Project PML

Nosequiensoy007

5/12/2020

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
The data for this project come from this source: http://groupware.les.inf.puc-rio.br/har. If you use the document you create for this class for any purpose please cite them as they have been very generous in allowing their data to be used for this kind of assignment. Initialization
##load libraries

```
library(ggplot2)
library(lattice)
library(caret)
library(rpart)
library(rpart.plot)
library(RColorBrewer)
library(rattle)
library(randomForest)
library(gbm)
library(plyr)
library(corrplot)
library(survival)
library(splines)
library(tibble)
library(bitops)
library(parallel)
```

I downloaded, change wd and loaded variables

```
setwd("C:/Users/juanfidel18/Desktop/Coursera/Practical Machine Learning")
TrainData <- read.csv("pml-training.csv",header=TRUE)
TestData <- read.csv("pml-testing.csv",header=TRUE)
```

The training data set is made of 19622 observations on 160 columns. We can notice that many columns have NA values or blank values on almost every observation. So we will remove them, because they will not produce any information. The first seven columns give information about the people who did the test, and also timestamps. We will not take them in our model.

```
# Here we get the indexes of the columns having at least 90% of NA or
blank values on the training dataset
```

```
indColToRemove <- which(colSums(is.na(TrainData)
|TrainData=="")>0.9*dim(TrainData)[1])
TrainDataClean <- TrainData[,-indColToRemove]
TrainDataClean <- TrainDataClean[,-c(1:7)]
```

We do the same for the test set

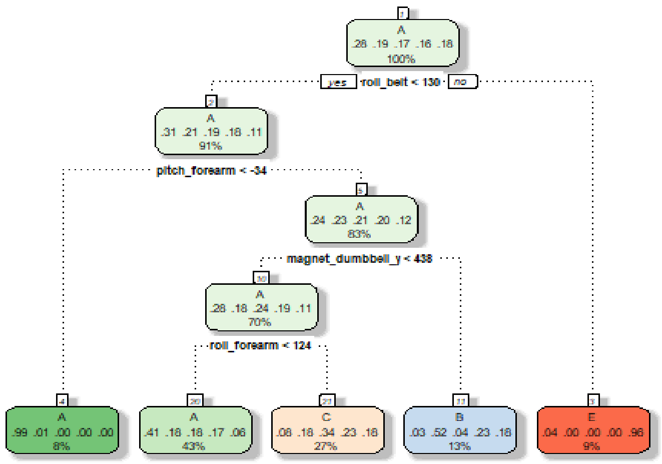
```
indColToRemove <- which(colSums(is.na(TestData)
|TestData=="")>0.9*dim(TestData)[1])
TestDataClean <- TestData[,-indColToRemove]
TestDataClean <- TestDataClean[,-1]
```

After cleaning, the new training data set has only 53 columns.

```
set.seed(12345)
inTrain1 <- createDataPartition(TrainDataClean$classe, p=0.75,
list=FALSE)
Train1 <- TrainDataClean[inTrain1,]
Test1 <- TrainDataClean[-inTrain1,]
```

In order to limit the effects of overfitting, and improve the efficicency of the models, we will use the *cross-validation technique. We will use 5 folds (usually, 5 or 10 can be used, but 10 folds gives higher run times with no significant increase of the accuracy).

```
trControl <- trainControl(method="cv", number=5)
model_CT <- train(classe~., data=Train1, method="rpart",
trControl=trControl)

fancyRpartPlot(model_CT$finalModel)
```



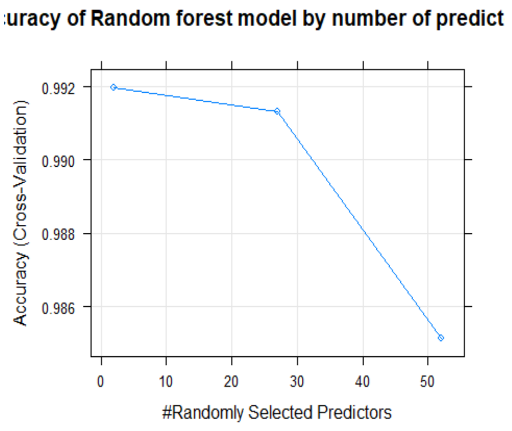
Rattle 2020-dic.-06 01:51:27 juanfidel18

```
# Change the factors to avoid Error: data and reference should be factors
with the same levels.
classe <- as.factor(Test1$classe)
trainpred <- predict(model_CT,newdata=Test1)
confMatCT <- confusionMatrix(classe,trainpred)
```

```
# Model accuracy
confMatCT$overall[1]

##  Accuracy
## 0.4877651
```

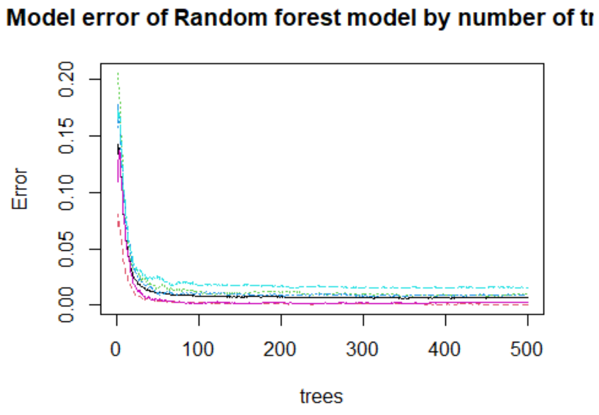
We can notice that the accuracy of this first model is very low (about 55%). This means that the outcome class will not be predicted very well by the other predictors. Train with random forests

```
model_RF <- train(classe~., data=Train1, method="rf",
trControl=trControl, verbose=FALSE)

plot(model_RF,main="Accuracy of Random forest model by number of
predictors")
```



```
trainpred <- predict(model_RF,newdata=Test1)
confMatRF <- confusionMatrix(classe,trainpred)
# Model accuracy
confMatRF$overall[1]

##  Accuracy
## 0.9938825

names(model_RF$finalModel)

##  [1] "call"            "type"            "predicted"       "err.rate"
##  [5] "confusion"       "votes"           "oob.times"       "classes"
##  [9] "importance"      "importanceSD"    "localImportance" "proximity"
## [13] "ntree"           "mtry"            "forest"          "y"
## [17] "test"            "inbag"           "xNames"
"problemType"
## [21] "tuneValue"       "obsLevels"       "param"

model_RF$finalModel$classes

## [1] "A" "B" "C" "D" "E"
```

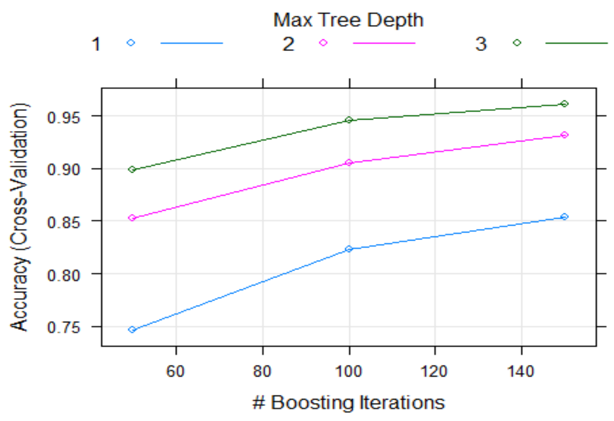
```
plot(model_RF$finalModel,main="Model error of Random forest model by
number of trees")
```



```
# Compute the variable importance
MostImpVars <- varImp(model_RF)
```

With random forest, we reach an accuracy of 99.3% using cross-validation with 5 steps. This is very good. But let's see what we can expect with Gradient boosting.At last, using more than about 30 trees does not reduce the error significantly.##Train with gradient boosting method

```
model_GBM <- train(classe~., data=Train1, method="gbm",
trControl=trControl, verbose=FALSE)

plot(model_GBM)
```



```
trainpred <- predict(model_GBM,newdata=Test1)
confMatGBM <- confusionMatrix(classe,trainpred)

confMatGBM$overall[1]
##  Accuracy
## 0.9604405
```

Precision with 5 folds is 95.9%.

```
##Conclusion
```

This shows that the random forest model is the best one. We will then use it to predict the values of classe for the test data set.

```
FinalTestPred <- predict(model_RF,newdata=TestDataClean)
FinalTestPred
```

```
##  [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
```