# Qualitative Activity Recognition

Machine Learning Class Project

*Erik Cornelsen*

## Synopsis

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 analysis we will use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants who were asked to perform barbell lifts correctly and incorrectly in 5 different ways. Our goal is to use to develop a machine learning model to correctly predict the manner in which they did the exercise based on new data. The "classe" variable in the training set is what we will predict.

## Data

Six young health participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions. Class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes as follows: exactly according to the specification (Class A), throwing the elbows to the front (Class B), lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) and throwing the hips to the front (Class E).

### Data Sourcing

The training and testing data for this project are available here:

```
* https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv
* https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv
* When loading the data I chose to split the pml-training data into training(75%) and validation(25%) da
* Loaded data sets are 'trn', 'vld', and 'tst'
```

The project page, documentation, and citation for the data and how it was gathered:

```
* Project Page: http://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise
* Further Documentation: http://groupware.les.inf.puc-rio.br/public/papers/2013.Velloso.QAR-WLE.pdf
* Citation for WLE data set:  Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. Qualitative
```

### Data Exploration

After downloading the data and inspecting it several things were notable:

```
* Training dataset had 160 variables and 19622 observations.
* Many columns were very sparse (almost entirely NA's).  These sparse variables may have predictive valu
* Every row had at least one NA as complete.cases() returns 0.
* ~300 rows had more data populated than most.  Seemed to correlate with new_window="yes".
```

```
df<-trn.all
# str(df, list.len=length(df))#check structure
sum(complete.cases(df)) #There are no complete cases due to NA's
## [1] 0
table(sapply(df,FUN=function(x) sum(is.na(x)))) #r1=num of NA's, r2=num of cols
##
##     0 19216 19217 19218 19220 19221 19225 19226 19227 19248 19293 19294
##    60    67     1     1     1     4     1     4     2     2     1     1
## 19296 19299 19300 19301 19622
##     2     1     4     2     6
table(df$new_window) #rows where new_window="yes" seem to have more complete data but are very few
##
##    no   yes
## 19216   406
```

### Data Processing

Took the following actions:

```
* Drop columns with 19k NA's (only keep columns with 0 NA's)
* Drop Factor columns that should not be used as predictors (X, user_name, raw_timestamp_part_1, raw_ti
* Convert new_window column to binary (instead of y/n factor)
* Use preProcess() to center and scale the variables
* Resulting Data sets are 'trn2', 'vld2', and 'tst2'
```

## Models

I decided to apply multiple models to the data and see which one performs best. Models to check include: Naive Bayes (nb), Random Forest (rf), Random Forest with PCA (rfPCA), and a 'Stacked Model'.

To reduce code and save time I created a helper function 'trainModel()' that is a wrapper around the 'train()' function. After a model is created using train() it will save it to a 'models/' folder. The next time it runs it will try to re-use the model from the models folder if it exists. This greatly reduces the time for each iteration. If I want to rebuild a model completly then I just need to delete it from the models folder.

### Cross Validation

With 19k observations in the training set, I was tempted to assume that cross-validation was not needed. However, when looking at the data I found that there were often only microseconds difference between the observations. It turns out that the data comprises 6 people completaketing 10 reps each, which is only 60 total 'observations'. The tools used to measure the reps are taking many measurements very quickly and closely together. Because we're really dealing with a total sample of 60 we do need to apply some cross validation while training our models. I chose to do cross validation with 3 folds (CV=3).

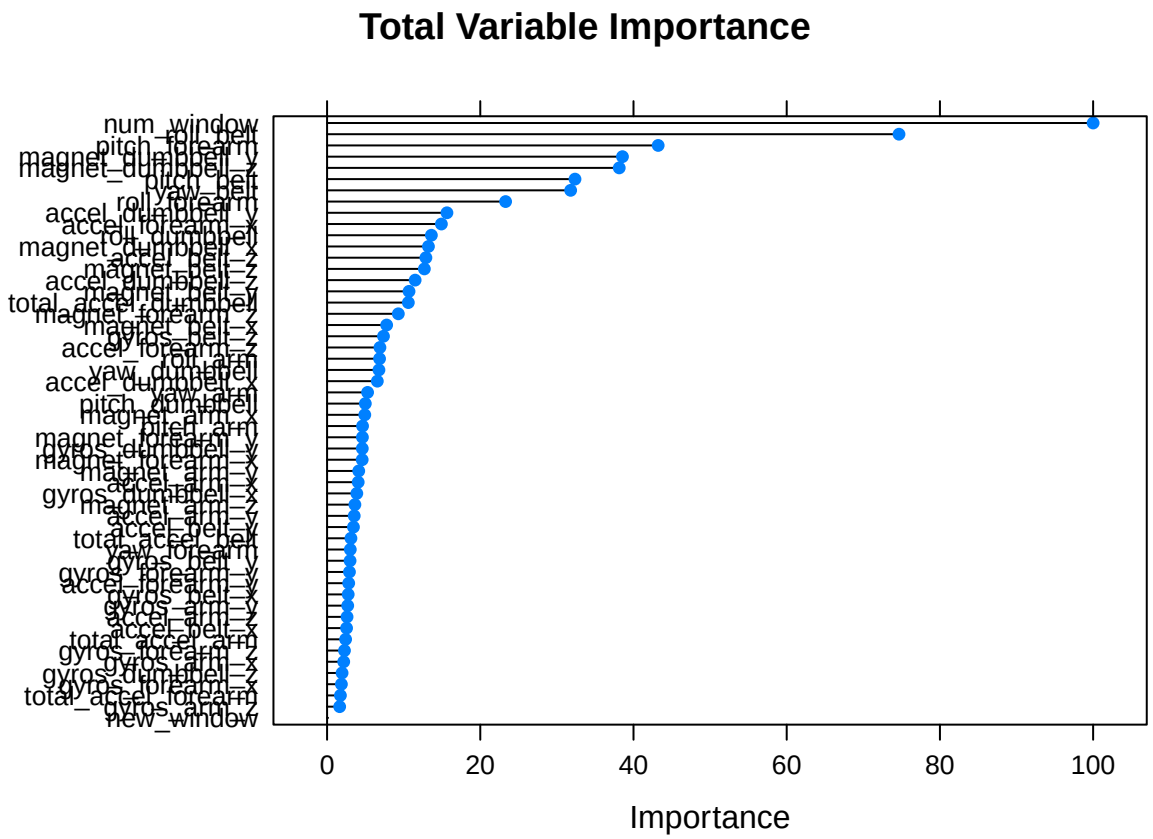
On further research I found that Cross-validation isn't really needed for Random Forests because the model inherently keeps a portion of the data out of each creation of a tree. This already prevents the problems that cross validation is used to address. So I'll use it for Naive Bayes but not for the Random Forest based models.

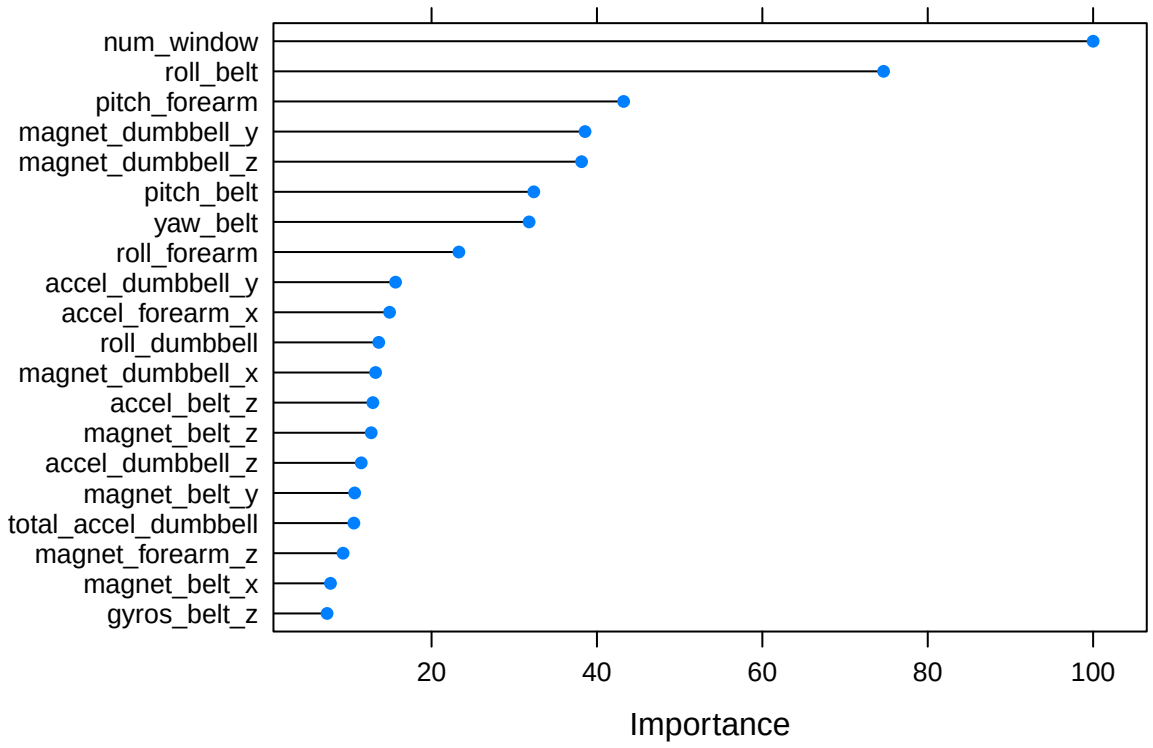
### Identify Most Important Vars by using Random Forest

With a subset of the data we will use a random forest to determine variable importance. Then we'll use this information to cull out any variables that are deemed unimportant before actually creating the RF model. This will help prevent overfitting of the model on the training data.

The plots show the normalized importance of the remaining input variables. The first plot shows all the variables and the second plot show the top 20. We learn that the top 20% to 25% of the variables dominate in importance, so the rest can be dropped.

```
## [1] "Training Model: method = rf ; fileName = models/rfModelVars.rds ; pp =  ; useCV = FALSE"
## [1] "Re-Using Model From  models/rfModelVars.rds"
## [1] "trainModel() complete"
```






We will use the top 25% of varibles based on their importance and then train the actual RF model(s) that we will use. We'll save the accuracy measures to help determine which one to keep and use. We're left with the following

## Model: Naive Bayes (nb)

Naive Bayes is a simple model to use as a baseline. It tries to classify instances based on the probabilities of previously seen attributes/instances, assuming complete attribute independence. It can handle missing data pretty well.

```
## [1] "Training Model: method = nb ; fileName = models/nbModel.rds ; pp =  ; useCV = TRUE"
## [1] "Re-Using Model From  models/nbModel.rds"
## [1] "trainModel() complete"
## Naive Bayes
##
## 14718 samples
##    14 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 9813, 9811, 9812
## Resampling results across tuning parameters:
##
##   usekernel  Accuracy   Kappa
```

```
##   FALSE      0.4813818  0.3424585
##    TRUE      0.7396378  0.6693923
##
## Tuning parameter 'fL' was held constant at a value of 0
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were fL = 0, usekernel = TRUE
##  and adjust = 1.
## Loading required package: klaR
## Loading required package: MASS
```

## Model: Random Forests (rf,parRF,gbm)

Random Forests grows many classification trees. To classify a new object from an input vector it puts the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

** Train Random Forest Models ** To compare how they each perform, I trained several random Forest models, predict using a validation data set, and capture their accuracies. Models tested were: 'rf', 'rf' with PCA, 'parRF', and 'gbm'.

```
#TRAIN & VALIDATE  #Random Forest
set.seed(123456)
rfModel <- trainModel(trn2, "rf", "rfModel.rds")
rfPredict_vld <- predict(rfModel,newdata = vld2);
# confusionMatrix(vld2$classe,rfPredict_vld)
rfAccuracy <- postResample(vld2$classe,rfPredict_vld)

#TRAIN & VALIDATE  #Random Forest With PCA
set.seed(1234567)
rfpcaModel <- trainModel(trn2, "rf", "rfpcaModel.rds",useCV=FALSE, "pca")
rfpcaPredict_vld <- predict(rfpcaModel,newdata = vld2);
# confusionMatrix(vld2$classe,rfpcaPredict_vld)
rfpcaAccuracy <- postResample(vld2$classe,rfpcaPredict_vld)

#TRAIN & VALIDATE  #parallel random Forest
set.seed(12345678)
parrfModel <- trainModel(trn2, "parRF", "parrfModel.rds")
parrfPredict_vld <- predict(parrfModel,newdata = vld2);
# confusionMatrix(vld2$classe,parrfPredict_vld)
parrfAccuracy <- postResample(vld2$classe,parrfPredict_vld)

#TRAIN & VALIDATE  #boosted trees, Stochastic Gradient Boosting
set.seed(123456789)
gbmModel <- trainModel(trn2, "gbm", "gbmModel.rds")
gbmPredict_vld <- predict(gbmModel,newdata = vld2);
# confusionMatrix(vld2$classe,gbmPredict_vld)
gbmAccuracy <- postResample(vld2$classe,gbmPredict_vld)

# Runtimes:    user  system elapsed
# nb         423.08    0.33  426.11
# rf         667.90    4.74  675.10
```

```
# rfpca           ?       ?       ?
# parrf      335.99    2.42  340.69
# gbm        359.95    0.81  366.53
```

## Results: Final Model Selection & Predictions

Here is the comparison of the different models. Based on this I choose to use "rf" model to make my predictions.

```
data.frame(nbAccuracy,rfAccuracy,rfpcaAccuracy,parrfAccuracy,gbmAccuracy)
##          nbAccuracy rfAccuracy rfpcaAccuracy parrfAccuracy gbmAccuracy
## Accuracy   0.742863  0.9981648     0.9528956     0.9979608   0.9926591
## Kappa      0.673440  0.9976787     0.9404028     0.9974208   0.9907139

finalPredictions <- predict(rfModel,newdata = tst2);
#finalPredictions
```

### Expected Out-Of-Sample-Error

We need to account for possible overfitting of the model on the train and validation data. This could be done with:

```
* In Sample Error (resubstitution error): the error rate you get on the same data set you used to build
* Out of Sample Error(generalization error): The error rate you get on a new data set.
```

We want to know the Out of Sample Error:

```
missClass = function(values, prediction) {
    sum(prediction != values)/length(values)
}
errRate = missClass(vld2$classe, finalPredictions)
errRate
## [1] 0.7781403
```

Based on the missclassificaiton rate on the validation subset, an unbiased estimate of the random forest's out-of-sample error rate is 77.8140294%.