Activity Classification

Executive Summary

A randomForest model, tuned by the {caret package} yields an accuracy close to that of the published boosting model – 99.4% – albeit with a greater number of predictors.

Getting & Cleaning the Data

Dowloading the Data

```r
fileurl <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
filename <- "pml-training.csv"

downloadData <- function() {
        if (!file.exists(filename)) {
                download.file(fileurl, dest=filename, method="curl")
        }
        data <- read.csv("pml-training.csv", header = TRUE)
        data
}

activity <- downloadData()
```

Cleaning the Data

Once downloaded, I did some minor cleaning of the data:

I removed the summary variables from the data set. These variables are labeled "avg" and "sd" and, as functions of other variables, they contain only a few hundred values with the rest being "NAs."

The `read.csv` function coded a number of variables that I believe should be numeric as factors. These variables had hundreds of levels. I re-coded them as numeric.

```r
## remove user, dates/times
activity.m <- activity[, -c(1:7)]

## remove summary variables (avg & sd)
navalues <- is.na(activity.m[1, ])
activity.mi <- activity.m[, !navalues]

## change 2, 200-300 level factor variables to numeric
for(i in 1:dim(activity.mi)[2]-1) {
        if(is.factor(activity.mi[, i])==TRUE) {
                activity.mi[, i] <- as.numeric(activity.mi[, i])
        }
}

activity.min <- activity.mi
```

Classification Model

Partitioning the Data

The first step was to partition the data set into training and testing sets. I chose to randomly select 70% of the data to train the model.

```
library(caret)

set.seed(3244)
inTrain <- createDataPartition(activity.min$classe, p=0.7, list=FALSE)

training <- activity.min[inTrain, ]
testing <- activity.min[-inTrain, ]
```

Identifying Near-Zero Variance Predictors

To limit the number of ineffective variables run through the classification algorithms, I used the `nearZeroVar()` function in the `{caret package}`. Twenty-four variables were determined to have near-zero variance by the function. These were removed from the training set.

```
## near zero variance vars
nsv <- nearZeroVar(training, saveMetrics=TRUE)

nzvar <- nsv$nzv
training <- training[, !nzvar]
```

Random Forest Model

I evaluated two types of models: a Random Forest Model and a Gradient Boosting Model. Although the paper[1] that reports these same data uses an Ada Boost variety, I found that random forest initially performed better than the gbm model. That is not to say that a boosting algorithm, properly tuned, would not ultimately out-perform the rF. But the rF model appeared to require less tuning to achieve results close to those of Ugulino, et. al. Here, I present only my procedure for fitting the Random Forest Model.

Backwards Feature Selection I ran the `rfe` backwards feature selection function to find an efficient number of variables from which to build a final model (and so as not to overfit the model to the training set). My control was five-fold cross-validation, repeated three times. The function evaluated the variables based on accuracy and kappa values, measuring the best 5, 10, 15, 20, 25, and 30-variable random forest models.

```
library(doMC)

y <- training$classe
x <- training[, -dim(training)[2]]

normalization <- preProcess(x)
x <- predict(normalization, x)
x <- as.data.frame(x)

subsets <- seq(5, 30, by=5)
```

```r
ctrl <- rfeControl(functions = rfFuncs,
                   method = "repeatedcv",
                   number = 5,
                   repeats = 3,
                   verbose = FALSE)

registerDoMC(cores = 4)

set.seed(20)
rfProfile <- rfe(x, y,
                 sizes = subsets,
                 rfeControl = ctrl)

best <- predictors(rfProfile)
rfProfile
```

```
##
## Recursive feature selection
##
## Outer resampling method: Cross-Validated (5 fold, repeated 3 times)
##
## Resampling performance over subset size:
##
##  Variables Accuracy Kappa AccuracySD KappaSD Selected
##          5    0.962 0.952    0.00315 0.00400
##         10    0.983 0.979    0.00339 0.00429
##         15    0.987 0.984    0.00257 0.00325
##         20    0.990 0.987    0.00220 0.00278
##         25    0.990 0.988    0.00205 0.00259
##         30    0.991 0.989    0.00207 0.00262
##         52    0.993 0.991    0.00172 0.00218        *
##
## The top 5 variables (out of 52):
##    roll_belt, yaw_belt, magnet_dumbbell_z, pitch_belt, magnet_dumbbell_y
```

The results show that a ten-variable model yields an in-sample accuracy above 98%. The accuracy increases at a diminishing rate. Compared to a 30-variable model, including all 52 variables increases the accuracy minimally. The following plot shows the increase in the model accuracy as the number of variables is increased.

The most important variables are roll_belt, yaw_belt, magnet_dumbbell_z, pitch_belt, and magnet_dumbbell_y.

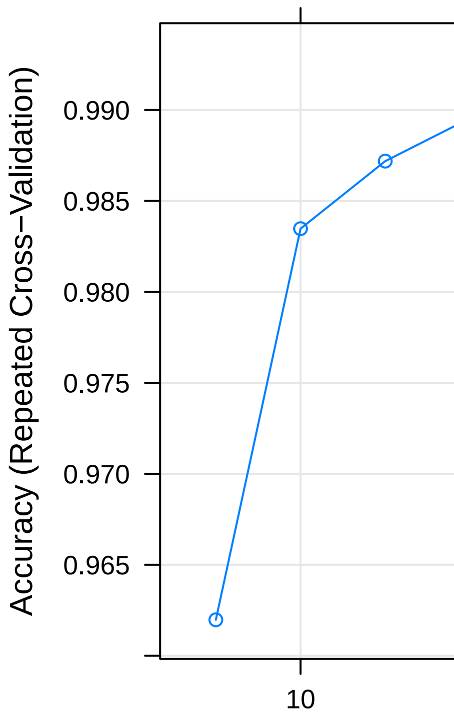

Fig. 1. Random Forest Accuracy vs. Number of Feature-Selected Variables

rF Model Fit I decided to train the final rF model on a selection of thirty variables, chosen according to the `rfe()` results.

```
## match indices from var "best" to training vars
matches <- as.integer()
indices <- function(n) {
        ind <- best[1:n]
        for(i in 1:n) {
                mat <- match(ind[i], names(training))
                matches <- c(matches, mat)
                }
        matches
        }

indexes <- indices(30)
training.min <- training[, c(indexes, dim(training)[2])]
```

The rF model was built with five-fold cross-validation, repeated three times. I set the model to 500 trees per iteration, and again, the predictors were normalized.

```
## 5-fold cv, repeated twice
set.seed(1231)
ctrl.rf <- trainControl(method = "repeatedcv",
                     number = 5,
                     repeats = 3)

registerDoMC(cores = 4)
```

```
## fit model "rf"
rf <- train(training.min$classe ~ .,
             data = training.min,
             method = "rf",
             preProcess = c("center", "scale"),
             ntree=500,
             trControl = ctrl.rf)
```

The default "m" parameter for random forest classification models is sqrt(p). Here, the best "m" splitting parameter ("mtry") as determined by the {caret package} is:

```
##   mtry
## 1    2
```

Evaluating the Model

The rF model was applied to predict the values of the "classe" variable of the out-of-sample testing set.

Table 2. rF Predictions on Testing Set

```
##
## pred    A    B    C    D    E
##    A 1672    7    0    0    0
##    B    2 1124   13    0    0
##    C    0    8 1008   10    2
##    D    0    0    5  953    5
##    E    0    0    0    1 1075
```

The out-of-sample accuracy is:

```
## [1] 0.991
```

The misclassification rate is:

```
## [1] 0.009006
```

Conclusions

A random forest model, tuned by the {caret package} yields an accuracy close to that of the published boosting model – 99.4% – albeit with a greater number of predictors.

[1] Ugulino, W.; Cardador, D.; Vega, K.; Velloso, E.; Milidiu, R.; Fuks, H. "Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements." *Proceedings of 21st Brazilian Symposium on Artificial Intelligence. Advances in Artificial Intelligence - SBIA 2012.* In: Lecture Notes in Computer Science. , pp. 52-61. Curitiba, PR: Springer Berlin / Heidelberg, 2012. ISBN 978-3-642-34458-9. DOI: 10.1007/978-3-642-34459-6_6.