Practical Machine Learning Project

Mukesh Kanchan

Wednesday, September 17, 2014

Overview

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 Â- 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 data set, the participants 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 (http://groupware.les.inf.puc-rio.br/har) (see the section on the Weight Lifting Exercise Dataset).

In this project, the goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants toto predict the manner in which praticipants did the exercise.

The dependent variable or response is the “classe” variable in the training set.

Data

Download and load the data

```
#download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv", destfile = "./PML/pml-training.csv")
#download.file("http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv", destfile = "./PML/pml-testing.csv")



trainingOrg = read.csv("pml-training.csv", na.strings=c("", "NA", "NULL"))
# data.train =  read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv", na.strings=c("", "NA", "NULL"))

testingOrg = read.csv("pml-testing.csv", na.strings=c("", "NA", "NULL"))
dim(trainingOrg)
```

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

```
dim(testingOrg)
```

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

Process data

Reduce the number of predictors:

1. Remove variable with high NAs

```
training.dena <- trainingOrg[ , colSums(is.na(trainingOrg)) == 0]
#head(training1)
#training3 <- training.decor[ rowSums(is.na(training.decor)) == 0, ]
dim(training.dena)
```

```
## [1] 19622    60
```

2. Remove irrelevent variables

```
remove = c('X', 'user_name', 'raw_timestamp_part_1', 'raw_timestamp_part_2', 'cvtd_timestamp', 'new_w
indow', 'num_window')
training.dere <- training.dena[, -which(names(training.dena) %in% remove)]
dim(training.dere)
```

```
## [1] 19622    53
```

3. Variables with extremely low variance

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.1.1
```

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

```
zeroVar= nearZeroVar(training.dere[sapply(training.dere, is.numeric)], saveMetrics = TRUE)
training.nonzerovar = training.dere[,zeroVar[, 'nzv']==0]
dim(training.nonzerovar)
```

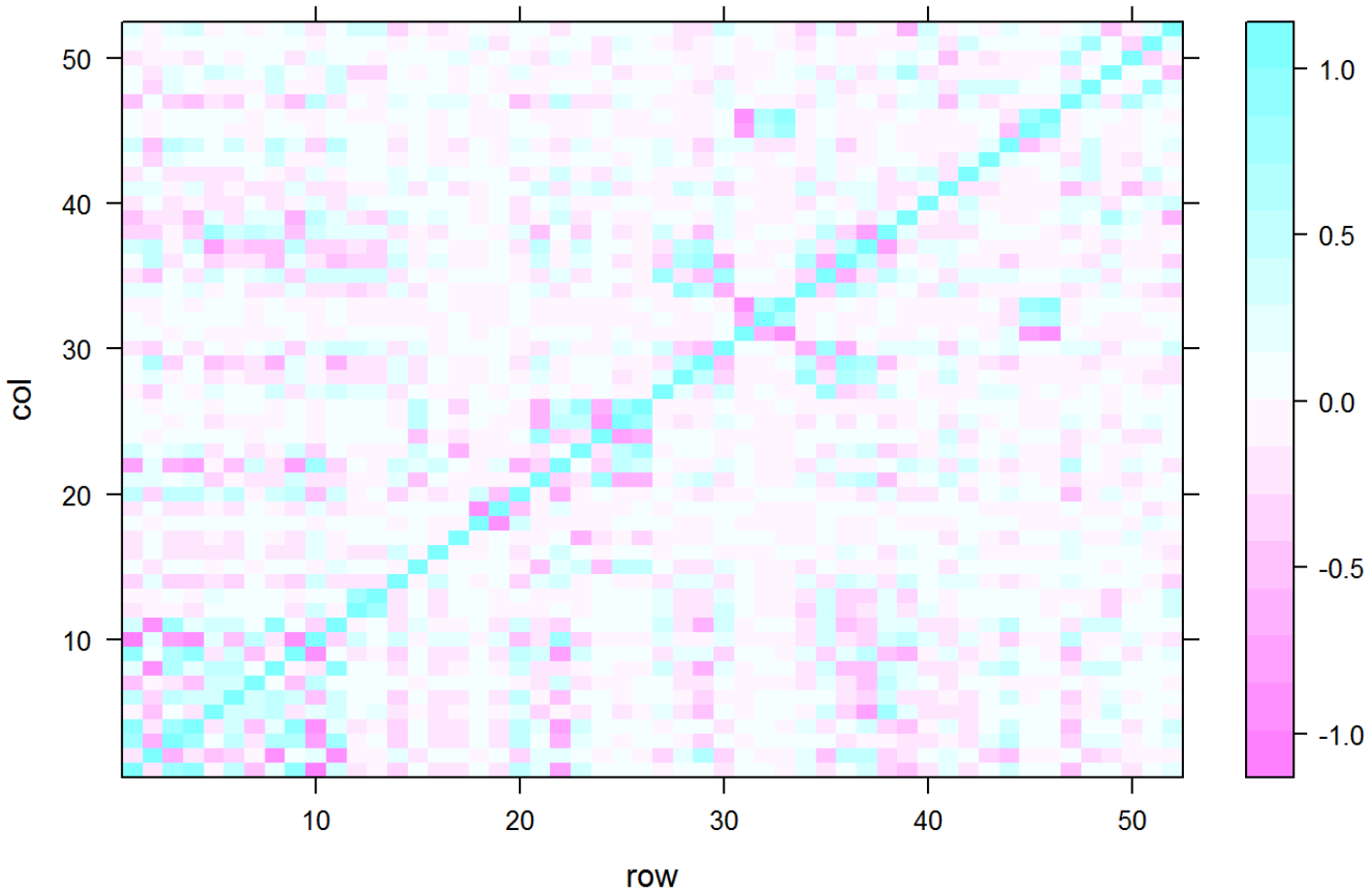
```
## [1] 19622    53
```

4. Highly coorelated variables

```
corrMatrix <- cor(na.omit(training.nonzerovar[sapply(training.nonzerovar, is.numeric)]))
dim(corrMatrix)
```

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

```
corrDF <- expand.grid(row = 1:52, col = 1:52)
corrDF$correlation <- as.vector(corrMatrix)
levelplot(correlation ~ row+ col, corrDF)
```




```
removecor = findCorrelation(corrMatrix, cutoff = .90, verbose = FALSE)
training.decor = training.nonzerovar[,-removecor]
dim(training.decor)
```

```
## [1] 19622    46
```

Split data to training and testing for cross validation.

```
inTrain <- createDataPartition(y=training.decor$classe, p=0.7, list=FALSE)
training <- training.decor[inTrain,]; testing <- training.decor[-inTrain,]
dim(training);dim(testing)
```

```
## [1] 13737    46
```

```
## [1] 5885   46
```

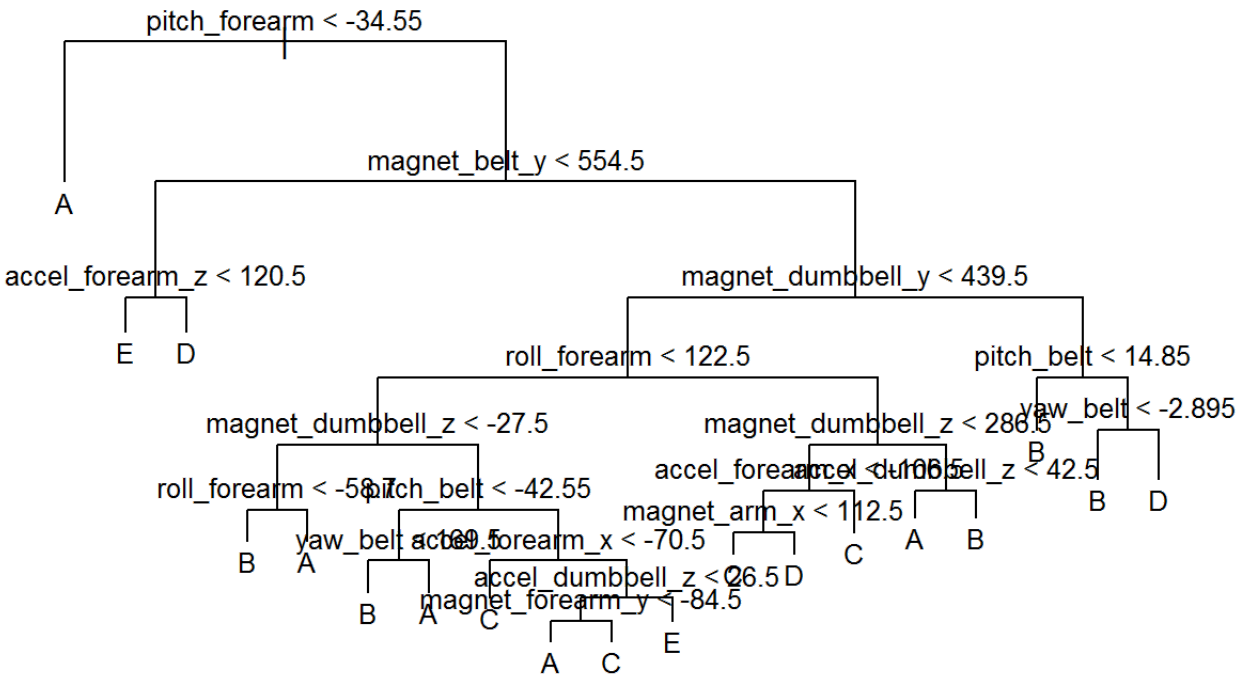
Analysis

1. Regression Tree

```
library(tree)
```

```
## Warning: package 'tree' was built under R version 3.1.1
```

```
set.seed(12345)
tree.training=tree(classe~.,data=training)
plot(tree.training)
text(tree.training,pretty=0, cex =.8)
```



2. Cross Validation Lets check the performance of tree on testing data by cross validation

```
modFit <- train(classe ~ .,method="rpart",data=training)
```

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

```
tree.pred=predict(tree.training,testing,type="class")
predMatrix = with(testing,table(tree.pred,classe))
sum(diag(predMatrix))/sum(as.vector(predMatrix))
```

```
## [1] 0.6562
```

This is not very accurate.

```
tree.pred=predict(modFit,testing)
predMatrix = with(testing,table(tree.pred,classe))
sum(diag(predMatrix))/sum(as.vector(predMatrix))
```

```
## [1] 0.5023
```

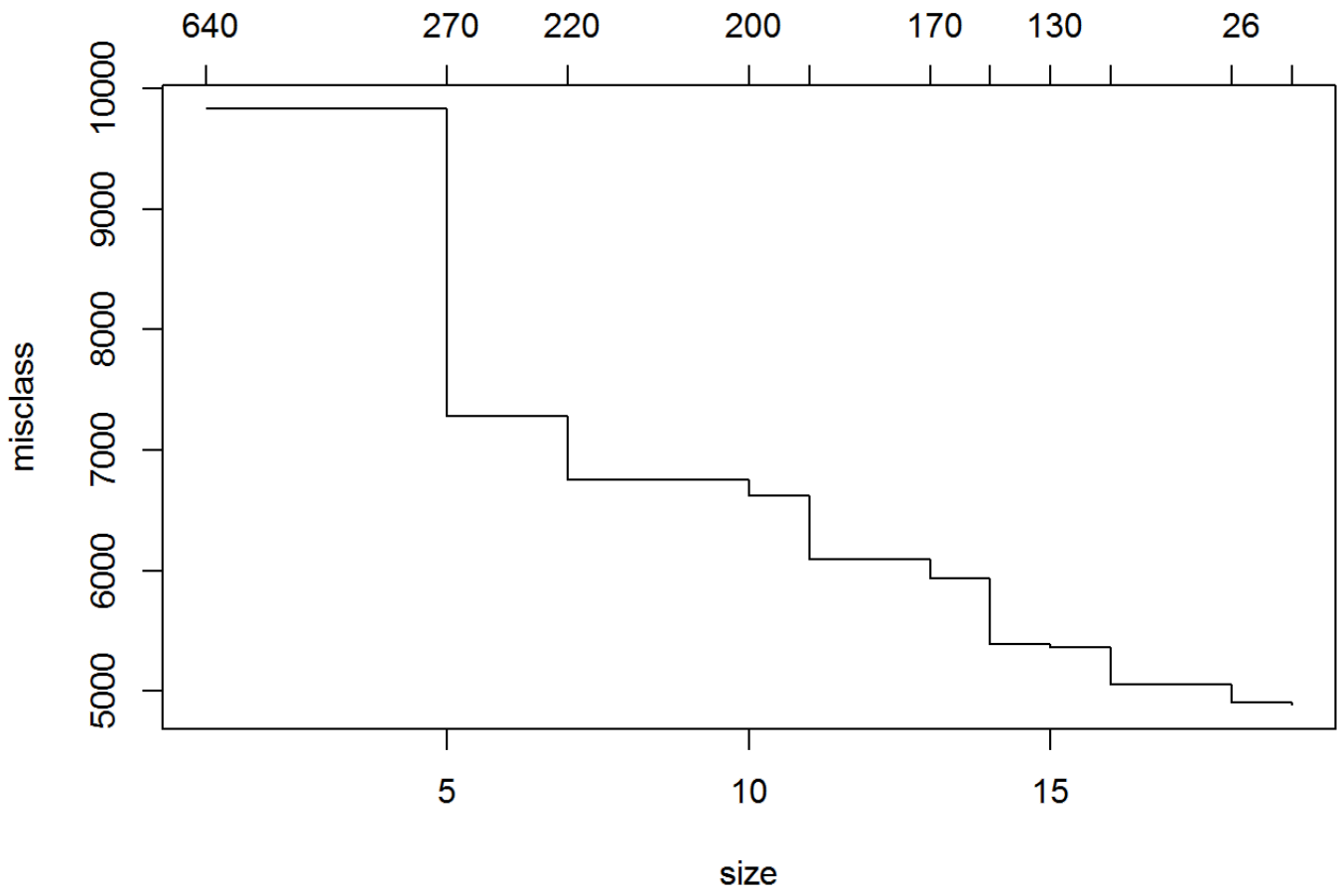
This from ‘caret’ package is much lower than the result from ‘tree’ package.

3. Purning Tree Use cross validation to purne the heavy/dip branches

```
cv.training=cv.tree(tree.training,FUN=prune.misclass)
cv.training
```

```
## $size
##  [1] 19 18 16 15 14 13 11 10  7  5  1
##
## $dev
##  [1] 4887 4903 5054 5363 5390 5940 6096 6619 6755 7284 9831
##
## $k
##  [1]  -Inf  26.0 104.0 133.0 136.0 172.0 188.5 200.0 219.3 270.0 638.8
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
plot(cv.training)
```



It shows that when the size of the tree goes down, the deviance goes up. It means the 21 is a good size (i.e. number of terminal nodes) for this tree. We do not need to prune it.

4. Random Tree Random forests build lots of bushy trees, and then average them to reduce the variance.

```
require(randomForest)
```

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

```
## Warning: package 'randomForest' was built under R version 3.1.1
```

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

```
set.seed(12345)
rf.training=randomForest(classe~.,data=training,ntree=100, importance=TRUE)
rf.training
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = training, ntree = 100,      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 0.65%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3898    5    2    1    0    0.002048
## B   13 2638    7    0    0    0.007524
## C    1   19 2374    2    0    0.009182
## D    0    0   28 2221    3    0.013766
## E    0    0    3    5 2517    0.003168
```

```
varImpPlot(rf.training,)
```

It shows which variable has higher impact on perdiction.

Out of Sample Accuracy

Random Forest model shows OOB estimate of error rate: 0.72% for the training data. Now lets predict it for out-of sample accuracy.

```
tree.pred=predict(rf.training,testing,type="class")
predMatrix = with(testing,table(tree.pred,classe))
sum(diag(predMatrix))/sum(as.vector(predMatrix))
```

```
## [1] 0.9942
```

0.99 means we got a very accurate estimate.

Conclusion

We can predict the testing data from the website.

```
answers <- predict(rf.training, testingOrg)
answers
```

```
##  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
```

Those answers are going to submit to website for grading. It shows that this random forest model did a good job.