Predicting Exercise Activity from Belt, Forearm, Arm, and Dumbell Accelerometers

Joseph Retzer

Preliminaries, reading / cleaning / subsetting data:

Read in training and test data

```
setwd("~/Documents/My Documents/Coursera/DataScienceTrack/PracticalMachineLearning/project/data")
library(Amelia); library(caret); library(xgboost); library(Ckmeans.1d.dp); library(data.table)
```

```
## Warning: package 'Amelia' was built under R version 3.2.3
```

```
## Loading required package: Rcpp
## ##
## ## Amelia II: Multiple Imputation
## ## (Version 1.7.4, built: 2015-12-05)
## ## Copyright (C) 2005-2016 James Honaker, Gary King and Matthew Blackwell
## ## Refer to http://gking.harvard.edu/amelia/ for more information
## ##
```

```
## Warning: package 'caret' was built under R version 3.2.3
```

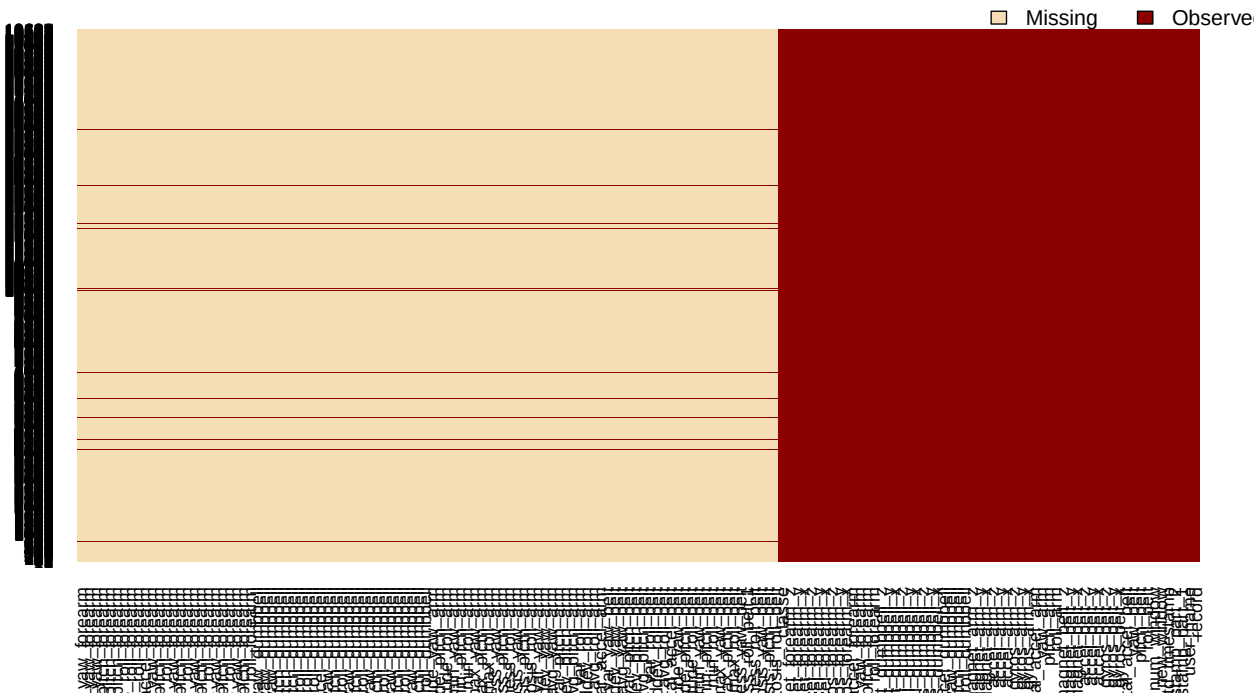
```
## Loading required package: lattice
## Loading required package: ggplot2
```

```
## Warning: package 'data.table' was built under R version 3.2.3
```

```
trainDat <- read.csv(file = "pml-training.csv", header = TRUE, stringsAsFactors = FALSE, check.names =
names(trainDat) <- tolower(names(trainDat))
names(trainDat)[1] <- "record"

testDat  <- read.csv(file = "pml-testing.csv",  header = TRUE, stringsAsFactors = FALSE, check.names =
names(testDat) <- tolower(names(testDat))
names(testDat)[1] <- "record"

missmap(trainDat)         # view missing pattern
```



The missing data mapping shows clearly that variables are either complete (no missings) or virtually empty. The next step involves identifying all variables with mostly missing values and dropping them from the data sets. Additional non-informative variables are also eliminated.

```
# find variables w/ virtually all missing
 propmiss <- function(dataframe) {
    m <- sapply(dataframe, function(x) {
     data.frame(
     nmiss=sum(is.na(x)),
     n=length(x),
     propmiss=sum(is.na(x))/length(x)
     )
    })
    d <- data.frame(t(m))
    d <- sapply(d, unlist)
    d <- as.data.frame(d)
    d$variable <- row.names(d)
    row.names(d) <- NULL
    d <- cbind(d[ncol(d)],d[-ncol(d)])
    return(d[order(d$propmiss), ])
    }

 # show same set of variables have have approx 98% missing in both training and test data sets

  trainProp <-  propmiss(trainDat); testProp  <-  propmiss(trainDat)    # prop of missing either 0 or
  missTrain <- subset(trainProp, select=variable, nmiss != 0); missTest <- subset(testProp, select=vari
  # cbind(missTrain, missTest)         # show 98% missing variables same in both data sets

  # drop all vars with 98% missing values

  trainDat2 <- subset(trainDat, select = !(names(trainDat) %in% as.character(missTrain$variable)))
```

```
testDat2  <- subset(testDat,  select = !(names(testDat)  %in% as.character(missTest$variable)))

trainDat2$classe <- as.factor(trainDat2$classe)

# also drop non-informative variables: record

dropVars  <- c("raw_timestamp_part_1", "raw_timestamp_part_2", "cvtd_timestamp", "num_window", "prob

trainDat3 <- subset(trainDat2, select = !(names(trainDat2) %in% dropVars))
```

Model Building Process

This paper predicts exercise activity:

A. sitting-down,
B. standing-up,
C. standing,
D. walking,
E. sitting

using accelererometer data from:

A. Belt,
B. Forearm
B. Arm
B. Dumbell

K-fold Cross Validation in Model Building

The k-fold cross validation method involves splitting the dataset into k-subsets. For each subset is held out while the model is trained on all other subsets. This process is completed until accuracy is determine for each instance in the dataset, and an overall accuracy estimate is provided.

Three models were built: gbm, lda and xgbTree. K-fold cross validation was employed using the "train" function from the caret package.

```
inTrain  = createDataPartition(trainDat3$classe, p = 3/4)[[1]]
training = trainDat3[ inTrain,]
testing  = trainDat3[-inTrain,]

set.seed(8675309)

   fitControl <- trainControl(method = "cv", number = 10, repeats = 2, search = "random")

 # mod2 <- train(classe ~.,method="gbm",    data=training)
 #    save(mod2,file="gbmMod2.RData")
  load("~/Documents/My Documents/Coursera/DataScienceTrack/PracticalMachineLearning/project/data/gbmM

 # mod3 <- train(classe ~.,method="lda"    ,data=training)
 #    save(mod3,file="ldaMod3.RData")
```

```
    load("~/Documents/My Documents/Coursera/DataScienceTrack/PracticalMachineLearning/project/data/ldaM

   # mod4 <- train(classe ~.,method="xgbTree",data=training, trControl = fitControl)
   #    save(mod4,file="xgboostMod4.RData")
    load("~/Documents/My Documents/Coursera/DataScienceTrack/PracticalMachineLearning/project/data/xgbo
```

CV based Model Performance

Each models final parameter set was determined by comparing k-fold CV averaged predictive performace shown below:

```
# CV results
   mod2
```

```
## Stochastic Gradient Boosting
##
## 14718 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 14718, 14718, 14718, 14718, 14718, 14718, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa      Accuracy SD
##   1                   50      0.7492613  0.6821643  0.007317127
##   1                  100      0.8181972  0.7699285  0.006380304
##   1                  150      0.8514757  0.8120643  0.005421326
##   2                   50      0.8525278  0.8131679  0.005951952
##   2                  100      0.9052730  0.8801090  0.004228898
##   2                  150      0.9291965  0.9104029  0.003598788
##   3                   50      0.8950922  0.8671728  0.005041517
##   3                  100      0.9387051  0.9224373  0.003556771
##   3                  150      0.9574623  0.9461809  0.002927828
##   Kappa SD
##   0.009255667
##   0.008071090
##   0.006853474
##   0.007555691
##   0.005333718
##   0.004532515
##   0.006344975
##   0.004467551
##   0.003675014
##
## 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.
```

```
mod3
```

```
## Linear Discriminant Analysis
##
## 14718 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 14718, 14718, 14718, 14718, 14718, 14718, ...
## Resampling results
##
##   Accuracy   Kappa      Accuracy SD  Kappa SD
##   0.7001334  0.6207655  0.005738434  0.007242442
##
##
```

```
mod4
```

```
## eXtreme Gradient Boosting
##
## 14718 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 13246, 13248, 13246, 13245, 13246, 13245, ...
## Resampling results across tuning parameters:
##
##   eta         max_depth  gamma      colsample_bytree  min_child_weight
##   0.05849168  2          0.2998670  0.4323682         17
##   0.05849168  2          0.5135390  0.4160469          0
##   0.05849168  2          0.7682912  0.5093364         17
##   0.05849168  2          4.4426655  0.5671522          9
##   0.05849168  2          5.1946720  0.3418706         20
##   0.05849168  2          5.6547822  0.4323682         17
##   0.05849168  2          6.9664761  0.5093364         17
##   0.05849168  2          8.1115694  0.4160469          0
##   0.05849168  2          8.7137345  0.3418706         20
##   0.05849168  2          9.5361553  0.5671522          9
##   0.22170471  9          0.5830640  0.5187909          3
##   0.22170471  9          0.7620013  0.5187909          3
##   0.22170471  9          1.2456287  0.3454626          2
##   0.22170471  9          2.6387740  0.5369308          3
##   0.22170471  9          2.8565179  0.3060554          9
##   0.22170471  9          4.1552519  0.4384999         13
##   0.22170471  9          6.7497977  0.5369308          3
##   0.22170471  9          8.1892965  0.4384999         13
##   0.22170471  9          8.6974608  0.3060554          9
##   0.22170471  9          8.9321864  0.3454626          2
##   0.48783587  8          0.5833858  0.3179095          2
```

```
##   0.48783587  8          2.3892724  0.6907137         11
##   0.48783587  8          2.4203794  0.6907137         11
##   0.48783587  8          3.0376060  0.4144921         17
##   0.48783587  8          6.4622404  0.3782736          6
##   0.48783587  8          7.7744741  0.3434315         14
##   0.48783587  8          9.2232623  0.3179095          2
##   0.48783587  8          9.6264240  0.4144921         17
##   0.48783587  8          9.8229270  0.3434315         14
##   0.48783587  8          9.9208665  0.3782736          6
##   nrounds  Accuracy   Kappa      Accuracy SD  Kappa SD
##    18      0.7223779  0.6465654  0.010910397  0.013940677
##   362      0.9281813  0.9091441  0.008860984  0.011234637
##   746      0.9663669  0.9574557  0.003620373  0.004578866
##   943      0.9614745  0.9512659  0.005603228  0.007087998
##    80      0.8203542  0.7721624  0.011075869  0.014124158
##   273      0.9103790  0.8865921  0.010279070  0.013020066
##   586      0.9480885  0.9343197  0.005728371  0.007260284
##   819      0.9462543  0.9320041  0.007204564  0.009127269
##    38      0.7652521  0.7016791  0.015049132  0.019045218
##   930      0.9400034  0.9240856  0.006217394  0.007875580
##   869      0.9949037  0.9935539  0.002033319  0.002572045
##   341      0.9947678  0.9933821  0.001472058  0.001861618
##   316      0.9932050  0.9914048  0.002175326  0.002752018
##    89      0.9902832  0.9877091  0.002380330  0.003010559
##   572      0.9910307  0.9886545  0.001568004  0.001983528
##   373      0.9881097  0.9849600  0.001577637  0.001994815
##   354      0.9840329  0.9798024  0.001674071  0.002116351
##   427      0.9828104  0.9782543  0.002479202  0.003140612
##   160      0.9800245  0.9747330  0.002029855  0.002564821
##   609      0.9820631  0.9773099  0.001898180  0.002401861
##   255      0.9936130  0.9919217  0.002291853  0.002898708
##   303      0.9886519  0.9856464  0.002782721  0.003520048
##   898      0.9900789  0.9874513  0.002708748  0.003426890
##   508      0.9889245  0.9859900  0.002270239  0.002872315
##   120      0.9819277  0.9771410  0.002733678  0.003457022
##   132      0.9794139  0.9739573  0.003414661  0.004320329
##   233      0.9779174  0.9720662  0.003505203  0.004434325
##   622      0.9783252  0.9725804  0.001942633  0.002461842
##   603      0.9771703  0.9711216  0.002782059  0.003519716
##   128      0.9766272  0.9704350  0.002358687  0.002983131
##
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were nrounds = 869, max_depth = 9,
##  eta = 0.2217047, gamma = 0.583064, colsample_bytree = 0.5187909
##  and min_child_weight = 3.
```

Expected Out of Sample Error

Each models expected predictive performance was estimated by predicting the hold out test data.

```
# Expected Performance
        pred2 <- predict(mod2,testing); confusionMatrix(pred2,testing$classe)
```

```
## 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 2.1.1
## Loading required package: plyr
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1375   23    0    1    0
##          B   14  906   19    5    9
##          C    3   19  824   18    8
##          D    2    1   11  773    7
##          E    1    0    1    7  877
##
## Overall Statistics
##
##                Accuracy : 0.9696
##                  95% CI : (0.9644, 0.9742)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9616
##  Mcnemar's Test P-Value : 0.004765
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9857   0.9547   0.9637   0.9614   0.9734
## Specificity            0.9932   0.9881   0.9881   0.9949   0.9978
## Pos Pred Value         0.9828   0.9507   0.9450   0.9736   0.9898
## Neg Pred Value         0.9943   0.9891   0.9923   0.9925   0.9940
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2804   0.1847   0.1680   0.1576   0.1788
## Detection Prevalence   0.2853   0.1943   0.1778   0.1619   0.1807
## Balanced Accuracy      0.9894   0.9714   0.9759   0.9782   0.9856
```

```
        pred3 <- predict(mod3,testing); confusionMatrix(pred3,testing$classe)
```

```
## Loading required package: MASS
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction    A    B    C    D    E
##          A 1154  173   87   39   34
##          B   22  586   81   40  149
##          C  109  104  557   97   79
##          D  103   34  104  598   74
##          E    7   52   26   30  565
##
## Overall Statistics
##
##                Accuracy : 0.7055
##                  95% CI : (0.6926, 0.7183)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.627
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8272   0.6175   0.6515   0.7438   0.6271
## Specificity            0.9051   0.9262   0.9039   0.9232   0.9713
## Pos Pred Value         0.7761   0.6674   0.5888   0.6550   0.8309
## Neg Pred Value         0.9295   0.9098   0.9247   0.9484   0.9205
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2353   0.1195   0.1136   0.1219   0.1152
## Detection Prevalence   0.3032   0.1790   0.1929   0.1862   0.1387
## Balanced Accuracy      0.8662   0.7718   0.7777   0.8335   0.7992
```

```
pred4 <- predict(mod4,testing); confusionMatrix(pred4,testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    0    0    0    0
##          B    0  949    1    0    0
##          C    0    0  851    1    0
##          D    0    0    3  803    0
##          E    0    0    0    0  901
##
## Overall Statistics
##
##                Accuracy : 0.999
##                  95% CI : (0.9976, 0.9997)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9987
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
```

```
## Sensitivity            1.0000   1.0000   0.9953   0.9988   1.0000
## Specificity            1.0000   0.9997   0.9998   0.9993   1.0000
## Pos Pred Value         1.0000   0.9989   0.9988   0.9963   1.0000
## Neg Pred Value         1.0000   1.0000   0.9990   0.9998   1.0000
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2845   0.1935   0.1735   0.1637   0.1837
## Detection Prevalence   0.2845   0.1937   0.1737   0.1644   0.1837
## Balanced Accuracy      1.0000   0.9999   0.9975   0.9990   1.0000
```

Overall expected error rate:

```
 1. GBM:     .971   (Error = 1 - .971 = .029)
 2. LDA:     .702   (Error = 1 - .702 = .298)
 3: XGBoost: .999   (Error = 1 - .999 = .001)
```

Choices made

Varibles with extreme nubers of missing were dropped as they could not be reliably imputed. Multiple models were fit using all the data and compared based on predictive accuracy to see if:

```
1. additional data transformations were necessary
2. which model proved best
```

It was clear that the predictive performance of the xgboost algorithm was far superior to the others. In addition since its accuracy was remarkable high, no additional transformations were felt necessary and the xgboos model chosen for prediction.

Prediction for the test data provided (w/o labels) was performed as follows:

```
        predTest <- predict(mod4,testDat2)
        predTest
```

```
##  [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
```