Practical Machine Learning Project

Liangliang Su

March 31, 2018

Introduction

Background

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.

Goal

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. The goal of your project is to predict the manner in which they did the exercise. This is the “classe” variable in the training set. You may use any of the other variables to predict with.

Load, explore and clean the data

```
#library(lattice)
#library(ggplot2)
library(caret)
training<-read.csv(url("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"),header=TRUE)
testing<-read.csv(url("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"),header=TRUE)
print(dim(training))
```

```
## [1] 19622   160
```

```
print(dim(testing))
```

```
## [1]  20 160
```

```
#print(str(training))
#print(str(testing))
```

Some columns have missing values for most of rows. I will remove these kind of column. The first five columns contain people’s name and time information which is useless for prediction model. I will also remove these columns.

```
index<-which(colSums(is.na(training)|training=="")>0.5*dim(training)[1])
trainclean<-training[,-index]
trainclean<-trainclean[,-c(1:5)]
index<-colnames(trainclean)
colnames(testing)[colnames(testing)=="problem_id"] <- "classe"
testing<-testing[,index]
#print(index)
#print(colnames(testing))
```

After cleaning the training data and test data, I will split the training data into training part and test part.

```
set.seed(1111111)
inTrain<- createDataPartition(trainclean$classe, p=0.7, list=FALSE)
train<-trainclean[inTrain,]
test<-trainclean[-inTrain,]
print(dim(train))
```

```
## [1] 13737    55
```

```
print(dim(test))
```

```
## [1] 5885   55
```

Prediction model

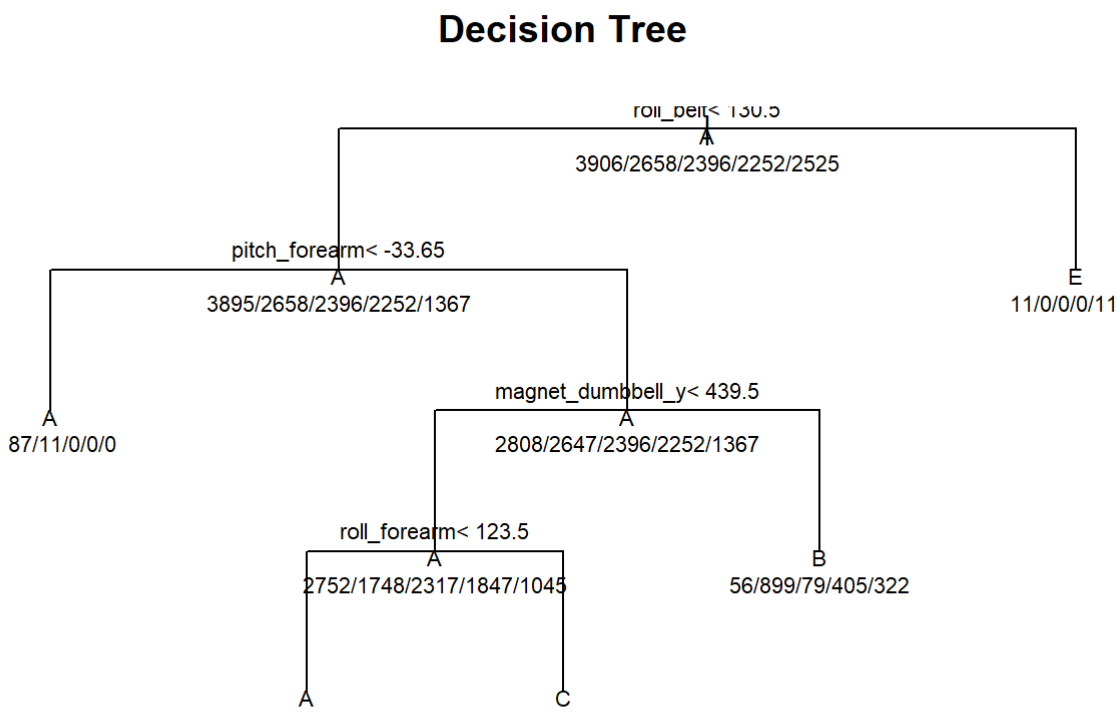
In this part, I will use 3 prediction models: decision trees, random forest ,gradient boosting method. I will use the cross-validation to limit overfitting. The train set will be splited into 5 folds.

Decision Trees

```
set.seed(54321)
control<-trainControl(method = "cv", number = 5)
DT<-train(classe ~ ., data = train, method = "rpart", trControl = control)
print(DT)
```

```
## CART
##
## 13737 samples
##    54 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10990, 10989, 10989, 10990, 10990
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.03763605  0.5359268  0.40183425
##   0.06031940  0.4428203  0.25361053
##   0.11667175  0.3323155  0.07306778
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.03763605.
```

```
plot(DT$finalModel,uniform=TRUE,main = 'Decision Tree')
text(DT$finalModel,use.n=TRUE,all=TRUE,cex=0.7)
```



fancyRpartPlot doesn't work for unknown reason. I use plot to represent the tree structure instead.

```
library(caret)
```

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

```
## Warning: package 'lattice' was built under R version 3.4.4
```

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

```
## Warning: package 'ggplot2' was built under R version 3.4.4
```

```
pred<- predict(DT,newdata=test)
confusionMatrix(pred, test$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1531  476  494  436  163
##          B   25  387   29  163  164
##          C  115  276  503  365  282
##          D    0    0    0    0    0
##          E    3    0    0    0  473
##
## Overall Statistics
##
##                Accuracy : 0.4918
##                  95% CI : (0.4789, 0.5046)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3351
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9146  0.33977  0.49025   0.0000  0.43715
## Specificity            0.6274  0.91972  0.78638   1.0000  0.99938
## Pos Pred Value         0.4939  0.50391  0.32641      NaN  0.99370
## Neg Pred Value         0.9487  0.85304  0.87960   0.8362  0.88741
## Prevalence             0.2845  0.19354  0.17434   0.1638  0.18386
## Detection Rate         0.2602  0.06576  0.08547   0.0000  0.08037
## Detection Prevalence   0.5268  0.13050  0.26185   0.0000  0.08088
## Balanced Accuracy      0.7710  0.62975  0.63831   0.5000  0.71826
```

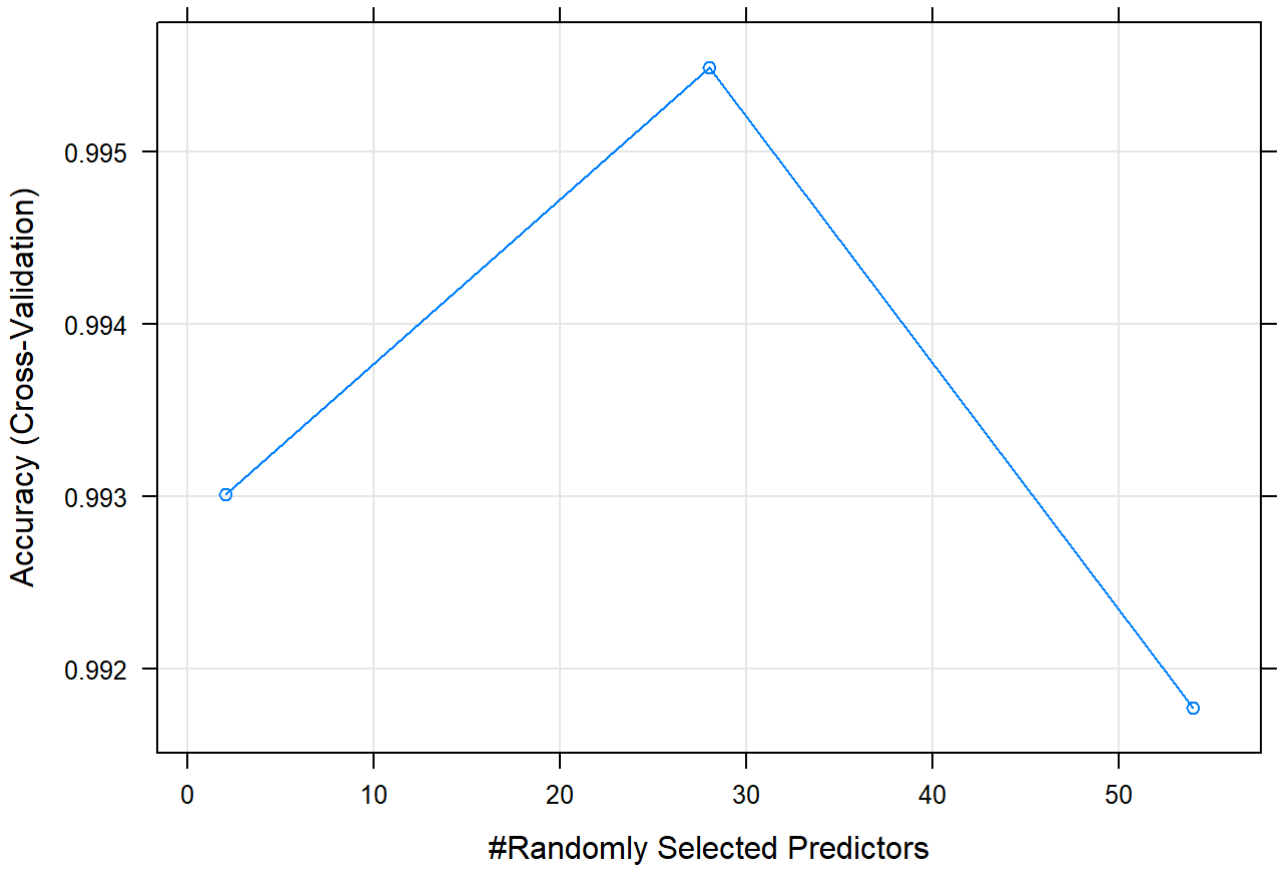
The accuracy of Decision Trees is about 49% which is a real poor performance.

Random Forest

```
RF<- train(classe~., data=train, method="rf", trControl=control, verbose=FALSE)
print(RF)
```

```
## Random Forest
##
## 13737 samples
##    54 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10989, 10988, 10990, 10991, 10990
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9930117  0.9911597
##   28    0.9954868  0.9942912
##   54    0.9917741  0.9895951
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 28.
```

```
plot(RF)
```




```
pred<- predict(RF,newdata=test)
confusionMatrix(pred, test$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1673    9    0    0    0
##          B    0 1130    0    0    0
##          C    0    0 1026    9    0
##          D    0    0    0  955    0
##          E    1    0    0    0 1082
##
## Overall Statistics
##
##                Accuracy : 0.9968
##                  95% CI : (0.995, 0.9981)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9959
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9994   0.9921   1.0000   0.9907   1.0000
## Specificity            0.9979   1.0000   0.9981   1.0000   0.9998
## Pos Pred Value         0.9946   1.0000   0.9913   1.0000   0.9991
## Neg Pred Value         0.9998   0.9981   1.0000   0.9982   1.0000
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2843   0.1920   0.1743   0.1623   0.1839
## Detection Prevalence   0.2858   0.1920   0.1759   0.1623   0.1840
## Balanced Accuracy      0.9986   0.9960   0.9991   0.9953   0.9999
```

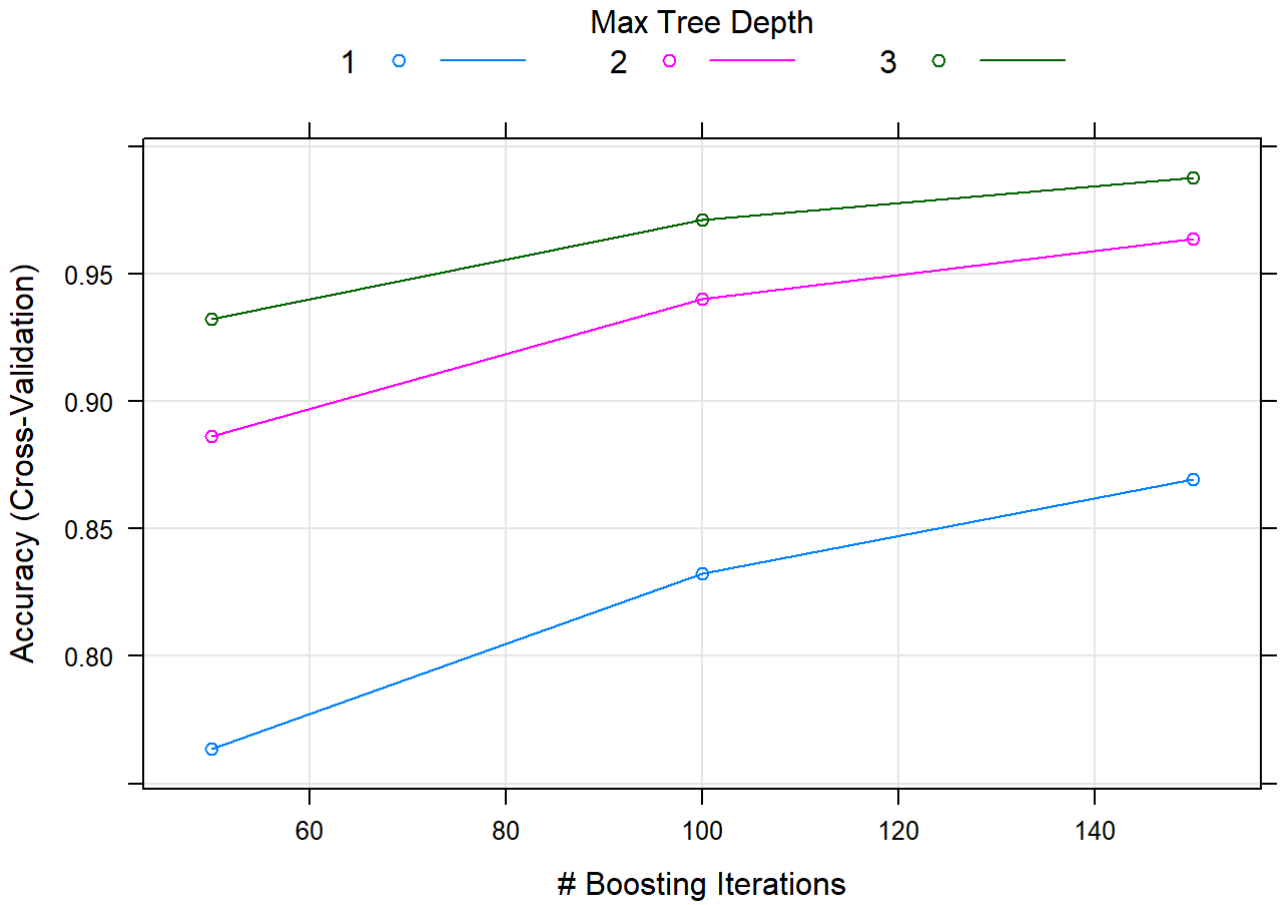
The accuracy of Random Forest is about 99.7% which is a really impressive performance.

Gradient Boosting Method

```
GB<-train(classe~., data=train, method="gbm", trControl=control, verbose=FALSE)
print(GB)
```

```
## Stochastic Gradient Boosting
##
## 13737 samples
##    54 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10990, 10991, 10990, 10988, 10989
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7634886  0.6998841
##   1                  100      0.8324964  0.7878593
##   1                  150      0.8694770  0.8348044
##   2                   50      0.8862935  0.8560160
##   2                  100      0.9399430  0.9240042
##   2                  150      0.9638198  0.9542239
##   3                   50      0.9321545  0.9141083
##   3                  100      0.9710270  0.9633449
##   3                  150      0.9875513  0.9842533
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150,
##  interaction.depth = 3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
plot(GB)
```




```
pred<- predict(GB,newdata=test)
confusionMatrix(pred, test$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1666    8    0    3    0
##          B    7 1118    5    5    3
##          C    0   11 1018   14    1
##          D    1    2    3  942    7
##          E    0    0    0    0 1071
##
## Overall Statistics
##
##                Accuracy : 0.9881
##                  95% CI : (0.985, 0.9907)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.985
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9952   0.9816   0.9922   0.9772   0.9898
## Specificity            0.9974   0.9958   0.9946   0.9974   1.0000
## Pos Pred Value         0.9934   0.9824   0.9751   0.9864   1.0000
## Neg Pred Value         0.9981   0.9956   0.9983   0.9955   0.9977
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2831   0.1900   0.1730   0.1601   0.1820
## Detection Prevalence   0.2850   0.1934   0.1774   0.1623   0.1820
## Balanced Accuracy      0.9963   0.9887   0.9934   0.9873   0.9949
```

The accuracy of Gradient Boosting Method is about 98.8% which is a also impressive performance.

Conclusion

Comparing with the three models above, the Random Forest has the best performance. Its accuracy is about 99.7%. The expected out-of-sample error rate is merely 0.3%. So I will use the Random Forest to predict the results.

```
pred<- predict(RF,newdata=testing)
print(pred)
```

```
##  [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
```