Practical Machine Learning - Project Writeup

Introduction

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is now possible to collect a large amount of data about personal activity relatively inexpensively. These type of devices are part of the quantified self movement - a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks. 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 project, your goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. More information is available from the website here: http://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

Environment Setup

```
library(caret)
```

```
## Loading required package: lattice
## Loading required package: ggplot2
```

```
library(ggplot2)
library(doParallel)
```

```
## Loading required package: foreach
## Loading required package: iterators
## Loading required package: parallel
```

```
setwd("~/../Desktop/Coursera -Data Science Track/Practical Machine Learning//Project")

#function to write an individual file for each problem id
pml_write_files = function(x){
  n = length(x)
  for(i in 1:n){
    filename = paste0("problem_id_",i,".txt")
    write.table(x[i],file=filename,quote=FALSE,row.names=FALSE,col.names=FALSE)
  }
}
```

Data Preparation

The training set will be read in and it will be split 60% for training and 40% for testing. Variables that mainly (> 90% of the time) contain empty strings or NA values will be excluded from use in the model. Additionally the "classe", "user_name" and "new_window" variables will be turned into factor variables. All other variables will be used in the model building process, as the algorithm that will be used, Gradient Boosting Machines, is robust enough to ignore variables that are not useful.

```
pml_train <- read.csv("pml-training.csv"
                      ,stringsAsFactors=FALSE
                      )

#turn these variables into factors
pml_train$classe <- as.factor(pml_train$classe)
pml_train$user_name <- as.factor(pml_train$user_name)
pml_train$new_window <- as.factor(pml_train$new_window)

#use 60% of the data for training
train_size <- .6
train_x <- sample(x=pml_train$X,size = ceiling(nrow(pml_train) * train_size))
train <- pml_train$X %in% train_x

#create a testing set to validate against
test <- !(train)


#replace blank values with NA
replaceBlankWithNA <- function(x)
{
  ifelse(x == "",NA,x)
}

cols <- colnames(pml_train[train,])
for(i in 1:length(cols))
{
  if(class(pml_train[,cols[i]]) == "character")
  {
    pml_train[,cols[i]] <- replaceBlankWithNA(pml_train[,cols[i]])
  }
}


#do not include these variables as predictors
vars_to_exclude <- c('X'
                     ,'classe'
                     ,'raw_timestamp_part_1'
                     ,'raw_timestamp_part_2'
                     ,'cvtd_timestamp'
)

#find all the variables that contain NAs in them
nas <- sapply(pml_train,is.na)
nas_list <- colSums(nas)
nas_to_exclude <- names(nas_list[nas_list > 0])


predictors <- cols[!(cols %in% vars_to_exclude | cols %in% nas_to_exclude)]

#create a new data set containing all of the original observations, but only a subset of the variables
pml_train_subset <- pml_train[,c(predictors,'classe')]
```

```r
#clean up some of the objects from the environment
rm(nas)
rm(nas_list)
rm(pml_train)
```

Model Building

The model built will be a Gradient Boosting Machine (gbm). Various tuning parameters will be tried to achieve a good model. The best model of the canidate models will be choosen using 10 fold cross validation.

```r
#create a cluster to execute the training in parallel.
registerDoParallel(cores = 7)

## use 10-fold Cross Validation
fitControl <- trainControl(
  method = "cv",
  number = 10)

#set seed
set.seed(1234)

#define the different tuning parameters to try and compare for GBM
gbmGrid <-  expand.grid(interaction.depth = c(5,10,20,25),
                        n.trees = (1:50)*2,
                        shrinkage = c(0.1))

#use caret to build the GBM model, with the grid of different tuning parameters and CV.
m1 <- train(classe ~ ., data = pml_train_subset[train,c(predictors,'classe')]
            , trControl = fitControl
            , method = "gbm",tuneGrid = gbmGrid )
```

```
## Loading required package: gbm
## Loading required package: survival
## Loading required package: splines
##
## Attaching package: 'survival'
##
## The following object is masked from 'package:caret':
##
##     cluster
##
## Loaded gbm 2.1
## Loading required package: plyr

## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.5453
##      2        1.2704             nan     0.1000    0.3468
##      3        1.0556             nan     0.1000    0.2561
##      4        0.8957             nan     0.1000    0.2010
##      5        0.7710             nan     0.1000    0.1586
##      6        0.6730             nan     0.1000    0.1429
##      7        0.5848             nan     0.1000    0.1067
```

```
##       8         0.5169             nan     0.1000    0.0988
##       9         0.4560             nan     0.1000    0.0775
##      10         0.4061             nan     0.1000    0.0784
##      20         0.1298             nan     0.1000    0.0167
##      40         0.0295             nan     0.1000    0.0037
##      60         0.0089             nan     0.1000    0.0006
##      80         0.0031             nan     0.1000    0.0002
##     100         0.0011             nan     0.1000    0.0000
```

Model Evaluation

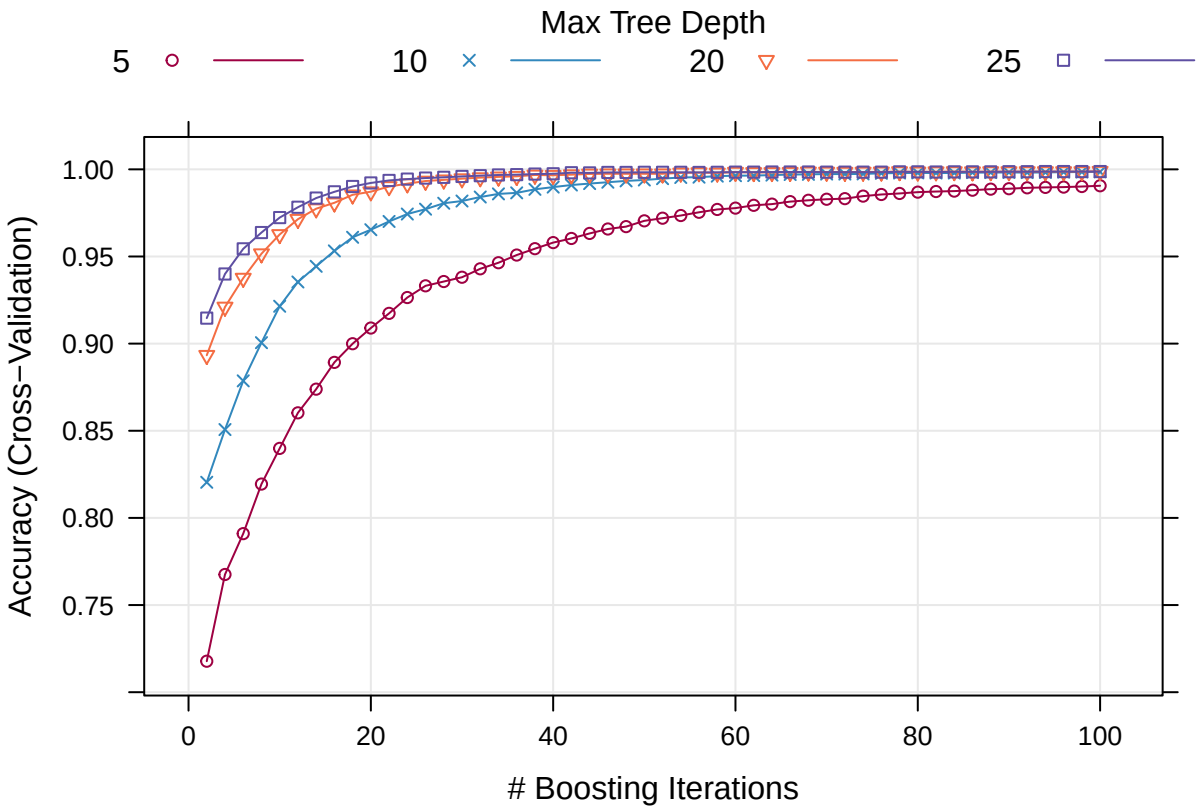
The final model has a max tree depth of: 20 and uses 100 number of trees. The Accuracy from 10 fold cross validation on this model is: .9988961. The accuracy from the cross validation should be very close to the actual accuracy on the test set.

```
#best model
m1$results[m1$results$n.trees==100 & m1$results$interaction.depth ==20,]$Accuracy
```

```
## [1] 0.9988961
```

The below chart shows how the accuracy changes while using different tuning parameters.

```
#plot the Accuracy from Cross Validation
trellis.par.set(caretTheme())
plot(m1)
```



As further validation of the accuracy the test set that was withheld from training is predicted using the final model, and you can see that the accuracy (0.9983435) on this test set is very close to the accuracy measured during cross validation (.9988961). This provides additional confidence that the model will perform well.

```
#predict on the test set that was split from the training set
test_predictions <- predict(m1, newdata = pml_train_subset[test,c(predictors)])

#Accuracy on the test set that was split from the training set
sum(ifelse(test_predictions == pml_train_subset[test,]$classe,1,0)) / length(test_predictions)
```

```
## [1] 0.9983435
```

Here's a confusion matrix of predicted values on the withheld test set.

```
#View a confusion matrix of the truth and predicted values
table(test_predictions,pml_train_subset[test,]$classe)
```

```
##
## test_predictions    A    B    C    D    E
##                A 2253    1    0    0    0
##                B    0 1535    0    0    0
##                C    0    0 1368   10    0
##                D    0    0    0 1263    2
##                E    0    0    0    0 1416
```

Predictions on Test Set

Finally the test set that was provided for the project will be predicted on using the final model. The predictions will be written out to individuals files.

```
#read in the provided test set
pml_test <- read.csv("pml-testing.csv",stringsAsFactors=FALSE)

#turn these variables into factors.
pml_test$user_name <- as.factor(pml_test$user_name)
pml_test$new_window <- as.factor(pml_test$new_window)

#predict the classes for the provided test set
answers <- predict(m1, newdata = pml_test[,c(predictors)])

#write out individuals files for the
pml_write_files(answers)

#turn off cluster for parallel execution
stopImplicitCluster()
```