Practical Machine Learning Assignment

Long Huynh

27 March 2016

```r
pml_train <- suppressWarnings(read_csv('https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training
pml_test <- suppressWarnings(read_csv('https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.

vStartDt <-  pml_train$cvtd_timestamp %>% ymd_hms() %>% min()
vEndDt <- pml_train$cvtd_timestamp %>% ymd_hms() %>% max()
vTrainRows <- pml_train$user_name %>% unique %>% length()
vTestRows <- pml_train$classe %>% unique %>% length()
```

Introduction

In this assignment I will be using a series of machine learning techniques to predict the correct execution of weight lifting techniques. The data is based off a project undertaken in 2011 between November to December with the source found in the following link http://groupware.les.inf.puc-rio.br/har. In the project there were 6 subjects that were asked to perform barbell lifts in 5 different ways (each lift being categorized and ranked between A to E). This data was captured from a accelerometer on the belt, forearm, arm and dumbbell and each lift was assigned to 5different classes. For this assignment I will using the data taken from the accelerometers in 2011 to predict the classes of the 5 different barbell lifts.

Reviewing the structure of the data

Before beginning to choose my machine learning strategy I needed to explore the data, ensuring the data is suitable to undertake analysis. The dataset from the study is split into 2 parts a test and training dataset. I will be building my machine learning models on the training dataset. In total there are 160 variables with train dataset containing 19622 observations and the test set containing 20.

```r
str(pml_train) #Review of the structure of the dataset.
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    19622 obs. of  160 variables:
##  $                         : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ user_name               : chr  "carlitos" "carlitos" "carlitos" "carlitos" ...
##  $ raw_timestamp_part_1    : int  1323084231 1323084231 1323084231 1323084232 1323084232 1323084232 
##  $ raw_timestamp_part_2    : int  788290 808298 820366 120339 196328 304277 368296 440390 484323 484
##  $ cvtd_timestamp          : chr  "05/12/2011 11:23" "05/12/2011 11:23" "05/12/2011 11:23" "05/12/20
##  $ new_window              : chr  "no" "no" "no" "no" ...
##  $ num_window              : int  11 11 11 12 12 12 12 12 12 12 ...
##  $ roll_belt               : num  1.41 1.41 1.42 1.48 1.48 1.45 1.42 1.42 1.43 1.45 ...
##  $ pitch_belt              : num  8.07 8.07 8.07 8.05 8.07 8.06 8.09 8.13 8.16 8.17 ...
##  $ yaw_belt                : num  -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 ...
##  $ total_accel_belt        : int  3 3 3 3 3 3 3 3 3 3 ...
##  $ kurtosis_roll_belt      : chr  "" "" "" "" ...
##  $ kurtosis_picth_belt     : chr  "" "" "" "" ...
##  $ kurtosis_yaw_belt       : chr  "" "" "" "" ...
##  $ skewness_roll_belt      : chr  "" "" "" "" ...
##  $ skewness_roll_belt.1    : chr  "" "" "" "" ...
##  $ skewness_yaw_belt       : chr  "" "" "" "" ...
##  $ max_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_belt          : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_belt            : chr  "" "" "" "" ...
##  $ min_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_belt          : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_belt            : chr  "" "" "" "" ...
##  $ amplitude_roll_belt     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_pitch_belt    : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_yaw_belt      : chr  "" "" "" "" ...
##  $ var_total_accel_belt    : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_roll_belt        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_pitch_belt       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_yaw_belt         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ gyros_belt_x            : num  0 0.02 0 0.02 0.02 0.02 0.02 0.02 0.02 0.03 ...
##  $ gyros_belt_y            : num  0 0 0 0 0.02 0 0 0 0 0 ...
##  $ gyros_belt_z            : num  -0.02 -0.02 -0.02 -0.03 -0.02 -0.02 -0.02 -0.02 -0.02 0 ...
##  $ accel_belt_x            : int  -21 -22 -20 -22 -21 -21 -22 -22 -20 -21 ...
##  $ accel_belt_y            : int  4 4 5 3 2 4 3 4 2 4 ...
##  $ accel_belt_z            : int  22 22 23 21 24 21 21 21 24 22 ...
##  $ magnet_belt_x           : int  -3 -7 -2 -6 -6 0 -4 -2 1 -3 ...
##  $ magnet_belt_y           : int  599 608 600 604 600 603 599 603 602 609 ...
##  $ magnet_belt_z           : int  -313 -311 -305 -310 -302 -312 -311 -313 -312 -308 ...
##  $ roll_arm                : num  -128 -128 -128 -128 -128 -128 -128 -128 -128 -128 ...
##  $ pitch_arm               : num  22.5 22.5 22.5 22.1 22.1 22 21.9 21.8 21.7 21.6 ...
##  $ yaw_arm                 : num  -161 -161 -161 -161 -161 -161 -161 -161 -161 -161 ...
##  $ total_accel_arm         : int  34 34 34 34 34 34 34 34 34 34 ...
##  $ var_accel_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_roll_arm         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_pitch_arm        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_yaw_arm             : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_yaw_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_yaw_arm             : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ gyros_arm_x             : num  0 0.02 0.02 0.02 0 0.02 0 0.02 0.02 0.02 ...
##  $ gyros_arm_y             : num  0 -0.02 -0.02 -0.03 -0.03 -0.03 -0.03 -0.02 -0.03 -0.03 ...
##  $ gyros_arm_z             : num  -0.02 -0.02 -0.02 0.02 0 0 0 0 -0.02 -0.02 ...
##  $ accel_arm_x             : int  -288 -290 -289 -289 -289 -289 -289 -289 -288 -288 ...
##  $ accel_arm_y             : int  109 110 110 111 111 111 111 111 109 110 ...
##  $ accel_arm_z             : int  -123 -125 -126 -123 -123 -122 -125 -124 -122 -124 ...
##  $ magnet_arm_x            : int  -368 -369 -368 -372 -374 -369 -373 -372 -369 -376 ...
##  $ magnet_arm_y            : int  337 337 344 344 337 342 336 338 341 334 ...
##  $ magnet_arm_z            : int  516 513 513 512 506 513 509 510 518 516 ...
##  $ kurtosis_roll_arm       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ kurtosis_picth_arm      : chr  "" "" "" "" ...
##  $ kurtosis_yaw_arm        : chr  "" "" "" "" ...
##  $ skewness_roll_arm       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_pitch_arm      : chr  "" "" "" "" ...
##  $ skewness_yaw_arm        : chr  "" "" "" "" ...
##  $ max_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_arm             : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_arm             : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_roll_arm      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_pitch_arm     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_yaw_arm       : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ roll_dumbbell           : num  13.1 13.1 12.9 13.4 13.4 ...
##  $ pitch_dumbbell          : num  -70.5 -70.6 -70.3 -70.4 -70.4 ...
##  $ yaw_dumbbell            : num  -84.9 -84.7 -85.1 -84.9 -84.9 ...
##  $ kurtosis_roll_dumbbell  : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ kurtosis_picth_dumbbell : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ kurtosis_yaw_dumbbell   : chr  "" "" "" "" ...
##  $ skewness_roll_dumbbell  : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_pitch_dumbbell : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_yaw_dumbbell   : chr  "" "" "" "" ...
##  $ max_roll_dumbbell       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_dumbbell        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_roll_dumbbell       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_dumbbell        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_roll_dumbbell : num  NA NA NA NA NA NA NA NA NA NA ...
##   [list output truncated]
##  - attr(*, "problems")=Classes 'tbl_df', 'tbl' and 'data.frame': 185 obs. of  4 variables:
##   ..$ row     : int  2231 2231 2255 2255 2282 2282 2314 2314 2422 2422 ...
##   ..$ col     : chr  "kurtosis_roll_arm" "skewness_roll_arm" "kurtosis_roll_arm" "skewness_roll_arm"
##   ..$ expected: chr  "a double" "a double" "a double" "a double" ...
##   ..$ actual  : chr  "#DIV/0!" "#DIV/0!" "#DIV/0!" "#DIV/0!" ...
```

Removing variable unsuitable for models

When reviewing the structure of the train dataset I noticed a couple of issues that needed to resolved before I could begin my machine learning models:

- Some variables were incorrectly classed e.g. numeric variables allocated to character variables.
- Some variables contained NA values.

I also wanted to identify if there were any columns that were unsuitable for machine learning techniques.

- Do any columns show significant variability?
- Are there any columns that are too closely correlated?

Identifying incorrectly classed variables

```r
pml_train %<>% #Tidy Training data - Column classes
    mutate(user_name = factor(user_name), #few uses and there factor
           cvtd_timestamp = dmy_hm(cvtd_timestamp), #Turn into a date format
           new_window = factor(new_window), #New_window is factor variable
           classe = factor(classe)) %>% #Classe is factor variable
    mutate_each_(funs(suppressWarnings(as.numeric(.))), #Turn columns into numeric values
                 names(pml_train) %>% grep('belt|arm|dumbbell|forearm', x =.) %>% names(pml_train)[.])

vClasse <- pml_train$classe %>% unique #classes

pml_test %<>% #Tidy Training data - Column classes
    mutate(user_name = factor(user_name), #few uses and there factor
           cvtd_timestamp = dmy_hm(cvtd_timestamp), #Turn into a date format
           new_window = factor(new_window)) %>% #Classe is factor variable
    mutate_each_(funs(suppressWarnings(as.numeric(.))), #Turn columns into numeric values
                 names(pml_test) %>% grep('belt|arm|dumbbell|forearm', x =.) %>% names(pml_test)[.])
```

I realigned each column to its correct class based on my observations. E.g there were numeric variables catergorised as character variables.

Identifying Variables containing NA values

```r
keepNms <- #Name vector identifying columns with greater than 97% of values not NA
    pml_train %>%
    sapply(function(x){is.na(x) %>% sum}) %>% #Find the number of NA values across columns
    {(.)/dim(pml_train)[1]} %>% #Propotion NA values
    {(.) < 0.97} %>% # Find columns that have less that 97% NA values
    grep(T, x = .) %>% # Get names
    names(pml_train)[.] %>%  # Remove ID variable
    .[2:length(.)]


NaNms <- # Imputing missing data - magnet_dumbbell
    pml_train[, keepNms] %>%
    sapply(function(x){suppressWarnings(is.na(x)) %>% sum}) %>%
    {(.) > 0} %>%
    grep(T, x = .) %>%
    names(pml_train[, keepNms])[.]
```

We know that majority of machine learning techniques do not work well with NA values. Therefore I identified all columns that have less than 97% NA values. Of these column I then identified the remaining columns with 3% or below NA values. I will use k nearest neighbor to place an average value ensuring we do not lose remaining data.

Identify variables that lack of variability

```r
NmsLackVariablity <- # Identify columns with lack of variability
    pml_train[, keepNms] %>%
    caret::nearZeroVar() %>%
    names(pml_train[, keepNms])[.]
```

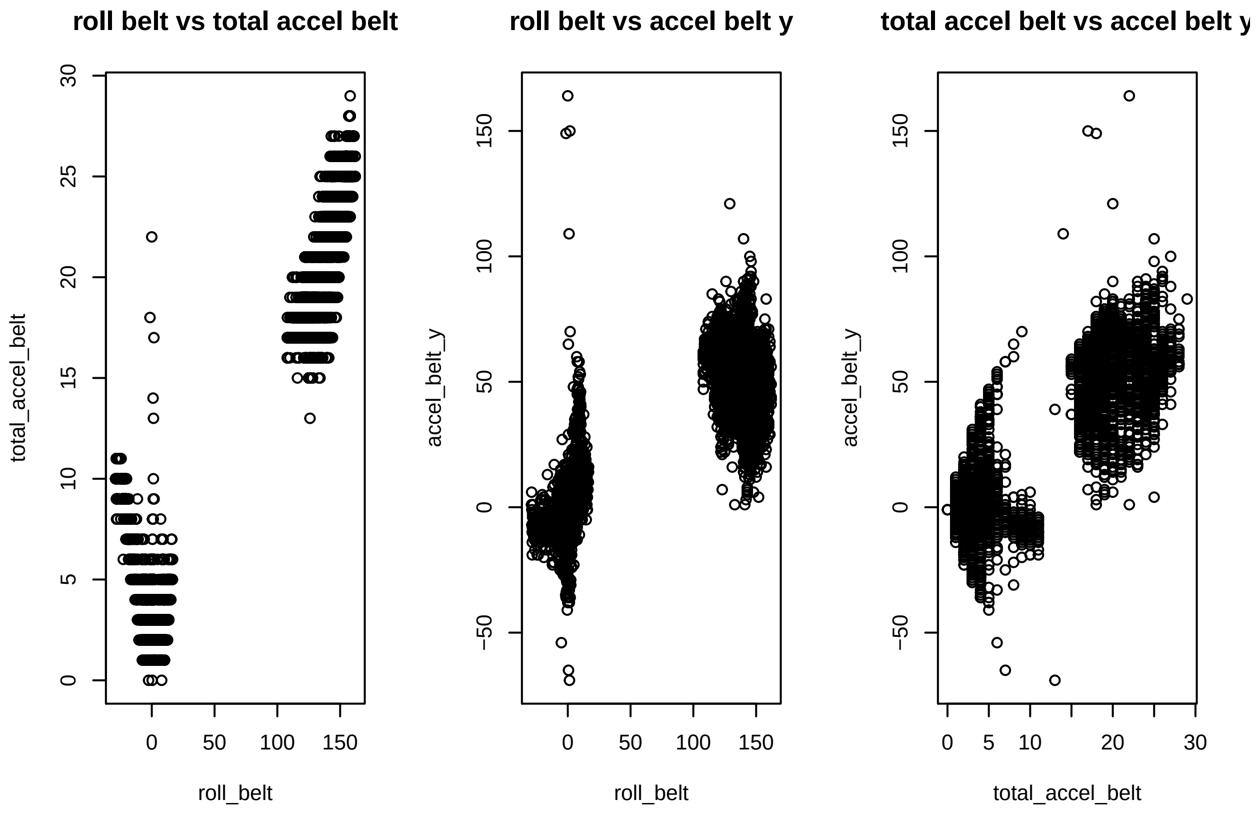
Columns that lack variability will not help the machine learning models, therefore I reviewed each columns using the nearZeroVar from the caret package. I found new_window lacked variability and I therefore removed this column.

Identify any columns that are highly correlated

```r
tempTrain <- #Temporary table to review highly correlated colums
    pml_train[, keepNms] %>%
    select(num_window:classe) %>%
    filter(!is.na(magnet_dumbbell_z))

NmsCorrelated <- # Identify columns that were highly corrleated
    tempTrain %>%
    select(-classe) %>%
    mutate_each(funs(as.numeric)) %>%
    cor() %>%
    {which(.>0.9, arr.ind = T)} %>% # Variables with 90% or greater correlation
    as_data_frame() %>%
    mutate(low = ifelse(row < col , row, col),
           high = ifelse(row > col , row, col)) %>%
    select(low:high) %>%
    filter(low != high) %>%
    unique

par(mfrow = c(1, 3))
for(i in 1:dim(NmsCorrelated)[1]){
    tempTrain[, c(NmsCorrelated[[i, 1]], NmsCorrelated [[i, 2]])] %>%
    plot(main = paste(names(.[, 1]),'vs',names(.[, 2])) %>% str_replace_all('_',' '))
}
```



I then wanted to review if there was any columns that were highly correlated and would transform/removed such columns. From reviewing the highly correlated columns (where the correlation exceeded 90%) I did not feel there was any columns I should transform/removed based on the plot patterns.

Clean and shuffle the training dataset based on findings.

```
set.seed(34135) # Set random seed
trainClean <- # Clean data that can be split into
    pml_train[, keepNms] %>%
    dplyr::select(-one_of(NmsLackVariablity), -user_name:-cvtd_timestamp) %>%
    as.data.frame() %>% #kNN only works when convered into base data frame
    VIM::kNN() %>% #Use k nearest neighbour to impute NA values
    as_data_frame() %>% #Convert back to tibble
    select(-ends_with('_imp')) %>%
    .[sample(nrow(.)),] #Shuffle the dataset
```

```
## Time difference of 3.086055 secs
```

```
testClean <-
    pml_test[, keepNms[1:length(keepNms)-1]] %>%
    dplyr::select(-one_of(NmsLackVariablity), -user_name:-cvtd_timestamp)
```

On both my train and test sets I removed columns with a high NA value and columns not needed for testing. Additionally I imputed any remaining NA values using k nearest neighbor. I opted not to transform the highly correlated columns (90% or greater) as there seems to be an interesting pattern based on the plots that were drawn

K Fold Cross validation

```r
trainProp <- 0.7; testProp <- 1 - trainProp; n <- nrow(trainClean)
train <- trainClean[1:round(trainProp * n),]
test <- trainClean[(round(trainProp * n) + 1):n,]
```

I split the data set into a test (30%) and training set (70%).

```r
nFolds <- 3
perf <- function(x, y) {sum(x == y)/length(x)}
# Set random seed. Don't remove this line.
set.seed(35343)
KFolds <- rep(0, nFolds) %>% as.list()

for(i in 1:nFolds){
    KFolds[[i]] <-
        train %>%
        mutate(TestID = 1, TestID = cumsum(TestID) %>% cut(nFolds, labels(1:nFolds)),
               set = ifelse(TestID == i, 'Test', 'Train')) %>%
        select(-TestID, -num_window) %>%
        group_by(set) %>%
        nest() %>%
        spread(set, data)
}
KFolds %<>% do.call('rbind', .); KFolds
```

```
## Source: local data frame [3 x 2]
##
##                 Test              Train
##               <list>             <list>
## 1 <tbl_df [4579,53]> <tbl_df [9156,53]>
## 2 <tbl_df [4578,53]> <tbl_df [9157,53]>
## 3 <tbl_df [4578,53]> <tbl_df [9157,53]>
```

To decide on the best machine learning model I will cross validate my training data set into 3 k folds. This will ensure that we provide a fair estimate of each models accuracy. Due to the size of the dataset and complexity of the different models I decided on only 3 partitions.

```r
accsRF <- rep(0, nFolds)
accsGBM <- rep(0, nFolds)
accsSVM <- rep(0, nFolds)
set.seed(34137)
for (i in 1:nFolds) {

  accsRF[i] <- # Random Forrest
      KFolds[i, 2] %>%
      unnest() %>%
      randomForest(classe ~ ., data = ., distribution = 'multinomial') %>%
      predict(KFolds[i, 1] %>% unnest, type = 'class') %>%
      perf(KFolds[i, 1] %>% unnest %>% .$classe)

  accsGBM[i] <- # Generalised Boosting
```

```
      KFolds[i, 2] %>%
      unnest() %>%
      gbm(classe ~ ., data = ., distribution = 'multinomial',
          n.trees = 200, interaction.depth = 5, shrinkage = 0.005) %>%
      predict(KFolds[i, 1] %>% unnest, n.trees = 200, type = 'response') %>%
      apply(1, which.max) %>%
      unique(vClasse)[.] %>%
      perf(KFolds[i, 1] %>% unnest %>% .$classe)

  accsSVM[i] <- # SVM
      KFolds[i, 2] %>%
      unnest() %>%
      svm(classe ~ ., data = .) %>%
      predict(KFolds[i, 1] %>% unnest, type = 'class') %>%
      perf(KFolds[i, 1] %>% unnest %>% .$classe)

}

# Print out the mean of accuracy rates
accsRF %>% print() #Random Forest
```

```
## [1] 0.9917012 0.9908257 0.9897335
```

```
summary(accsRF)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.9897  0.9903  0.9908  0.9908  0.9913  0.9917
```

```
accsGBM %>% print() #Generalised Boosting
```

```
## [1] 0.8196113 0.8202272 0.8245959
```

```
summary(accsGBM)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.8196  0.8199  0.8202  0.8215  0.8224  0.8246
```

```
accsSVM %>% print() #Support Vector Machine
```

```
## [1] 0.9290238 0.9191787 0.9298820
```

```
summary(accsSVM)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.9192  0.9241  0.9290  0.9260  0.9295  0.9299
```

I undertook 3 powerful machine learning techniques Random Forrest, Generalised Boosting and Support Vector Machines, which have historically performed well in Kaggle competitions. The data highlights that the Random Forest performed the best out of the 3 models. Its mean was 0.99 which performed significantly better than the 2 other models. Random Forest tends to perform better with a larger number of variables and are not significantly affect by outliers (unlike SVM). Due to the excellent performance of Random Forest I did not feel I needed to ensemble the models. Additionally I did not want to add complexity to the model.

Out of Sample Error rate

```
set.seed(34138)
fitModelRF <- randomForest(classe ~ ., data = train, distribution = 'multinomial')
#fitModelRF <- readRDS('./fitModelRF.Rds')
#saveRDS(fitModelRF, 'fitModelRF.Rds')

predictRF <- predict(fitModelRF, test, type = 'class')
perf(predictRF, test$classe)
```

```
## [1] 0.9969424
```

```
fitModelRF
```

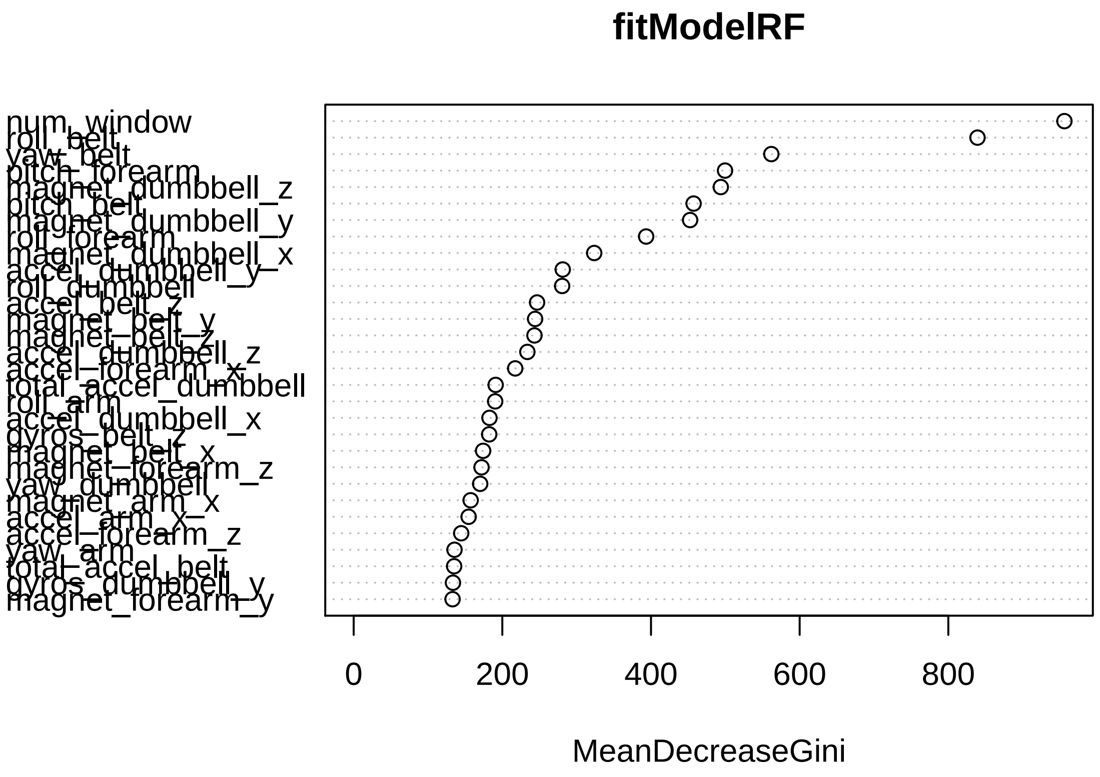
```
##
## Call:
##  randomForest(formula = classe ~ ., data = train, distribution = "multinomial")
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.26%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3930    0    0    0    1 0.0002543882
## B    3 2642    1    0    0 0.0015117158
## C    0   11 2380    1    0 0.0050167224
## D    0    0   13 2235    1 0.0062249889
## E    0    0    0    5 2512 0.0019864919
```

```
fitModelRF %>% summary
```

```
##                 Length Class  Mode
## call                4  -none- call
## type                1  -none- character
## predicted       13735  factor numeric
## err.rate         3000  -none- numeric
## confusion          30  -none- numeric
## votes           68675  matrix numeric
## oob.times       13735  -none- numeric
## classes             5  -none- character
## importance         53  -none- numeric
## importanceSD        0  -none- NULL
## localImportance     0  -none- NULL
## proximity           0  -none- NULL
## ntree               1  -none- numeric
## mtry                1  -none- numeric
## forest             14  -none- list
## y               13735  factor numeric
## test                0  -none- NULL
## inbag               0  -none- NULL
## terms               3  terms  call
```

The results from my model against my test set was 99.6942415% which is extremely high. I think the model maybe over fitting and I can actually reduce the number of variables.

```
varImpPlot(fitModelRF)
```



The plot above indicates that there are some variables that contributed to the model significantly more then others. I could therefore reduce the number of variables to increase its interpret-ability and reduce the risk of over fitting. Therefore I reduced the model to the top 15 variables which are shown on the plot above.

```
importantVar <- # Get the 15 most explained variables
    fitModelRF$importance %>% #Mean Decrease Gini variables
    data.frame %>% #Convert to dataframe for dplyr
    add_rownames(var = 'Variables') %>% #Need row names for formula
    arrange(desc(MeanDecreaseGini)) %>% #Arrange in descending order
    .$Variables %>%
    .[1:15] %>% # Top 15 only
    as.vector() %>%
    paste(collapse = ' + ') %>%
    paste('classe ~', .) # create formula for model

set.seed(34139)
fitModelRF2 <- randomForest(as.formula(importantVar), data = train, distribution = 'multinomial')
predictRF2 <- predict(fitModelRF, test, type = 'class')
perf(predictRF2, test$classe)
```

```
## [1] 0.9969424
```

```
fitModelRF2
```

```
##
## Call:
##  randomForest(formula = as.formula(importantVar), data = train,      distribution = "multinomial")
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 0.3%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3929    0    0    1    1 0.0005087764
## B    8 2630    6    2    0 0.0060468632
## C    0    7 2384    1    0 0.0033444816
## D    0    0    7 2240    2 0.0040017786
## E    0    1    0    5 2511 0.0023837902
```

```
fitModelRF2 %>% summary
```

```
##                 Length Class  Mode
## call                4  -none- call
## type                1  -none- character
## predicted       13735  factor numeric
## err.rate         3000  -none- numeric
## confusion          30  -none- numeric
## votes           68675  matrix numeric
## oob.times       13735  -none- numeric
## classes             5  -none- character
## importance         15  -none- numeric
## importanceSD        0  -none- NULL
## localImportance     0  -none- NULL
## proximity           0  -none- NULL
## ntree               1  -none- numeric
## mtry                1  -none- numeric
## forest             14  -none- list
## y               13735  factor numeric
## test                0  -none- NULL
## inbag               0  -none- NULL
## terms               3  terms  call
```

Despite reducing my variables my final result was very close to my original model which included all the variables. Therefore my final model is a Random Forrest using just 15 variables.

Test results based on my final model

```
predict(fitModelRF2, pml_test, type = 'class')
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  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
```

The final result from the test set is based against my final model.

Reference

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. http://groupware.les.inf.puc-rio.br/har#ixzz447iiJyEm