Practical Machine Learning Course Project

Jason Liu

Oct 7, 2016

Acknowledgement

Author:

Jiashen Liu

https://nl.linkedin.com/in/jiashen-liu

This project serves as the graduation project of the course 'Practical Machine Learning', taught by JHU. The data is provided by **http://groupware.les.inf.puc-rio.br/har** and will only be used for educational purpose.

Background and requirement of the project

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, 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. More information is available from the website here: http://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

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. You should create a report describing how you built your model, how you used cross validation, what you think the expected out of sample error is, and why you made the choices you did. You will also use your prediction model to predict 20 different test cases.

Download Data and Project Design

We first download the data from the given URLs.

```
url_train<- 'https://d396qusza40orc.cloudfront.net/predmachlearn/pml-
training.csv'
url_test <- 'https://d396qusza40orc.cloudfront.net/predmachlearn/pml-
testing.csv'
download.file(url=url_train, destfile='training.csv', method="curl")
download.file(url=url_test,  destfile ='test.csv' ,method='curl' )
train<- read.csv('training.csv',na.strings = c('NA',''))
test <- read.csv('test.csv',na.strings = c('NA',''))
```

The overall question is a classification problem. Therefore, after data preprocessing, the key remaining question will be the model selection. I plan to fit the data into three classic models, namely the classification tree, the random forest and the gradiant boosting model. Then I plan to stack three models together to see whether a combined model can improve the performance of the predictive analytic. We load the required packages below.

```
library(gbm)
library(randomForest)
library(ggplot2)
library(rpart.plot)
library(rpart)
library(rattle)
library(caret)
library(e1071)
library(xgboost)
```

Data Preprocessing

We first split the training data into two parts with the ratio of 7/3. Before that, we need to transform the response variable into factor.

```
train$classe<-as.factor(train$classe)
set.seed(32423)
intrain<- createDataPartition(train$classe,p=0.7,list = FALSE)
training<- train[intrain,]
testing <- train[-intrain,]
```

Before fitting the models, we need to perform data cleansing process. The response variable of the data is 'classe', which represent ways of the barbell lift movement. The first step one need to do is to delete the unnecessary variables in the dataset. It is known that the id column is useless in terms of model training. Therefore, we drop it directly.

```
training<- training[,-1]
testing <- testing [,-1]
test<- test[,-1]
```

For the classification problem, the feature engineering is always not as complicated as the regression problem. But we still need to check whether there is a variable without variability existing in the dataset.

```
zeroVar <- nearZeroVar(training, saveMetrics=TRUE)
nzv_columns<- row.names(zeroVar[which(zeroVar$nzv==T),])
zerovar<-c()
for(i in 1:length(nzv_columns)){
  zerovar<-append(zerovar,nzv_columns[i])
}
training<- training[,-which(colnames(training)%in%zerovar)]
```

Also, due to the fact that the integrity of the dataset is poor, we have to clean the variables that have too many 'NA' values. We create a benchmark indicating that if the percentage of 'NA' values reaches to 80%, then we drop the corresponding variables off.

```
Drop_Index<- c()
Benchmark<-0.8
Num_NA<-sapply(training,function(y)length(which(is.na(y)==T)))
NA_Percentage<- Num_NA/nrow(training)
for(i in 1:length(NA_Percentage)){
  if(NA_Percentage[i]>=Benchmark){
    Drop_Index<-append(i,Drop_Index)
  }
}
training <- training[,-Drop_Index]
```

The data cleansing process is finished at this step. We perform the same process on the testing set and final test set.

```
Indexes<- match(colnames(training),colnames(testing))
testing<- testing[,Indexes]
Ind_2<- match(colnames(training[,-58]),colnames(test))
test<- test[,Ind_2]
```

Model Fitting and Evaluation

A: Classification Tree

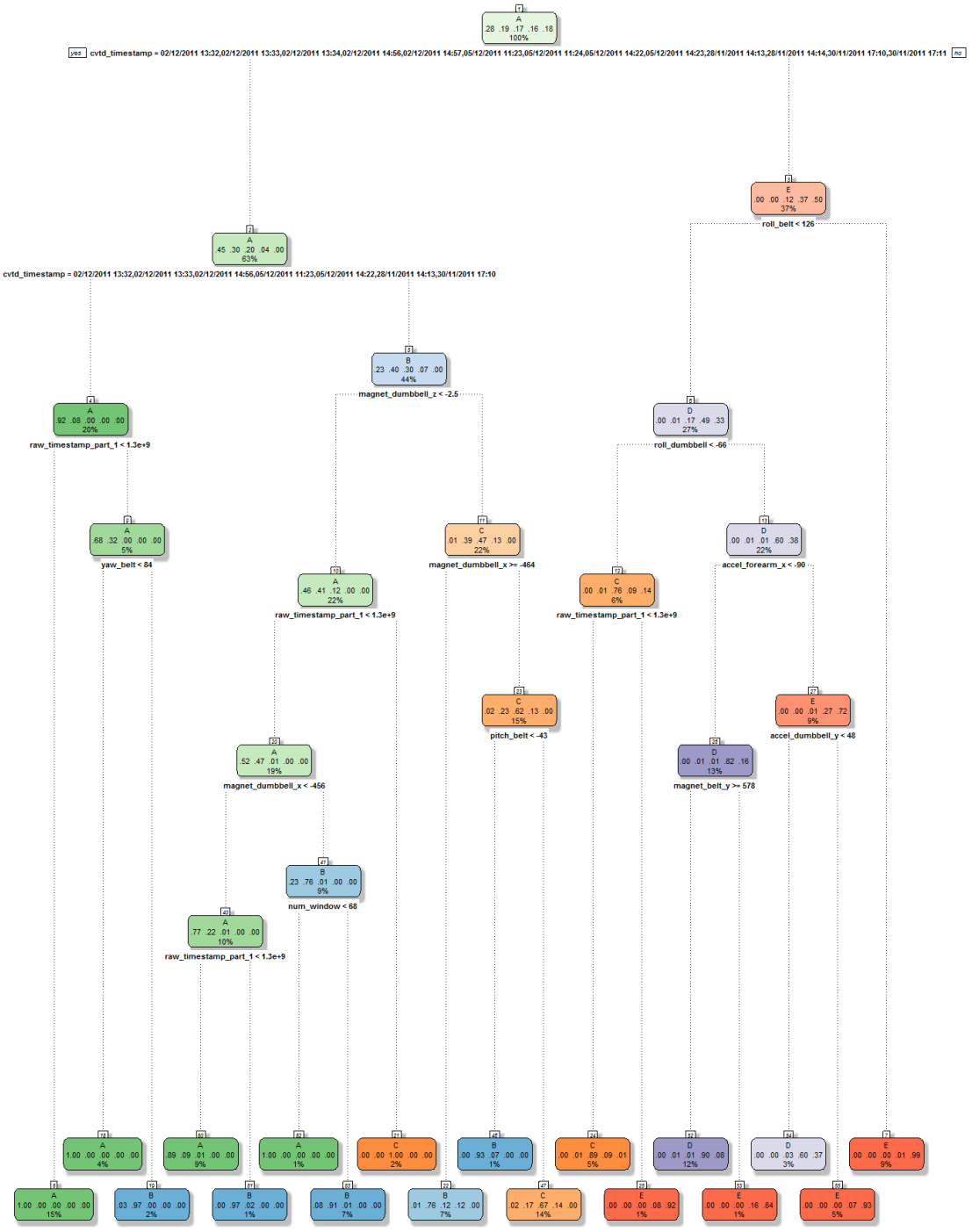
First model I fit is classification tree model.

```
treeFit<- rpart(classe~.,data=training,method = 'class')
print(treeFit)
```

```
## n= 13737
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
```

```
##  1) root 13737 9831 A (0.28 0.19 0.17 0.16 0.18)
##    2) cvtd_timestamp=02/12/2011 13:32,02/12/2011 13:33,02/12/2011
13:34,02/12/2011 14:56,02/12/2011 14:57,05/12/2011 11:23,05/12/2011
11:24,05/12/2011 14:22,05/12/2011 14:23,28/11/2011 14:13,28/11/2011
14:14,30/11/2011 17:10,30/11/2011 17:11 8717 4811 A (0.45 0.3 0.2 0.045
0)
##      4) cvtd_timestamp=02/12/2011 13:32,02/12/2011 13:33,02/12/2011
14:56,05/12/2011 11:23,05/12/2011 14:22,28/11/2011 14:13,30/11/2011
17:10 2729  229 A (0.92 0.084 0 0 0)
##        8) raw_timestamp_part_1< 1.322833e+09 2011    0 A (1 0 0 0 0)
*
##        9) raw_timestamp_part_1>=1.322833e+09 718  229 A (0.68 0.32 0
0 0)
##         18) yaw_belt< 84 482    0 A (1 0 0 0 0) *
##         19) yaw_belt>=84 236    7 B (0.03 0.97 0 0 0) *
##      5) cvtd_timestamp=02/12/2011 13:34,02/12/2011 14:57,05/12/2011
11:24,05/12/2011 14:23,28/11/2011 14:14,30/11/2011 17:11 5988 3579 B
(0.23 0.4 0.3 0.065 0)
##       10) magnet_dumbbell_z< -2.5 2956 1590 A (0.46 0.41 0.12 0.0017
0)
##         20) raw_timestamp_part_1< 1.323095e+09 2619 1253 A (0.52
0.47 0.011 0.0019 0)
##           40) magnet_dumbbell_x< -455.5 1421  330 A (0.77 0.22 0.013
0.0035 0)
##             80) raw_timestamp_part_1< 1.323084e+09 1221  130 A (0.89
0.092 0.011 0.0041 0) *
##             81) raw_timestamp_part_1>=1.323084e+09 200    5 B (0
0.97 0.025 0 0) *
##           41) magnet_dumbbell_x>=-455.5 1198  285 B (0.23 0.76
0.0083 0 0)
##             82) num_window< 68.5 197    0 A (1 0 0 0 0) *
##             83) num_window>=68.5 1001   88 B (0.078 0.91 0.01 0 0) *
##         21) raw_timestamp_part_1>=1.323095e+09 337    0 C (0 0 1 0
0) *
##       11) magnet_dumbbell_z>=-2.5 3032 1616 C (0.013 0.39 0.47 0.13
0)
##         22) magnet_dumbbell_x>=-464.5 927  227 B (0.0076 0.76 0.12
0.12 0) *
##         23) magnet_dumbbell_x< -464.5 2105  797 C (0.016 0.23 0.62
0.13 0)
##           46) pitch_belt< -43.35 178   13 B (0 0.93 0.073 0 0) *
##           47) pitch_belt>=-43.35 1927  632 C (0.017 0.17 0.67 0.14
0) *
##    3) cvtd_timestamp=02/12/2011 13:35,02/12/2011 14:58,02/12/2011
14:59,05/12/2011 11:25,05/12/2011 14:24,28/11/2011 14:15,30/11/2011
17:12 5020 2495 E (0 0.004 0.12 0.37 0.5)
##      6) roll_belt< 125.5 3722 1881 D (0 0.0054 0.17 0.49 0.33)
##       12) roll_dumbbell< -66.03498 759  181 C (0 0.0066 0.76 0.092
0.14)
##         24) raw_timestamp_part_1< 1.323084e+09 650   72 C (0 0.0077
```

```
0.89 0.094 0.0092) *
##         25) raw_timestamp_part_1>=1.323084e+09 109    9 E (0 0 0
0.083 0.92) *
##       13) roll_dumbbell>=-66.03498 2963 1192 D (0 0.0051 0.012 0.6
0.38)
##         26) accel_forearm_x< -89.5 1776  324 D (0 0.0084 0.012 0.82
0.16)
##           52) magnet_belt_y>=578.5 1587  166 D (0 0.0095 0.014 0.9
0.081) *
##           53) magnet_belt_y< 578.5 189   31 E (0 0 0 0.16 0.84) *
##         27) accel_forearm_x>=-89.5 1187  334 E (0 0 0.013 0.27 0.72)
##           54) accel_dumbbell_y< 48.5 455  184 D (0 0 0.033 0.6 0.37)
*
##           55) accel_dumbbell_y>=48.5 732   48 E (0 0 0 0.066 0.93) *
##      7) roll_belt>=125.5 1298   19 E (0 0 0 0.015 0.99) *

fancyRpartPlot(treeFit)
```



Rattle 2016-10月-07 20:15:47 apple

```
Prediction1<- predict(treeFit,newdata=testing,type = 'class')
TreeAcu<-confusionMatrix(Prediction1,testing$classe)$overall[1]
TreeAcu
```

```
##  Accuracy
## 0.8769754
```

The accuracy of tree model is good, but not quite perfect. We can still see the potential improvement on the prediction.

B: Naive Bayes

Secondly, we choose to build a Naive Bayes classifier.

```
NBFit<- naiveBayes(classe~.,data=training)
Prediction3<- predict(NBFit,newdata = testing,type = 'class')
NBAccu<- confusionMatrix(Prediction3,testing$classe)$overall[1]
NBAccu
```

```
##  Accuracy
## 0.6807137
```

The naive bayesian model is quite basic and cannot produce a satisfied result.

C: Random Forest

The third model I choose to fit is a random forest model.

```
set.seed(32423)
rfFit<- randomForest(classe~.,data= training)
print(rfFit)
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = training)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.14%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3905    1    0    0    0 0.0002560164
## B    2 2656    0    0    0 0.0007524454
## C    0    5 2389    2    0 0.0029215359
## D    0    0    5 2245    2 0.0031083481
## E    0    0    0    2 2523 0.0007920792
```

```
Prediction2<- predict(rfFit,newdata = testing[,-58])
rfAccu<- confusionMatrix(Prediction2,testing$classe)$overall[1]
rfAccu
```

```
##  Accuracy
## 0.9984707
```

The accuracy of the random forest model is amazingly high. However, I am also interested in the performance of the boosting model.

D:Support Vector Machine

The fourth model I choose to fit is a logistic regression model.

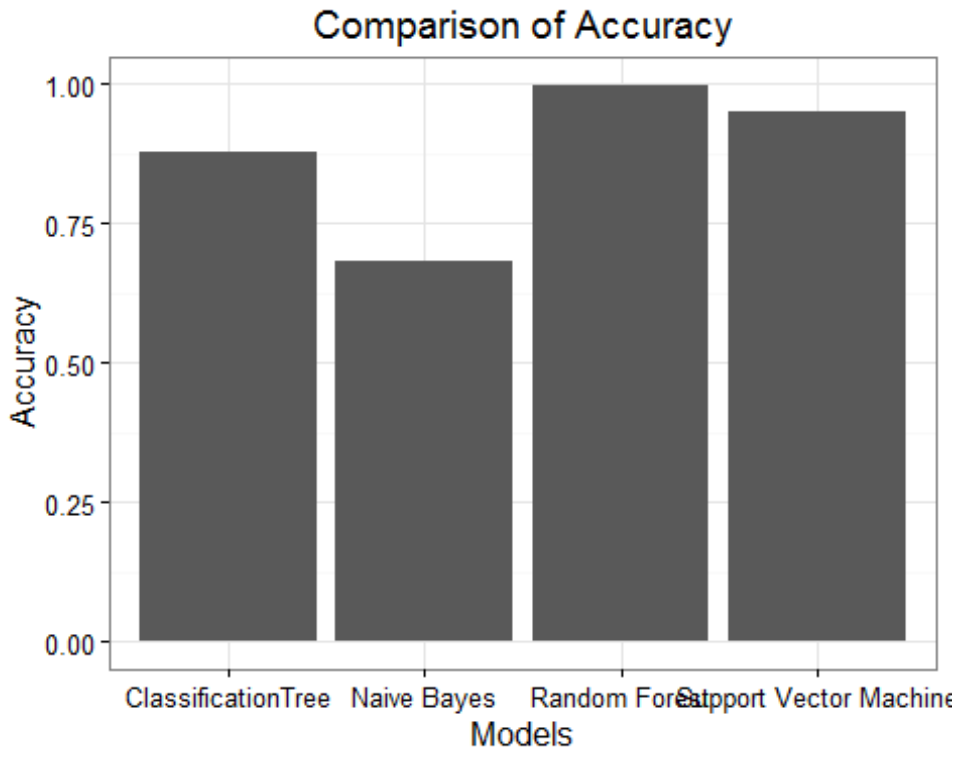
```
set.seed(32423)
svmFit<- svm(classe~.,data = training)
Prediction4<- predict(svmFit,newdata= testing,type='class')
svmAccu <- confusionMatrix(Prediction4,testing$classe)$overall[1]
svmAccu

##  Accuracy
## 0.9519116
```

The accuracy for this method is also good.

Concluding Remark

After trying four different classifiers, I have decided to use random forest as the model that can fit the data best. The accuracies of five models are plotted below.

```
Accuracy<- data.frame(Models=c('ClassificationTree','Naive
Bayes','Random Forest','Support Vector Machine'), Accuracy=
c(TreeAcu,NBAccu,rfAccu,svmAccu))
g<- ggplot(Accuracy,aes(x =
Models,y=Accuracy))+geom_bar(stat='identity')+theme_bw()+ggtitle('Compa
rison of Accuracy')
g
```



The final prediction for the 'real test set' is presented below.

```
common <- intersect(names(training), names(test))
for (p in common) {
  if (class(training[[p]]) == "factor") {
    levels(test[[p]]) <- levels(training[[p]])
  }
}
Final_Pred<- predict(rfFit,test)
Final_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
```