Coursera Practical Machine Learning Project

Mark S.

Wednesday, July 22, 2015

Project Summary

The goal of this project is to build a model that can predict the manner in which an exercise is being performed. We are given two sets of data, a training set and a test set. The source and description of the data is found here: http://groupware.les.inf.puc-rio.br/har. The data can be downloaded via the following links:

Training: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv

Testing: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

Data Overview and Strategy

My interpretation of the problem is as follows: "Given a set of measurements *in this moment*, predict how the exercise is being performed by any person *in this moment*."

The "*in this moment*" phrase is emphasized to establish the fact that we want a model that can generalize to any moment in time. We don't want to make a prediction based off of measurements that occured in the past; rather we would like to make predictions based on single measurements occuring at a single moment in time. Also, we would like a model that can generalize to any user. With this reasoning, we will ignore any variables provided that are considered "window statistics", variables involving time, and the names of the participants. This includes the first 7 variables of the data set, as well as any variable name containing min, max, avg, var, std, total, kurtosis, skewness, or amplitude.

Since this is a classification task, we'll use a random forest model. The entire process will be structured as follows:

1. Remove unwanted variables (explained above)
2. Split training data into 2 sets; call them "A" and "B" with a 70/30 split.
3. Perform data analysis and preprocessing using data set "A"
4. Perform feature selection using cross validation.
5. Build a random forest with selected features; evaluate its out of sample error with the hold-out set "B"
6. Make a prediction with the "final test" set; the data we must predict for submission.

Downloading and Cleaning the Data

First we download the data, and remove the unwanted columns, as explained previously:

```
#Set global seed for reproducibility
set.seed(8080)

#Load the data; use library RCurl to allow download of data in knitr
#note that some of the data contain "NA", "#DIV/0!" and empty values. These will be considered "NA"
temp<-getURL("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv",ssl.verifypeer=0L,f
data<- tbl_df(read.csv(text=temp, na.strings = c("NA","#DIV/0!","")))

temp<-getURL("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv",ssl.verifypeer=0L,fo
```

```
final.test<- tbl_df(read.csv(text=temp, na.strings = c("NA","#DIV/0!","")))

#remove temp variable
rm(temp)

#First 7 columns aren't measurements; they won't be included
data<- data[,-(1:7)]
final.test<- final.test[,-(1:7)]

#Create a vector of all feature names based on time or window statistics
feats<- grep("^min|max|avg|var|std|total|kurtosis|skewness|amplitude",names(data))

#Remove these features from the data
data<- data[,-feats]
final.test<-final.test[,-feats]
```

Now we split the data into train and test sets with a 70/30 split. The "final.test" set is the one we will make predictions on and submit.

```
#split data into train/test sets
n<- createDataPartition(y=data$classe,p=0.7,list=FALSE)
train<- data[n,]
test<- data[-n,]

#convert data to tbl_df to use with dplyr
train<- tbl_df(train)
test<- tbl_df(test)
```

Exploratory Data Analysis

First check the dimensionality of the data, and get an idea of the distribution of the classe variable:

```
dim(train)
```

```
## [1] 13737    49
```

```
table(train$classe)
```

```
##
##    A    B    C    D    E
## 3906 2658 2396 2252 2525
```

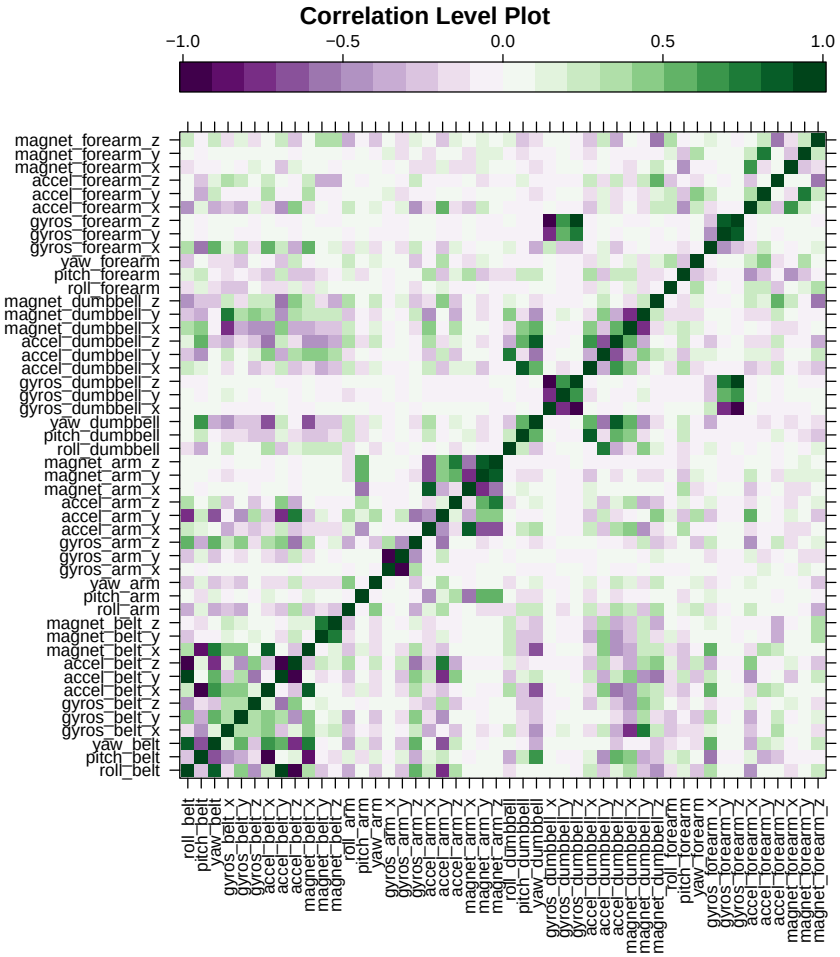
The classe variable isn't perfectly balanced, but it shouldn't be enough to cause any problems.

We've already eliminated many of the features (reducing from 159 to 48). We should also check to see if we have any features with near-zero variance:

```
nearZeroVar(train)
```

```
## integer(0)
```

It would be difficult to plot feature pairs, but we can visualze the correlations to identify variables to inspect:

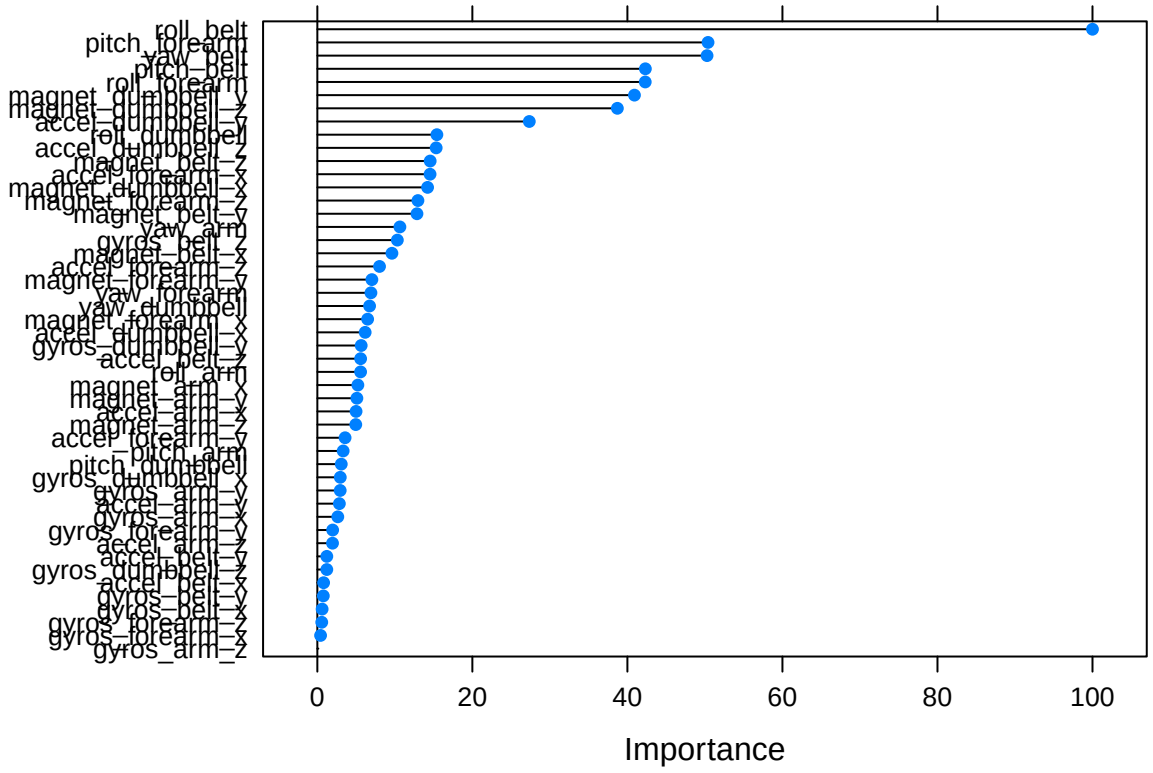


Based on the correlation plot it seems that only a handful of variables are highly correlated. We'll build a random forest using 10-fold cross validation to determine the most important features to use in the final model:

```
fitControl<- trainControl(method="cv",number=10)
mod0<- train(classe~.,data=train, trControl= fitControl, method="rf", ntree = 20, allowParallel=TRUE)
```

We can plot the important features to see which ones we should keep in our final model:

```
plot(varImp(mod0))
```



It appears we could get away using the top 10 to 15 features. We'll err on the safe side and keep the top 15 variables for the final model. We'll extract these features and prepare the data for the final model:

```
#Feature Importances
names <- rownames(varImp(mod0)$importance)
importance <- varImp(mod0)$importance$Overall
imps<- tbl_df(data.frame(names,importance))

#Custom function to Pull out column as vector from dplyr data frame
#Usage: pull(data,"column_name") or pull(data, column_name) or pull(data,column_number)
pull <- function(x,y) {x[,if(is.name(substitute(y))) deparse(substitute(y)) else y, drop = FALSE][[1]]}

#Extract important feature names, use it to only keep important variables in data
imp.feats<- as.character(pull(arrange(imps,-importance),names))[1:15]
imp.train<- train[,c(imp.feats,"classe")]
imp.test<- test[,c(imp.feats,"classe")]
```

Now we'll build our final random forest model.

Building the Model

First, we build the model:

```
#Build the model; number of trees was chosen to reduce the time to build the model
mod1<- train(classe~., data = imp.train, method = "rf",ntree= 20, allowParallel=TRUE)
```

Now, test the model on the test set to get an estimate of the out-of-sample error:

```
#Make predictions on the test set
pred1<- predict(mod1,imp.test[,-16])

#Print the confusion matrix for the model
confusionMatrix(pred1,imp.test$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1670   15    0    0    0
##          B    2 1104   14    4    2
##          C    2   19 1003    9    3
##          D    0    0    9  950    3
##          E    0    1    0    1 1074
##
## Overall Statistics
##
##                Accuracy : 0.9857
##                  95% CI : (0.9824, 0.9886)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9819
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9976   0.9693   0.9776   0.9855   0.9926
## Specificity            0.9964   0.9954   0.9932   0.9976   0.9996
## Pos Pred Value         0.9911   0.9805   0.9681   0.9875   0.9981
## Neg Pred Value         0.9990   0.9926   0.9953   0.9972   0.9983
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2838   0.1876   0.1704   0.1614   0.1825
## Detection Prevalence   0.2863   0.1913   0.1760   0.1635   0.1828
## Balanced Accuracy      0.9970   0.9823   0.9854   0.9915   0.9961
```

The has an out-of-sample error estimate of approximately 1.4%. This model will be used to predict the final test set:

```
predict(mod1,final.test[,-16])
```

```
##  [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
```