory.limit(size=10000000000000)
```

[1] 1e+13

```r
t1<-Sys.time()
modFit_3<-train(classe~.,
              trdat2_3,
              method = "rf",
              metric = "Accuracy",
              maximize = TRUE,
              trControl = tc,
              tuneGrid = tgrid,
              tuneLength=50,
              prox=TRUE
)
t2<-Sys.time()
Mod3_time<-t2-t1
```

I decided to capture the time it took to run each model for general interest.

```r
Mod1_time
```

Time difference of 1.516933 mins

```r
Mod2_time
```

Time difference of 1.473889 mins

```r
Mod3_time
```

Time difference of 1.504183 mins

Results

Here are the model results for the training and cross-validation sets.

```r
modFit_1
```

Random Forest

4579 samples 58 predictor 5 classes: 'A', 'B', 'C', 'D', 'E'

No pre-processing Resampling results

Accuracy Kappa
0.9934484 0.9917259

Tuning parameter 'mtry' was held constant at a value of 50

```
modFit_2
```

Random Forest

4579 samples 58 predictor 5 classes: 'A', 'B', 'C', 'D', 'E'

No pre-processing Resampling results

Accuracy Kappa
0.9954138 0.9941847

Tuning parameter 'mtry' was held constant at a value of 50

```
modFit_3
```

Random Forest

4579 samples 58 predictor 5 classes: 'A', 'B', 'C', 'D', 'E'

No pre-processing Resampling results

Accuracy Kappa
0.9958506 0.994753

Tuning parameter 'mtry' was held constant at a value of 50

```
acc_1<-confusionMatrix(predict(modFit_1, cvdat2), cvdat2$classe);acc_1
```

Confusion Matrix and Statistics

```
      Reference
```

Prediction A B C D E A 1674 8 0 0 0 B 0 1127 17 0 2 C 0 4 1006 8 5 D 0 0 3 956 7 E 0 0 0 0 1068

Overall Statistics

```
           Accuracy : 0.9908
             95% CI : (0.988, 0.9931)
No Information Rate : 0.2845
P-Value [Acc > NIR] : < 2.2e-16

              Kappa : 0.9884
```

Mcnemar's Test P-Value : NA

Statistics by Class:

```
                 Class: A Class: B Class: C Class: D Class: E
```

Sensitivity 1.0000 0.9895 0.9805 0.9917 0.9871 Specificity 0.9981 0.9960 0.9965 0.9980 1.0000 Pos Pred Value 0.9952 0.9834 0.9834 0.9896 1.0000 Neg Pred Value 1.0000 0.9975 0.9959 0.9984 0.9971 Prevalence 0.2845 0.1935 0.1743 0.1638 0.1839 Detection Rate 0.2845 0.1915 0.1709 0.1624 0.1815 Detection Prevalence 0.2858 0.1947 0.1738 0.1641 0.1815 Balanced Accuracy 0.9991 0.9927 0.9885 0.9948 0.9935

```
acc_2<-confusionMatrix(predict(modFit_2,cvdat2), cvdat2$classe); acc_2
```

Confusion Matrix and Statistics

```
      Reference
```

Prediction A B C D E A 1674 6 0 0 0 B 0 1130 6 0 0 C 0 3 1020 9 0 D 0 0 0 954 4 E 0 0 0 1 1078

Overall Statistics

```
           Accuracy : 0.9951
             95% CI : (0.9929, 0.9967)
No Information Rate : 0.2845
P-Value [Acc > NIR] : < 2.2e-16

              Kappa : 0.9938
```

Mcnemar's Test P-Value : NA

Statistics by Class:

```
                 Class: A Class: B Class: C Class: D Class: E
```

Sensitivity 1.0000 0.9921 0.9942 0.9896 0.9963 Specificity 0.9986 0.9987 0.9975 0.9992 0.9998 Pos Pred Value 0.9964 0.9947 0.9884 0.9958 0.9991 Neg Pred Value 1.0000 0.9981 0.9988 0.9980 0.9992 Prevalence 0.2845 0.1935 0.1743 0.1638 0.1839 Detection Rate 0.2845 0.1920 0.1733 0.1621 0.1832 Detection Prevalence 0.2855 0.1930 0.1754 0.1628 0.1833 Balanced Accuracy 0.9993 0.9954 0.9958 0.9944 0.9980

```
acc_3<-confusionMatrix(predict(modFit_3,cvdat2), cvdat2$classe);acc_3
```

Confusion Matrix and Statistics

```
      Reference
```

Prediction A B C D E A 1674 2 0 0 0 B 0 1137 4 0 0 C 0 0 1015 5 0 D 0 0 7 955 0 E 0 0 0 4 1082

Overall Statistics

```
           Accuracy : 0.9963
             95% CI : (0.9943, 0.9977)
No Information Rate : 0.2845
P-Value [Acc > NIR] : < 2.2e-16

              Kappa : 0.9953
```

Mcnemar's Test P-Value : NA

Statistics by Class:

```
                 Class: A Class: B Class: C Class: D Class: E
```

Sensitivity 1.0000 0.9982 0.9893 0.9907 1.0000 Specificity 0.9995 0.9992 0.9990 0.9986 0.9992 Pos Pred Value 0.9988 0.9965 0.9951 0.9927 0.9963 Neg Pred Value 1.0000 0.9996 0.9977 0.9982 1.0000 Prevalence 0.2845 0.1935 0.1743 0.1638 0.1839 Detection Rate 0.2845 0.1932 0.1725 0.1623 0.1839 Detection Prevalence 0.2848 0.1939 0.1733 0.1635 0.1845 Balanced Accuracy 0.9998 0.9987 0.9941 0.9946 0.9996

All that is left is to predict against the test set.

```
predict(modFit_1, tedat2)
```

[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

```
predict(modFit_2, tedat2)
```

[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

```
predict(modFit_3, tedat2)
```

[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

Final Thoughts

Each model performs admirably well against the test data.

There are a number of additional steps and considerations that should be made along the way. I was not fond of removing the columns of data to simplify the analysis; however, in this case, I believe the results warranted the action. While I performed plenty of exploratory analysis, it is not included here. I encountered numerous outliers, but testing the algorithms showed that even with outliers, the models predicted at an accurate level.