Human Activity recognition based on sensor data

Rajib Biswas

August 23, 2015

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, we will use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants to predict the manner in which they did the exercise. This is the "classe" variable in the training set. We will use 54 variables to predict with, and detail how to build the best model, cross validation, sample error and why the model is best choice. We will also use the prediction model to predict 20 different test cases.

Dataset

- training data : training (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv)
- test data : testing (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv)

Libraries

```
seedVar<-7689
set.seed(seedVar)
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.2.1
```

```
## Warning: package 'ggplot2' was built under R version 3.2.1
```

```
library(parallel)
library(doParallel)
```

```
## Warning: package 'doParallel' was built under R version 3.2.2
```

```
## Warning: package 'foreach' was built under R version 3.2.1
```

```
## Warning: package 'iterators' was built under R version 3.2.1
```

```
#parallel processing with multicore
registerDoParallel(makeCluster(detectCores()))
```

Step1.Load and prepare dataset

Dataset is downloaded in current directory.

```
#Load raw data
training <- read.csv("pml-training.csv", header = TRUE)
test  <- read.csv('pml-testing.csv', header = TRUE)

#traing dataset summary
dim(training)
```

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

```
dim(test)
```

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

```
table(training$classe)
```

```
##
##    A    B    C    D    E
## 5580 3797 3422 3216 3607
```

Clean up missing data: remove column having 80% or more missing data. Non zero variance column also removed

```
naCol<- apply(training,2,function(x) {sum(is.na(x))});
training <- training[,which(naCol <  nrow(training)*0.8)];
dim(training)
```

```
## [1] 19622    93
```

```
#remove near zero variance predictors
nz <- nearZeroVar(training, saveMetrics = TRUE)
training <- training[, nz$nzv==FALSE]
dim(training)
```

```
## [1] 19622    59
```

Removing irrelevant variables

variables such as X,user_name, timestamps, new_window are not important in predicting the “Classe”" variable of the dataset. Therefore, we have removed the irrelevant variables.

```
#remove not relevant columns for classification

removeIndex<- grep("timestamp|X|user_name|new_window",names(training))
training <- training[,-removeIndex]
dim(training)
```

```
## [1] 19622    54
```

```
#class into factor
training$classe <- factor(training$classe)
```

step2.Split the data

Split the data: 80% for training, 20% for testing.

```
set.seed(seedVar)
trainIndex <- createDataPartition(y = training$classe, p=0.8,list=FALSE)
trainingSample <- training[trainIndex,]
testingSample <- training[-trainIndex,]


dim(trainingSample)
```

```
## [1] 15699    54
```

```
dim(testingSample)
```

```
## [1] 3923   54
```

step3.Create machine learning models

Decision Tree, Random forest(rf), and boosted trees(gbm) algorithm are used to comapre

Model Selection

```
set.seed(seedVar)

model_dt <- train(classe ~ .,  method="rpart", data=trainingSample)
save(model_dt,file="model_dt.rda")
model_rf <- train(classe ~ .,  method="rf", data=trainingSample)
save(model_rf,file="model_rf.rda")
model_gbm <-train(classe ~ ., method = "gbm", data = trainingSample)
save(model_gbm,file="model_gbm.rda")
```

Confusion Matrix

```
# Scoring - Confusion matrix
print("decision tree........  ")
```

```
## [1] "decision tree........  "
```

```
dt_predict<- predict(model_dt, testingSample)
```

```
## Loading required package: rpart
```

```
print(confusionMatrix(dt_predict, testingSample$classe))
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction   A   B   C   D   E
##          A 998 320 105 200  45
##          B  25 245  24  95 136
##          C  90 194 555 327 122
##          D   0   0   0   0   0
##          E   3   0   0  21 418
## 
## Overall Statistics
##                                           
##                Accuracy : 0.5649          
##                  95% CI : (0.5492, 0.5805)
##     No Information Rate : 0.2845          
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.4387          
##  Mcnemar's Test P-Value : < 2.2e-16       
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8943  0.32279   0.8114   0.0000   0.5798
## Specificity            0.7613  0.91150   0.7737   1.0000   0.9925
## Pos Pred Value         0.5983  0.46667   0.4309      NaN   0.9457
## Neg Pred Value         0.9477  0.84873   0.9510   0.8361   0.9130
## Prevalence             0.2845  0.19347   0.1744   0.1639   0.1838
## Detection Rate         0.2544  0.06245   0.1415   0.0000   0.1066
## Detection Prevalence   0.4252  0.13383   0.3283   0.0000   0.1127
## Balanced Accuracy      0.8278  0.61715   0.7925   0.5000   0.7861
```

```
print("Random forest ...... ")
```

```
## [1] "Random forest ...... "
```

```
rf_predict<- predict(model_rf, testingSample)
```

```
## Loading required package: randomForest
```

```
## Warning: package 'randomForest' was built under R version 3.2.2
```

```
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

```
print(confusionMatrix(rf_predict, testingSample$classe))
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1115    5    0    0    0
##          B    0  753    1    0    0
##          C    0    0  683    8    0
##          D    0    1    0  635    1
##          E    1    0    0    0  720
## 
## Overall Statistics
##                                           
##                Accuracy : 0.9957          
##                  95% CI : (0.9931, 0.9975)
##     No Information Rate : 0.2845          
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.9945          
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9991   0.9921   0.9985   0.9876   0.9986
## Specificity            0.9982   0.9997   0.9975   0.9994   0.9997
## Pos Pred Value         0.9955   0.9987   0.9884   0.9969   0.9986
## Neg Pred Value         0.9996   0.9981   0.9997   0.9976   0.9997
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2842   0.1919   0.1741   0.1619   0.1835
## Detection Prevalence   0.2855   0.1922   0.1761   0.1624   0.1838
## Balanced Accuracy      0.9987   0.9959   0.9980   0.9935   0.9992
```

```
print("Boosted trees GBM ......")
```

```
## [1] "Boosted trees GBM ......"
```

```
gbm_predict<- predict(model_gbm , testingSample)
```

```
## Loading required package: gbm
```

```
## Warning: package 'gbm' was built under R version 3.2.2
```

```
## Loading required package: survival
##
## Attaching package: 'survival'
##
## The following object is masked from 'package:caret':
##
##     cluster
##
## Loading required package: splines
## Loaded gbm 2.1.1
## Loading required package: plyr
```

```
## Warning: package 'plyr' was built under R version 3.2.1
```

```
print(confusionMatrix(gbm_predict, testingSample$classe))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1111    6    0    1    0
##          B    2  745    3    0    1
##          C    0    7  678   15    1
##          D    3    1    3  625    2
##          E    0    0    0    2  717
##
## Overall Statistics
##
##                Accuracy : 0.988
##                  95% CI : (0.9841, 0.9912)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9848
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9955   0.9816   0.9912   0.9720   0.9945
## Specificity            0.9975   0.9981   0.9929   0.9973   0.9994
## Pos Pred Value         0.9937   0.9920   0.9672   0.9858   0.9972
## Neg Pred Value         0.9982   0.9956   0.9981   0.9945   0.9988
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2832   0.1899   0.1728   0.1593   0.1828
## Detection Prevalence   0.2850   0.1914   0.1787   0.1616   0.1833
## Balanced Accuracy      0.9965   0.9898   0.9921   0.9846   0.9969
```

Random Forest algorithm is selected as, its having high accuracy of 99.4%

```
print(model_rf)
```

```
## Random Forest
##
## 15699 samples
##    53 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 15699, 15699, 15699, 15699, 15699, 15699, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa      Accuracy SD  Kappa SD
##    2    0.9938978  0.9922771  0.001131254  0.001430307
##   27    0.9968597  0.9960258  0.001154544  0.001461000
##   53    0.9940750  0.9925006  0.001822005  0.002308311
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 27.
```

Cross validation and tuning of Random forest

10 fold and 10 repeated cross validation

```
set.seed(seedVar)
registerDoParallel(makeCluster(detectCores()))

cv_control <- trainControl(method = "repeatedcv", number = 10, repeats = 10)
model_rf_CV <- train(classe ~ ., method="rf",  data=trainingSample, trControl = cv_contro
l)
save(model_rf_CV,file="model_rf_CV.rda")
```

Our final model (model rf CV) will have accuracy nearly > 99%.

step4.Prediction with testing data

As we have splitted the data into two sets, we have trained our model with training data. Now, We have our ready model and we will use the rest of 20% of data to test the model

Predict on sample test set

```
#Use best fit to predict testing data
set.seed(seedVar)
print("Random forest accuracy after Cross validation")
```

```
## [1] "Random forest accuracy after Cross validation"
```

```
rf_CV_accuracy<<- predict(model_rf_CV , testingSample)
print(confusionMatrix(rf_CV_accuracy, testingSample$classe))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1116    1    0    0    0
##          B    0  758    0    0    0
##          C    0    0  684    1    0
##          D    0    0    0  642    1
##          E    0    0    0    0  720
##
## Overall Statistics
##
##                Accuracy : 0.9992
##                  95% CI : (0.9978, 0.9998)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.999
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9987   1.0000   0.9984   0.9986
## Specificity            0.9996   1.0000   0.9997   0.9997   1.0000
## Pos Pred Value         0.9991   1.0000   0.9985   0.9984   1.0000
## Neg Pred Value         1.0000   0.9997   1.0000   0.9997   0.9997
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1932   0.1744   0.1637   0.1835
## Detection Prevalence   0.2847   0.1932   0.1746   0.1639   0.1835
## Balanced Accuracy      0.9998   0.9993   0.9998   0.9991   0.9993
```

Accuracy of prediction

```
set.seed(seedVar)
postResample(rf_CV_accuracy, testingSample$classe)
```

```
##  Accuracy     Kappa
## 0.9992353 0.9990327
```

Expected out of sample error

Accuracy of predictions is about 99.9% therefore the expected out of sample error is around less than 1% (1 - 0.99)

Predict 20 test cases for submission

```
set.seed(seedVar)
#predict
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)
  }
}


#Prediction
saveOut<- function(){
  prediction <- predict(model_rf_CV, test)
  print(prediction)
  answers <- as.vector(prediction)
  pml_write_files(answers)
}
#dump output
saveOut()
```

```
##  [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
```

Conclusion

- Accuracy of Decision Tree: 56.5%
- Accuracy of RF: 99.6%
- Accuracy of RF: 98.8%

Random Forest algorithm performed better than Decision Trees or boosted tree.

Note:- Prediction on 20 sample test set is found to 100% correct. :)

Appendix

Appendix 1

Variable importance

```
#Important Variables
print("Variables importance")
```

```
## [1] "Variables importance"
```

```
vi = varImp(model_rf_CV$finalModel)
vi$var<-rownames(vi)
vi = as.data.frame(vi[with(vi, order(vi$Overall, decreasing=TRUE)), ])
rownames(vi) <- NULL
print(vi)
```

```
##       Overall                  var
## 1  1966.04586           num_window
## 2  1260.62465            roll_belt
## 3   786.42774        pitch_forearm
## 4   615.59875             yaw_belt
## 5   592.63785    magnet_dumbbell_z
## 6   580.34977           pitch_belt
## 7   566.82451    magnet_dumbbell_y
## 8   482.54803         roll_forearm
## 9   270.85035     accel_dumbbell_y
## 10  224.74928        roll_dumbbell
## 11  224.16174    magnet_dumbbell_x
## 12  218.02030         accel_belt_z
## 13  216.31478      accel_forearm_x
## 14  190.62559 total_accel_dumbbell
## 15  165.47967        magnet_belt_y
## 16  162.10730     accel_dumbbell_z
## 17  156.23588        magnet_belt_z
## 18  148.20591     magnet_forearm_z
## 19  126.19355        magnet_belt_x
## 20  113.55748             roll_arm
## 21  108.55634         yaw_dumbbell
## 22  106.79007      accel_forearm_z
## 23   95.63392         gyros_belt_z
## 24   92.84934     accel_dumbbell_x
## 25   83.20314     gyros_dumbbell_y
## 26   80.67248              yaw_arm
## 27   78.96780     magnet_forearm_y
## 28   77.81491         magnet_arm_x
## 29   73.69676         magnet_arm_y
## 30   73.10030          yaw_forearm
## 31   72.97531          accel_arm_x
## 32   68.21375     magnet_forearm_x
## 33   67.60595       pitch_dumbbell
## 34   63.80004            pitch_arm
## 35   57.07322     total_accel_belt
## 36   50.05137          gyros_arm_y
## 37   43.28527         magnet_arm_z
## 38   42.37498         accel_belt_y
## 39   41.94336      accel_forearm_y
```

```
## 40   39.94867         gyros_belt_y
## 41   39.55005          accel_arm_y
## 42   37.12182      gyros_forearm_y
## 43   34.61046          gyros_arm_x
## 44   34.34569     gyros_dumbbell_x
## 45   32.64190         accel_belt_x
## 46   30.22265         gyros_belt_x
## 47   30.11573      total_accel_arm
## 48   29.69661          accel_arm_z
## 49   24.88457  total_accel_forearm
## 50   23.72905      gyros_forearm_z
## 51   23.19705     gyros_dumbbell_z
## 52   19.25695      gyros_forearm_x
## 53   14.90766          gyros_arm_z
```