Weight Lifting Exercise Prediction

Kirby Zhang

Monday, October 20, 2014

Abstract

Weight lifting data collected by Ugulino, Velloso, and Fuks is used to form a prediction model. The model classifies one of five barbell lifting methods based on accelerometer and gyro data recorded on the user. Using manual feature selection and cross validation within and between different modeling methods, a high degree of prediction accuracy is achieved.

Exploratory Analysis

We will go over the steps which were taken to preprocess the dataset and analyze its features (the data was obtained from here).

Tidy Data

```
library(knitr)
library(gbm)
library(caret)
library(lubridate)
library(plyr)
library(dplyr)
library(doParallel)

opts_chunk$set(warning=FALSE,message=FALSE)
```

```
setClass("quoted.numeric")
setAs("character", "quoted.numeric", function(from) as.numeric(gsub('"','',from)))
df <- read.csv("pml-training.csv",colClasses=c(
  "character",
  "factor",
  rep("character", 4),
  "integer",
  rep("quoted.numeric", 159-8 + 1),
  "factor"
))
```

The dataset consisted of 19622 observations with one outcome variable and 160 features. We created a "quoted.numeric" class to help us strip away quotation marks surrounding numeric values. A summary of the dataframe showed that columns 2 and 160 are appropriate as factor variables:

```
summary(df[,c(2,160)])
```

```
##     user_name    classe 
##  adelmo  :3892   A:5580  
##  carlitos:3112   B:3797  
##  charles :3536   C:3422  
##  eurico  :3070   D:3216  
##  jeremy  :3402   E:3607  
##  pedro   :2610
```

Feature Selection

Next we took a closer look at the 160 features. We start by looking at the "testing" dataset we received. We reasoned that features that are uninteresting in this small 20-element dataset cannot be useful predictor variables.

```
df_submit <- read.csv("pml-testing.csv")
dmy_hm(df$cvtd_timestamp) -> df$time
dmy_hm(df_submit$cvtd_timestamp) -> df_submit$time
select(df_submit, X, user_name, time, new_window, num_window) %>% arrange(user_name)
```

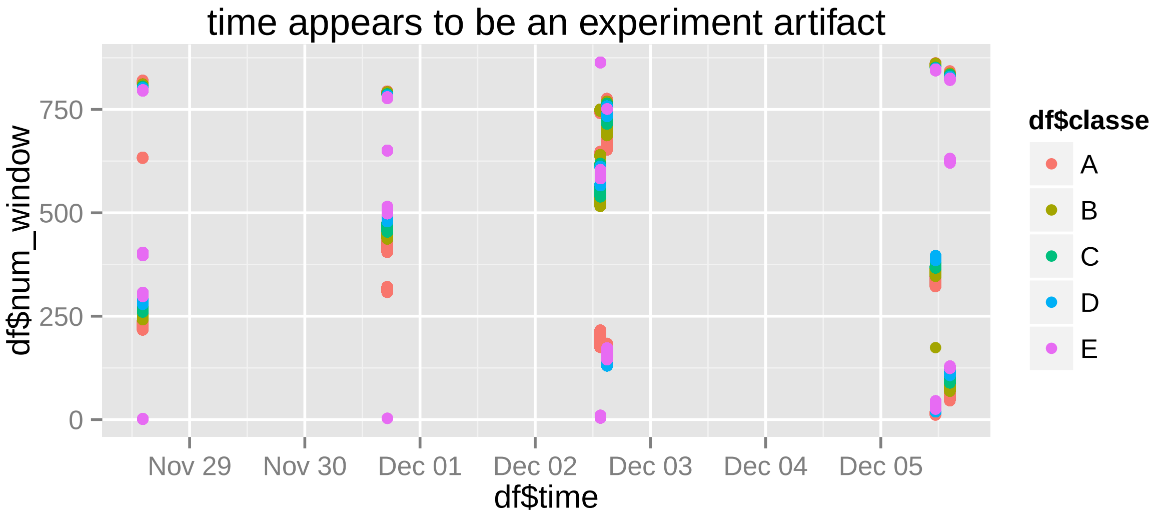
```
##     X user_name                time new_window num_window
## 1   4    adelmo 2011-12-02 13:33:00         no        194
## 2   9  carlitos 2011-12-05 11:24:00         no        323
## 3  11  carlitos 2011-12-05 11:24:00         no        859
## 4  18  carlitos 2011-12-05 11:24:00         no        361
## 5  10   charles 2011-12-02 14:57:00         no        664
## 6   5    eurico 2011-11-28 14:13:00         no        235
## 7  13    eurico 2011-11-28 14:14:00         no        257
## 8  16    eurico 2011-11-28 14:15:00         no        302
## 9  20    eurico 2011-11-28 14:14:00         no        255
## 10  2    jeremy 2011-11-30 17:11:00         no        431
## 11  3    jeremy 2011-11-30 17:11:00         no        439
## 12  6    jeremy 2011-11-30 17:12:00         no        504
## 13  7    jeremy 2011-11-30 17:12:00         no        485
## 14  8    jeremy 2011-11-30 17:11:00         no        440
## 15 12    jeremy 2011-11-30 17:11:00         no        461
## 16 14    jeremy 2011-11-30 17:10:00         no        408
## 17 15    jeremy 2011-11-30 17:12:00         no        779
## 18  1     pedro 2011-12-05 14:23:00         no         74
## 19 17     pedro 2011-12-05 14:22:00         no         48
## 20 19     pedro 2011-12-05 14:23:00         no         72
```

We were able to eliminate `new_window` (no variation) as well as the variable `X` (a simple row count variable).

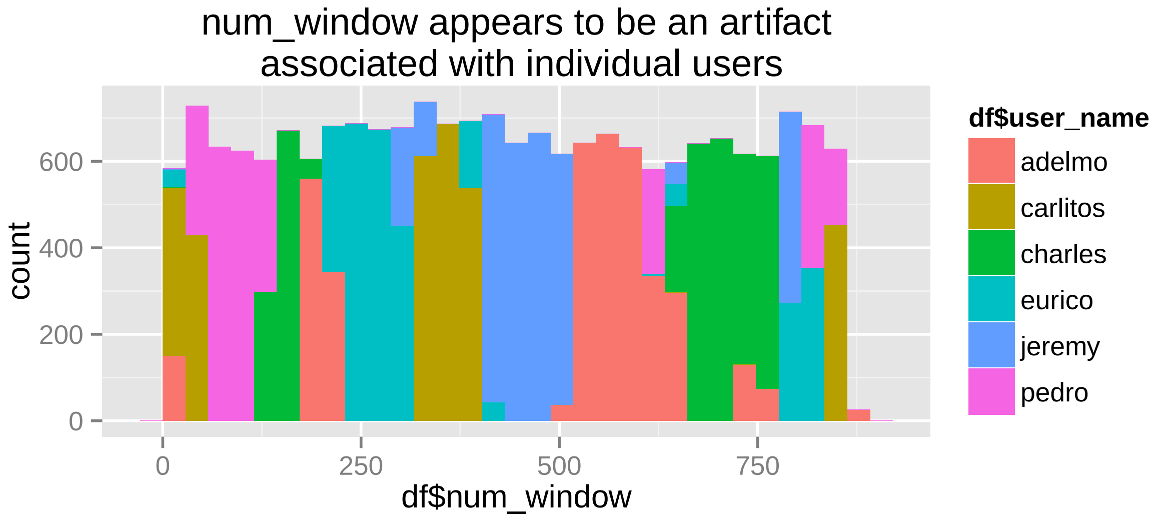
Timestamp and Window Data

Next we are interested in looking at the timestamp fields as well as the `num_window` variable.

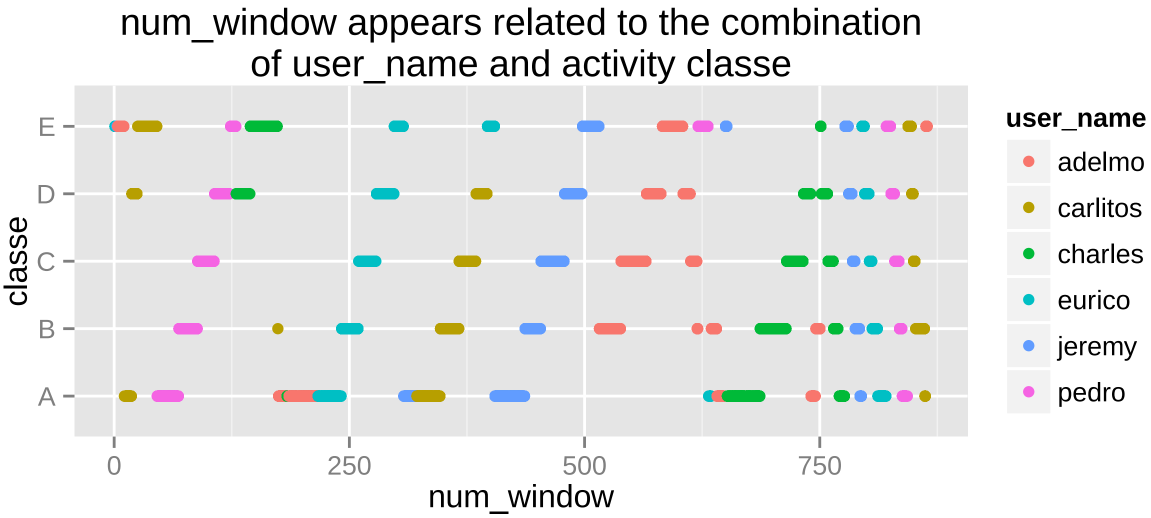
```
qplot(df$time, df$num_window, color=df$classe,
      main="time appears to be an experiment artifact")
```



```
#qplot(df$num_window, fill=df$classe, main="num_window appears distributed across classes")
qplot(df$num_window, fill=df$user_name, main="num_window appears to be an artifact\n associated with ind
```



```
qplot(num_window, classe, color=user_name, data=df, main="num_window appears related to the combination
```



Our first find was that the timestamp fields correspond to when the researchers conducted the weight lifting experiments. They should not be used in prediction. On the other hand, we found the `num_window` feature to be interesting. Even though the variable seems to be a data collection artifact and not likely to be a result of user activity, in the last plot above its value seems to be related to the classe on a per-user_name basis. We will therefore explore models both with and without this feature.

Predictor Variables

Next, we looked at the remaining columns corresponding acceleration and gyro data taken at different body positions. Without more domain specific knowledge (in the weigh lifting exercise domain), we chose to focus on eliminating inconsistent or low quality data fields. We found them by isolating the large number fields which contained NA's in the testing dataset.

We isolated columns which contained purely NA in the testing set.

```
bad_cols <- which(apply(is.na(df_submit), 2, sum)/nrow(df_submit) == 1)
```

We found 100 columns we identified as unusable due to high sparsity of data (100% in the testing data set), and we confirmed their unusability by finding that all of these same columns have greater than 96% missing data in the training dataset.

```
c(num_bad_in_testing=length(bad_cols),
     num_bad_in_training=sum(apply(is.na(df[,bad_cols]), 2, sum)/nrow(df) > 0.96))
```

```
##  num_bad_in_testing num_bad_in_training
##                 100                 100
```

```
#The following plot confirms the few data points which are valid do not strongly predict
#the "classe". This plot was omitted from the output for brevity.
#qplot(df[,bad_cols[1]], df[,"classe"])
```

Modeling with Cross Validation

Now that we have identified a solid set of features, we're ready to begin modeling.

Training and Testing Sets

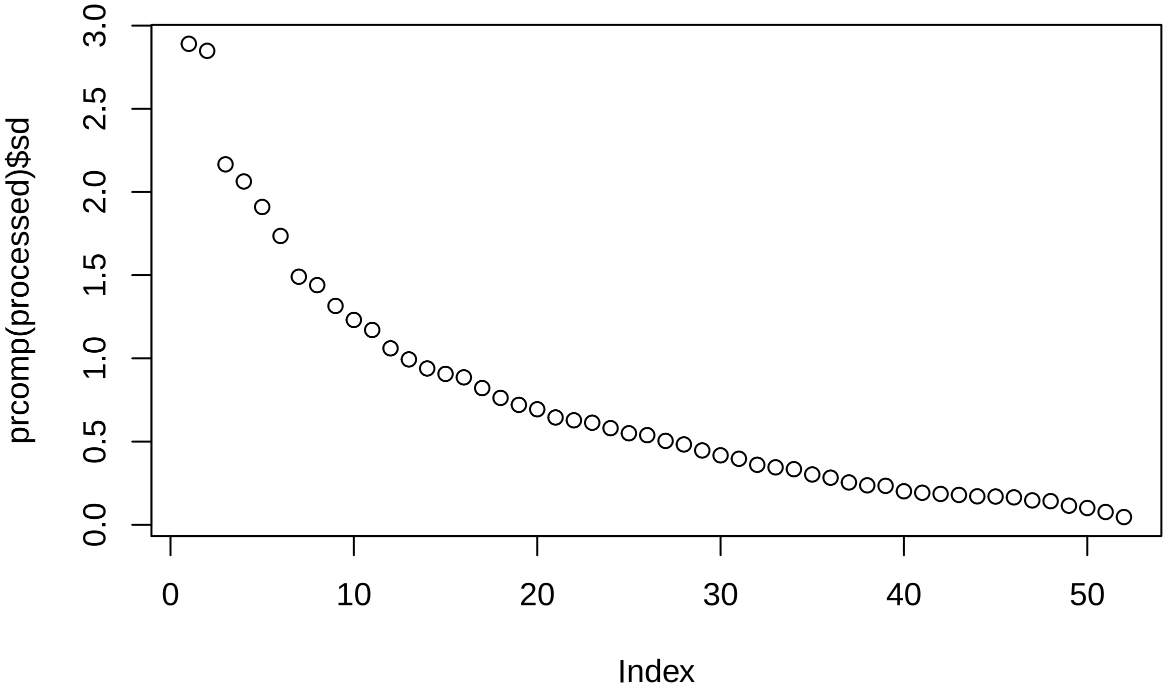
```
set.seed(1234)
inTrain <- createDataPartition(y=df$classe, p=0.80,list=FALSE)
training <- df[inTrain, -bad_cols]
# testing set will be used ONE TIME after model selection and parameter
# tuning are completed.
testing <- df[-inTrain, -bad_cols]

#the sensor columns
sensor_cols <- c(8:59)
```

We used an 80% and 20% split for training and testing sets. The testing set we refer to here was split off from the training data provided. *It will be set aside and used one time to predict our out-of-sample data rate.* We used a fixed seed in order to avoid accidentally training with the testing set when the partition is randomly re-created.

Principal Components Analysis

```
prepro_obj <- preProcess(training[,sensor_cols], method=c("center","scale"))
processed <- predict(prepro_obj, training[,sensor_cols])
plot(prcomp(processed)$sd)
```



After we scaled and centered the data, we looked at the plot of the standard deviations of its principal components. A lack of sharp drop off in variability suggests performing PCA may be of little use to our models. We will run an actual model with PCA to see if this hunch is correct.

The Models

We take advantage of the paralles package to use multiple cores to speed up modeling. Through experimentation, we found using one less than the number of detected cores improves system responsiveness during the modeling process. It was necessary to restart the clusters prior to each modeling run in order to reclaim unused memory.

```
restartClusters <- function() {
  if (exists("cl",mode="list")) try(stopCluster(cl))
  # leaving one core out improves system responsiveness
  cl <<- makeCluster(detectCores()-1)
  registerDoParallel(cl)
}
```

Generalized Boosted Models

The first set of models we run are Generalized Boosted Models. We run GBM under different options such as bootstrap versus 10-fold cross-validation, no preprocessing vs center and scaling vs Principal Component Analysis.

```
restartClusters()
proc1 <- system.time(mod1 <- train(training[,sensor_cols], training[,"classe"],
                                   method="gbm"))
```

```
cvControl <- trainControl(## 10-fold CV
  method = "repeatedcv",
  number = 10,
  ## repeated ten times
  repeats = 1)
restartClusters()
proc2 <- system.time(mod2 <- train(training[,sensor_cols], training[,"classe"],
                                   method="gbm", trControl = cvControl))
```

```
restartClusters()
proc3 <- system.time(mod3 <- train(training[,c(2,sensor_cols)], training[,"classe"],
                                   method="gbm", trControl = cvControl))
```

```
restartClusters()
proc4 <- system.time(mod4 <- train(training[,sensor_cols], training[,"classe"],
                                   preProcess=c("center","scale"),
                                   method="gbm", trControl = cvControl))
```

```
restartClusters()
proc5 <- system.time(mod5 <- train(training[,sensor_cols], training[,"classe"],
                                   preProcess=c("pca"),
                                   method="gbm", trControl = cvControl))
```

Random Forrests

Next we run Random Forrest models comparing boostrap vs. 10-fold cross validation, as well as introducing user_name and and num_window variables as predictors.

```
restartClusters()
proc6 <- system.time(mod6 <- train(training[,sensor_cols], training[,"classe"],
                                      method="rf"))
```

```
restartClusters()
proc7 <- system.time(mod7 <- train(training[,sensor_cols], training[,"classe"],
                                      method="rf", trControl = cvControl))
```

```
restartClusters()
proc8 <- system.time(mod8 <- train(training[,c(2,sensor_cols)], training[,"classe"],
                                      method="rf"))
```

```
restartClusters()
proc9 <- system.time(mod9 <- train(training[,c(2,7,sensor_cols)], training[,"classe"],
                                      method="rf"))
```

```
restartClusters()
proc10 <- system.time(mod10 <- train(training[,c(2,7,sensor_cols)], training[,"classe"],
                                      method="rf", trControl = cvControl))
```

```
models <- NULL

models <- rbind(models, list(method="gbm", CV="bootstrap",
                             time_used=proc1["elapsed"],
                             notes="", model=mod1))
models <- rbind(models, list(method="gbm", CV="10-fold",
                             time_used=proc2["elapsed"],
                             notes="", model=mod2))
models <- rbind(models, list(method="gbm", CV="10-fold",
                             time_used=proc3["elapsed"],
                             notes="user_name", model=mod3))
models <- rbind(models, list(method="gbm", CV="10-fold",
                             time_used=proc4["elapsed"],
                             notes="center,scale", model=mod4))
models <- rbind(models, list(method="gbm", CV="10-fold",
                             time_used=proc5["elapsed"],
                             notes="PCA", model=mod5))

models <- rbind(models, list(method="rf", CV="bootstrap",
                             time_used=proc6["elapsed"],
                             notes="", model=mod6))
models <- rbind(models, list(method="rf", CV="10-fold",
                             time_used=proc7["elapsed"],
                             notes="", model=mod7))
models <- rbind(models, list(method="rf", CV="bootstrap",
                             time_used=proc8["elapsed"],
                             notes="user_name", model=mod8))
models <- rbind(models, list(method="rf", CV="bootstrap",
                               time_used=proc9["elapsed"],
                               notes="user_name,num_window", model=mod9))
models <- rbind(models, list(method="rf", CV="10-fold",
                               time_used=proc10["elapsed"],
                               notes="user_name,num_window", model=mod10))

res <- t(sapply(models[1:5,"model"], function(x) tail(x[["results"]],1)[c("Accuracy","AccuracySD")]))

res <- rbind(res, t(sapply(models[6:10,"model"], function(x) {
with(x, filter(results, mtry==bestTune[1,1])[c("Accuracy","AccuracySD")])
})))
```

```
library(xtable)
tab <- xtable(cbind(models[,-ncol(models)], res), digits=4)
#print(tab, type="html", html.table.attributes="border=0 cellpadding=3")
#print(tab, type="xtable")
tab
```

% latex table generated in R 3.1.1 by xtable 1.7-4 package % Sun Oct 26 17:52:36 2014

	method	CV	time_used	notes	Accuracy	AccuracySD
1	gbm	bootstrap	346.0600		0.9583	0.0028
2	gbm	10-fold	156.5400		0.9625	0.0046
3	gbm	10-fold	163.3300	user_name	0.9639	0.0058
4	gbm	10-fold	158.8900	center,scale	0.9611	0.0041
5	gbm	10-fold	109.1900	PCA	0.8236	0.0143
6	rf	bootstrap	1033.9200		0.9903	0.0011
7	rf	10-fold	398.5400		0.9941	0.0029
8	rf	bootstrap	1018.6700	user_name	0.9905	0.0013
9	rf	bootstrap	1020.0700	user_name,num_window	0.9969	0.0007
10	rf	10-fold	405.3600	user_name,num_window	0.9984	0.0012

Results

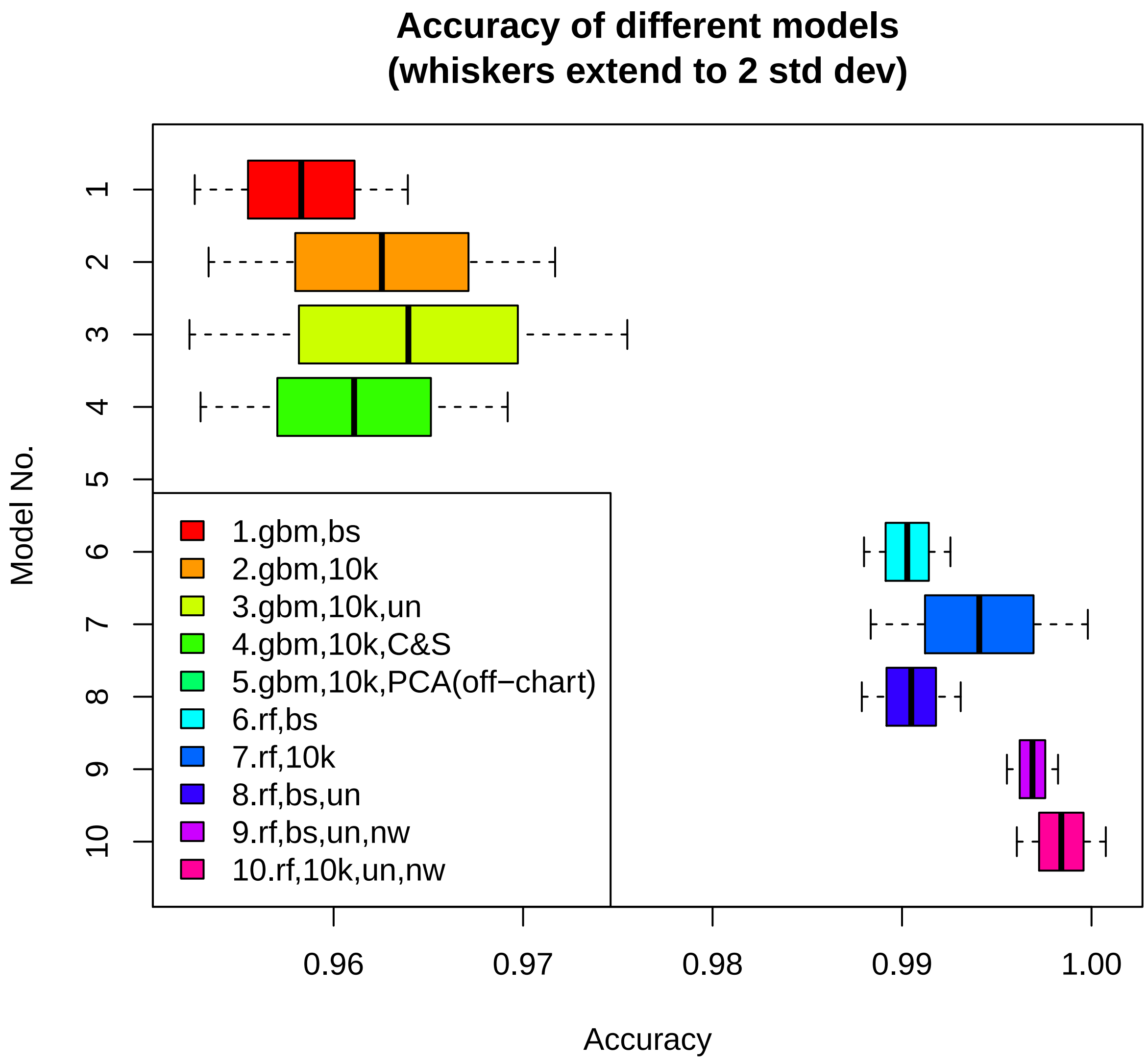
The previous table shows our progression toward ever better models. First we started with GBM and obtained accuracies of about 96%. We found that 10-fold cross-validation offered a small improvement over bootstrapping, but increased the variance. Adding in the user_name feature or centering and scaling the predictor variables made little impact on the accuracy. On the other hand, Principal Component Analysis greatly reduced the accuracy down to 82%. This result reflected our earlier analysis that the PCA vectors were not dropping sufficiently in their standard deviations.

Our best results came from using Random Forest algorithms, shooting up to an accuracy of 99% using bootstrapping, creeping up slightly under 10-fold cross-validation but at cost of higher variance. Finally, we saw that *introducing the combination of user_name and num_window noticeably increased the accuracy up to 99.7%*. This confirmed our earlier analysis that a relationship seemed to exist.

Boxplots are drawn where the whiskers correspond to 2 standard deviations of the accuracy across different cross-validation tests within each model.

```
acc <- unlist(res)
dim(acc) <- dim(res)
bpdat <- matrix(c(acc[,1]-2*acc[,2], acc[,1], acc[,1]+2*acc[,2]),
       3, nrow(acc), byrow=TRUE)
bpdat[,5] <- NA # the value for model no. 5 was not drawn for being too low
cols <- rainbow(10)
boxplot(
  bpdat,
  at=seq(ncol(bpdat),1,-1),
  col=cols,
  horizontal=TRUE,
  xlab="Accuracy",
  ylab="Model No.",
  main="Accuracy of different models\n(whiskers extend to 2 std dev)")

legend("bottomleft", c(
  "1.gbm,bs",
  "2.gbm,10k",
  "3.gbm,10k,un",
  "4.gbm,10k,C&S",
  "5.gbm,10k,PCA(off-chart)",
  "6.rf,bs",
  "7.rf,10k",
  "8.rf,bs,un",
  "9.rf,bs,un,nw",
  "10.rf,10k,un,nw"),
  fill=cols
)
```



The model we chose to use was Random Forest, boostrap, with user_name and num_window features included. These parameters gave us the best combination of accuracy and low variance on the cross-validated training dataset. Next, using this model, we predict the outcome classes of the testing dataset which we separated from the training dataset previously.

```
pred <- predict(mod9, testing[,c(2,7,sensor_cols)])
table(testing[,"classe"], pred) -> tab
tab
```

```
##    pred
##         A    B    C    D    E
##   A 1116    0    0    0    0
##   B    3  755    1    0    0
##   C    0    2  682    0    0
##   D    0    0    0  643    0
##   E    0    2    0    0  719
```

The table above counts the number of actual classes vs. the predicted classes. The number of correctly predicted classes is given in the diagonals. We find that only a few testing cases were mispredicted.

```
sum(diag(tab)) / nrow(testing) -> accuracy
```

The accuracy on the testing dataset is 99.7961%. This is the out-of-sample error we expect on unseen data. It is quite close to the cross-validated accuracy we saw on the training set of 99.6884%

Submission

The following results are submitted for the graded testing set of 20 observations.

```
ans <- predict(mod9, df_submit[,-bad_cols][,c(2,7,sensor_cols)])
ans
```

```
##  [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
```

```
pml_write_files(ans)
```