Practical Machine Learning Project

Jun Wen

September 26, 2016

Introduction

In the following study:

Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. Qualitative Activity Recognition of Weight Lifting Exercises. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human '13) . Stuttgart, Germany: ACM SIGCHI, 2013.

the researchers aim to using the Weight Lifting Exercises dataset to investigate how (well) an human activity was performed by the wearer. Six young health participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions: exactly according to the specification (Class A), throwing the elbows to the front (Class B), lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) and throwing the hips to the front (Class E). Class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes. Participants were supervised by an experienced weight lifter to make sure the execution complied to the manner they were supposed to simulate. The exercises were performed by six male participants aged between 20-28 years, with little weight lifting experience.

In this project we will use machine learing algorithm to predict the manner in which six participants did the exercise. The data for this project comes from this original source: http://groupware.les.inf.puc-rio.br/har.

Getting the Data

The training data for this project are available here:

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

The test data are available here:

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

Import these data now

```
set.seed(100)
url_training <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
url_testing <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
training <- read.csv(url(url_training), na.strings=c("NA", "", "#DIV/0!"))
testing <- read.csv(url(url_testing), na.strings=c("NA", "", "#DIV/0!"))
```

Then we partition the training set into training set and cross-validation set.

```
library(caret)
```

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

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

```
inTraining <- createDataPartition(y=training$classe, p=0.75, list=FALSE)
train <- training[inTraining, ]
test <- training[-inTraining, ]
myTraining <- train[!(names(train) %in% c("class"))]
```

Preprocessing the Data

We will filter out the numerical values of the dataset.

```
myTraining <- myTraining[-c(1:7)]
myTraining <- myTraining[sapply(myTraining,is.numeric)]
```

Because there is a large anount of missing values, we will firstly fill the missimg values by the median of columns.

```
f = function(x){
   if (sum(is.na(x)) >= length(x)*0.6) {
     x <- 0
   }
   x[is.na(x)] = median(x, na.rm = TRUE)
   x
}
myTraining <- data.frame(apply(myTraining, 2, f))
dim(myTraining)
```

```
## [1] 14718   146
```

```
myTraining <- myTraining[!(apply(myTraining == 0, 2, all))]
dim(myTraining)
```

```
## [1] 14718    52
```

```
myTraining <- data.frame(myTraining, classe = train$class)
myTesting <- test[colnames(myTraining)]
```

After preprocessing, our dataset is reduced to contain only 52 columns.

Using machine learning to evaluate the data

Support vector machines

SVMs seek an optimal hyperplane for seperating two classes in multidimensional space. The hyperplane is chosen to maximize the margin between two classes' closest points. SVMs are available in R using the svm() function in the e1071 package.

```
library(e1071)
fit.svm <- svm(classe ~., data = myTraining)

svm.pred <- predict(fit.svm, na.omit(myTesting))
```

```
svm.perf <- table(na.omit(myTesting)$classe, svm.pred,
                  dnn= c("Actual", "Predicted"))
confusionMatrix(svm.perf)
```

```
## Confusion Matrix and Statistics
##
##       Predicted
## Actual    A    B    C    D    E
##      A 1392    2    1    0    0
##      B   75  848   25    1    0
##      C    4   32  809    9    1
##      D    2    0   89  713    0
##      E    0    3   31   19  848
##
## Overall Statistics
##
##                Accuracy : 0.94
##                  95% CI : (0.933, 0.9465)
##     No Information Rate : 0.3004
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.924
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9450   0.9582   0.8471   0.9609   0.9988
## Specificity            0.9991   0.9749   0.9884   0.9781   0.9869
## Pos Pred Value         0.9978   0.8936   0.9462   0.8868   0.9412
## Neg Pred Value         0.9769   0.9906   0.9639   0.9929   0.9998
## Prevalence             0.3004   0.1805   0.1947   0.1513   0.1731
## Detection Rate         0.2838   0.1729   0.1650   0.1454   0.1729
## Detection Prevalence   0.2845   0.1935   0.1743   0.1639   0.1837
## Balanced Accuracy      0.9721   0.9665   0.9177   0.9695   0.9929
```

The svm() function scales each variable to a mean of 0 and standard deviation of 1 before fitting the model by default. The SVMs is also unable to accommodate missing predictor values when classifying new cases. As you can see, the predictive accurary is good.

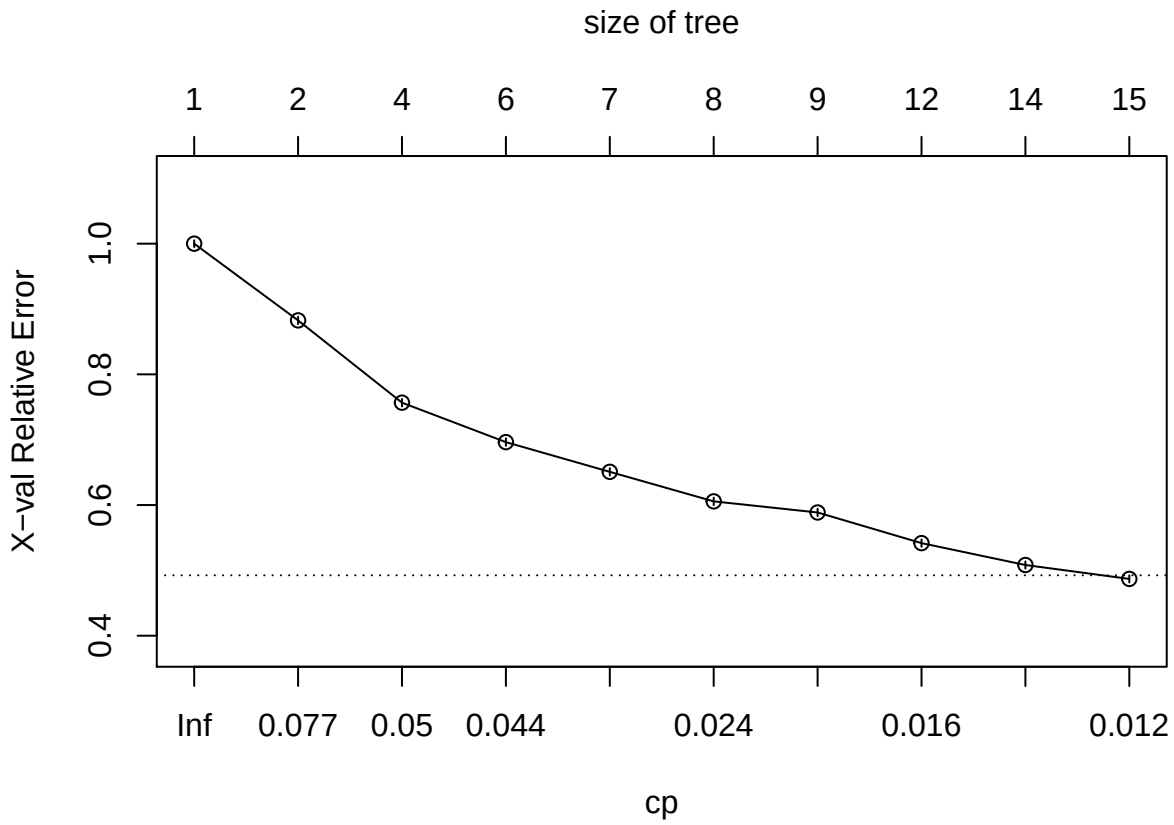
Decision trees

Decision trees are popular in data-mining contexts. They involve creating a set of binary splits on the predictor variables in order to create a tree that can be used to classify new observations into one of two groups. In R, decision trees can be grown and pruned using the raprt() and prune() functions in the rpart package.

```
library(rpart)
dtree <- rpart(classe ~ ., data = myTraining, method = "class",
               parms = list(split= "information"))
dtree$cptable
```

```
##            CP nsplit rel error    xerror        xstd
## 1  0.11782018      0 1.0000000 1.0000000 0.005195739
## 2  0.05060287      1 0.8821798 0.8825596 0.005555860
## 3  0.04879901      3 0.7809741 0.7566695 0.005739053
## 4  0.04053926      5 0.6833761 0.6962879 0.005758904
## 5  0.03076047      6 0.6428368 0.6508117 0.005745415
## 6  0.01822842      7 0.6120763 0.6056204 0.005707643
## 7  0.01713662      8 0.5938479 0.5886262 0.005687065
## 8  0.01566505     11 0.5348903 0.5417260 0.005611772
## 9  0.01357638     13 0.5035602 0.5082123 0.005540845
## 10 0.01000000     14 0.4899839 0.4869458 0.005488157
```

```
plotcp(dtree)
```



The plotcp() function plotthe cross-validated error agains the complexity parameter. The cross-validated parameter error is based on 10-fold cross validation. From the table, we see this error is high even before pruning.

Boosted tree model and parameter tuning

The accuracy of a predictive model can be improved using boosting algorithms like gradient boosting. The first step is tuning the model. Currently, k-fold cross-validation, leave-one-out cross-validation and bootstrap resampling methods can be used by train.

```
library(gbm)
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survival'

## The following object is masked from 'package:caret':
##
##     cluster

## Loading required package: splines

## Loading required package: parallel

## Loaded gbm 2.1.1
```

```
fitControl <- trainControl(method = "repeatedcv",
                           number = 5,
                           repeats = 1)
fit.gbm <- train(classe ~., data = myTraining, method = "gbm",
                 trControl = fitControl, verbose = FALSE)
```

```
## Loading required package: plyr
```

```
fit.gbm
```

```
## Stochastic Gradient Boosting
##
## 14718 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 1 times)
## Summary of sample sizes: 11775, 11775, 11773, 11775, 11774
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7501689  0.6830732
##   1                  100      0.8198798  0.7719345
##   1                  150      0.8553470  0.8169060
##   2                   50      0.8558901  0.8173437
##   2                  100      0.9079345  0.8834862
##   2                  150      0.9322590  0.9142910
##   3                   50      0.8956372  0.8678601
##   3                  100      0.9408195  0.9251148
##   3                  150      0.9610675  0.9507456
##
## 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.
```

```
gbm.pred <- predict(fit.gbm, myTesting)
gbm.perf <- table(na.omit(myTesting)$classe, gbm.pred,
                  dnn= c("Actual", "Predicted"))
confusionMatrix(gbm.perf)
```

```
## Confusion Matrix and Statistics
##
##       Predicted
## Actual    A    B    C    D    E
##      A 1378   12    5    0    0
##      B   30  894   24    0    1
##      C    0   34  804   14    3
##      D    0    1   32  769    2
##      E    0   14   10    7  870
##
## Overall Statistics
##
##                Accuracy : 0.9615
##                  95% CI : (0.9557, 0.9667)
##     No Information Rate : 0.2871
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9512
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9787   0.9361   0.9189   0.9734   0.9932
## Specificity            0.9951   0.9861   0.9873   0.9915   0.9923
## Pos Pred Value         0.9878   0.9420   0.9404   0.9565   0.9656
## Neg Pred Value         0.9915   0.9846   0.9825   0.9949   0.9985
## Prevalence             0.2871   0.1947   0.1784   0.1611   0.1786
## Detection Rate         0.2810   0.1823   0.1639   0.1568   0.1774
## Detection Prevalence   0.2845   0.1935   0.1743   0.1639   0.1837
## Balanced Accuracy      0.9869   0.9611   0.9531   0.9825   0.9927
```

Compared to the decision trees model, the accuracy of prediction is much improved.

Random forests

A random forest is an ensemble learning approach to supervised learning. The algorithm for a random forest involes sampling cases and variables to create a large number of decision trees. Each case is classified by each decision tree. The most common classification for that case is then used as the outcome. Random forests are grown using the randomForest() function in the randomForest package.

```
library(randomForest)
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
fit.forest <- randomForest(classe ~., data = myTraining,
                           na.action = na.roughfix, importance = TRUE)
forest.pred <- predict(fit.forest, myTesting)
forest.perf <- table(na.omit(myTesting)$classe, forest.pred,
                     dnn= c("Actual", "Predicted"))
confusionMatrix(forest.perf)
```

```
## Confusion Matrix and Statistics
##
##       Predicted
## Actual    A    B    C    D    E
##      A 1395    0    0    0    0
##      B    5  942    2    0    0
##      C    0    2  852    1    0
##      D    0    0   10  794    0
##      E    0    0    3    2  896
##
## Overall Statistics
##
##                Accuracy : 0.9949
##                  95% CI : (0.9925, 0.9967)
##     No Information Rate : 0.2855
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9936
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9964   0.9979   0.9827   0.9962   1.0000
## Specificity            1.0000   0.9982   0.9993   0.9976   0.9988
## Pos Pred Value         1.0000   0.9926   0.9965   0.9876   0.9945
## Neg Pred Value         0.9986   0.9995   0.9963   0.9993   1.0000
## Prevalence             0.2855   0.1925   0.1768   0.1625   0.1827
## Detection Rate         0.2845   0.1921   0.1737   0.1619   0.1827
## Detection Prevalence   0.2845   0.1935   0.1743   0.1639   0.1837
## Balanced Accuracy      0.9982   0.9981   0.9910   0.9969   0.9994
```

Random forests tend to be very accurate compared with other classification methods. Additionally, they can handle large problems, can handle large amounts of missing data in the training set, and can handle cases in which the number of variables is much greater than the number of observations.

Predicting results on the test data

As random forests gave the most accuracy in prediction, we will use it to make a prediction on the test data.

```
forest.pred <- predict(fit.forest, testing)
forest.pred
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  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
```