'Machine Learning Project'

Artem

16/12/2017

Executive Summary

In this project we'll use publicly available data set to predict type of activity performed by participants, to be more precise we'll predict how well did they execute exercies.

Background

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://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset). [1]

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

This project consists of three parts, they are:

1. Cleaning and preparing data.
2. Partitioning data sets and fitting the models.
3. Conluding on the results.

Part I

Preparing Environment and loading data

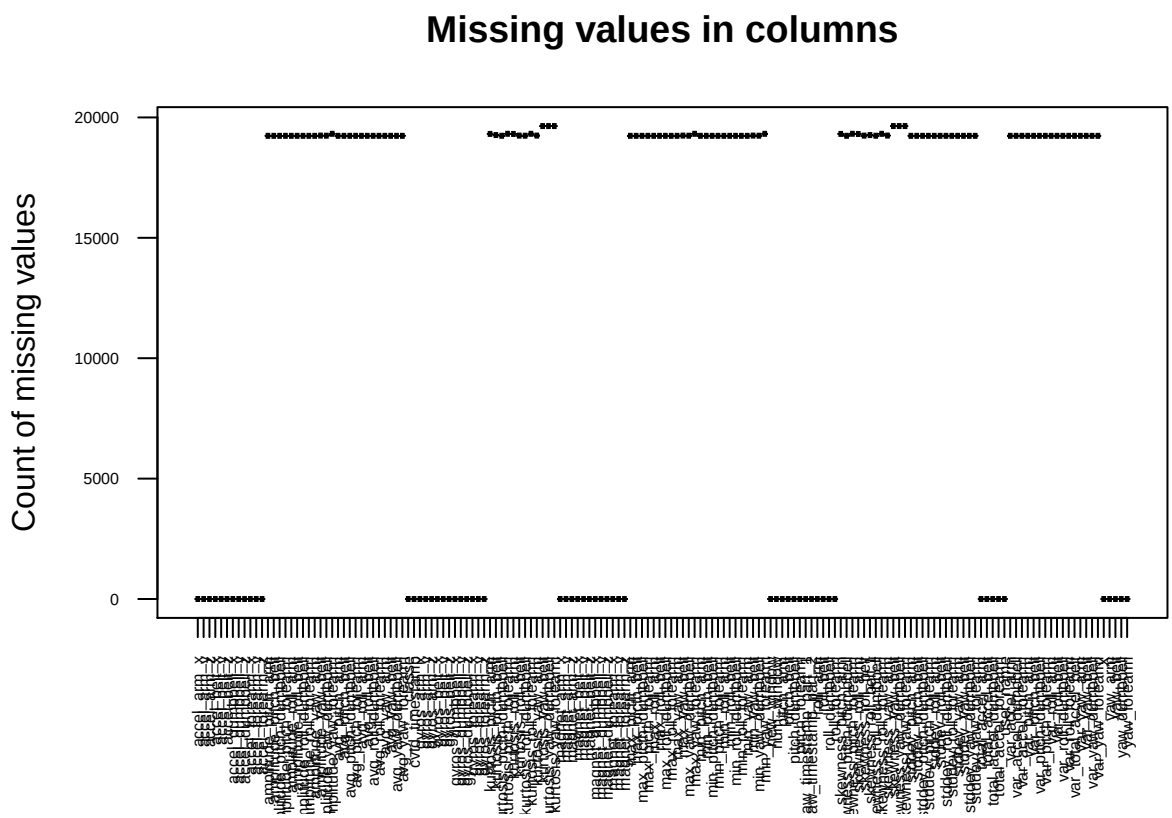
```
# Preparing global environment
rm(list = ls())
# download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv", method = 'curl'
#               destfile = "train.csv")
# download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv", method = 'curl',
#               destfile = 'test.csv')
train <- read.csv("train.csv", stringsAsFactors = F)
test <- read.csv("test.csv", stringsAsFactors = F)
# creating 'classe' variable so that we can bind train and test sets for cleaning
test$classe <- "Unknown"
```

Getting rid of unwanted columns with a lot of missing values.

Looking at data structure, we notice that there number of variables that have `char` class, but actually have numerical values. So in next round we'll trasnfrom those variables to see if there is any value for us in them.

```
# binding train and test to do data preparation and cleaning
all_data <- rbind(train, test[, -160])
all_data$user_name <- factor(all_data$user_name)
all_data$new_window <- factor(all_data$new_window)
all_data$classe <- factor(all_data$classe)
all_data$cvtd_timestamp <- as.Date(all_data$cvtd_timestamp, format = "%d/%m/%Y %H:%M")
#transforming character variables to numeric
all_data[,sapply(all_data,class) == "character"] <- apply(all_data[,sapply(all_data,class) == "characte
                                                    2, function(x) as.numeric(x))
# getting rid of columns with missing values
miss_values<- apply(all_data,2,function(x) sum(is.na(x)))
```

Plotting missing values by columns



As we can see there are quite a number of missing values in some variables, so we may remove them as probably they won't add any meaningful information.

Getting clean data set for further partitoning and model fitting

```
to_use <- names(miss_values[-which(miss_values>0)])
all_clean <- all_data[,to_use]
```

```
#dropping 'unknown' level and separating data back to train and test
all_clean <- droplevels(all_clean, exclude = 'Unknown')
train_clean <- subset(all_clean, !is.na(classe))
test_clean <- subset(all_clean, is.na(classe))
```

Part II

Setting up for model fitting

```
#partition train set for cross validation
library(caret)
```

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

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

```
inTrain <- createDataPartition(y = train_clean$classe, p = 0.7, list = F)
training <- train_clean[inTrain,]
testing <- train_clean[-inTrain,]
```

First fitting gbm model

```
# creating a cluster for parallel computing
library(parallel)
library(doParallel)
```

```
## Warning: package 'doParallel' was built under R version 3.4.2
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
cluster <- makeCluster(detectCores()-1)
registerDoParallel(cluster)

#fitting gradient boosting model
set.seed(54231)
gbm_Control <- trainControl(method = 'repeatedcv',
                            number = 10,
                            repeats = 5,
                            allowParallel = T)
#mod_gbm <- train(classe ~., data = training, method = 'gbm', verbose = F, trControl = gbm_Control)
#save(mod_gbm, file = 'mod_gbm.RData')
load('mod_gbm.RData')
gbm_pred <- predict(mod_gbm, testing)
#Checking model performance on validation set
confusionMatrix(gbm_pred, testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1673   14    0    0    0
##          B    1 1117    6    3    3
```

```
##          C    0    8 1019   11    1
##          D    0    0    1  950   11
##          E    0    0    0    0 1067
##
## Overall Statistics
##
##                Accuracy : 0.99
##                  95% CI : (0.9871, 0.9924)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9873
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9994   0.9807   0.9932   0.9855   0.9861
## Specificity            0.9967   0.9973   0.9959   0.9976   1.0000
## Pos Pred Value         0.9917   0.9885   0.9808   0.9875   1.0000
## Neg Pred Value         0.9998   0.9954   0.9986   0.9972   0.9969
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2843   0.1898   0.1732   0.1614   0.1813
## Detection Prevalence   0.2867   0.1920   0.1766   0.1635   0.1813
## Balanced Accuracy      0.9980   0.9890   0.9945   0.9915   0.9931
```

We can see that `gbm` perfoms on a decent level with 0.991 accuracy on validation set, but this is not what I expect to proceed using this model.

Next we'll try to fit Support Vector Machine algorithm

```
#fitting support vector machine algo
set.seed(9876)
trCtrl <- trainControl(method = 'repeatedcv', number = 5, repeats = 3, allowParallel = T)
# mod_svm <- train(classe ~., data = training, method = 'svmLinear',
#                  trControl = trCtrl,
#                  tuneLength = 10,
#                  verbose = F)
#save(mod_svm, file = 'mod_svm.RData')
load('mod_svm.RData')
svm_pred <- predict(mod_svm, testing)
confusionMatrix(svm_pred, testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1546  153   79   54   30
##          B   41  822   87   35   85
##          C   41   70  817  111   79
##          D   44   20   26  739   36
##          E    2   74   17   25  852
##
## Overall Statistics
```

```
##
##                Accuracy : 0.8116
##                  95% CI : (0.8013, 0.8215)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7606
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9235   0.7217   0.7963   0.7666   0.7874
## Specificity            0.9250   0.9477   0.9381   0.9744   0.9754
## Pos Pred Value         0.8303   0.7682   0.7308   0.8543   0.8784
## Neg Pred Value         0.9682   0.9342   0.9562   0.9552   0.9532
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2627   0.1397   0.1388   0.1256   0.1448
## Detection Prevalence   0.3164   0.1818   0.1900   0.1470   0.1648
## Balanced Accuracy      0.9242   0.8347   0.8672   0.8705   0.8814
```

Looking at results this is not what we expect, and probably `svm` would not be the best model selection for that problem. Again I will not proceed with testing it on whole train set.

Final fit would be the `Random Forests`.

For faster computing time we'll set up parallel computing using all cores but one.

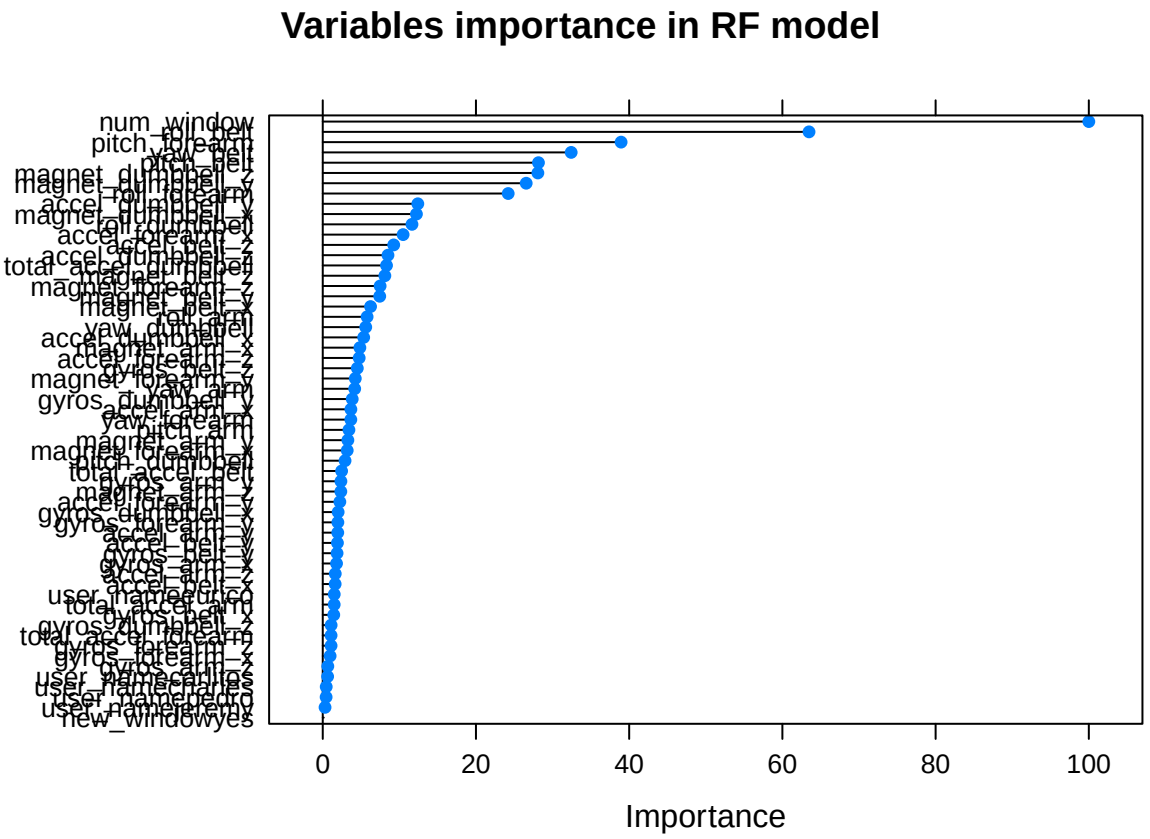
```
#fitting random forests
set.seed(12344)
fitControl <- trainControl(method = 'cv', number = 5, allowParallel = T )
#mod_rf <- train(classe ~., data =training, method = "rf", trControl = fitControl)
#save(mod_rf, file = 'mod_rf.RData')
load('mod_rf.RData')
rf_pred <- predict(mod_rf, testing)
confusionMatrix(rf_pred, testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    1    0    0    0
##          B    0 1138    0    0    0
##          C    0    0 1026    1    0
##          D    0    0    0  963    0
##          E    0    0    0    0 1082
##
## Overall Statistics
##
##                Accuracy : 0.9997
##                  95% CI : (0.9988, 1)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9996
```

```
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9991   1.0000   0.9990   1.0000
## Specificity            0.9998   1.0000   0.9998   1.0000   1.0000
## Pos Pred Value         0.9994   1.0000   0.9990   1.0000   1.0000
## Neg Pred Value         1.0000   0.9998   1.0000   0.9998   1.0000
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2845   0.1934   0.1743   0.1636   0.1839
## Detection Prevalence   0.2846   0.1934   0.1745   0.1636   0.1839
## Balanced Accuracy      0.9999   0.9996   0.9999   0.9995   1.0000
```

Excellent accuracy of 0.999, it would be interesting to see which varibales contribute the most to predictions acuracy.

Let's have a look at variables importance in our model.



That's interesting that 'num_window' scores that high, I was even considering deleting it before building a model.

We may now test it on whole training set.

```
train_pred <- predict(mod_rf, train_clean)
confusionMatrix(train_pred, train_clean$classe)
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 5580    1    0    0    0
##          B    0 3796    2    0    0
##          C    0    0 3420    3    0
##          D    0    0    0 3213    4
##          E    0    0    0    0 3603
##
## Overall Statistics
##
##                Accuracy : 0.9995
##                  95% CI : (0.9991, 0.9998)
##     No Information Rate : 0.2844
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9994
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9997   0.9994   0.9991   0.9989
## Specificity            0.9999   0.9999   0.9998   0.9998   1.0000
## Pos Pred Value         0.9998   0.9995   0.9991   0.9988   1.0000
## Neg Pred Value         1.0000   0.9999   0.9999   0.9998   0.9998
## Prevalence             0.2844   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2844   0.1935   0.1743   0.1637   0.1836
## Detection Prevalence   0.2844   0.1936   0.1744   0.1639   0.1836
## Balanced Accuracy      1.0000   0.9998   0.9996   0.9994   0.9994
```

```
# stoping cluster and deregistering cores back
stopCluster(cluster)
registerDoSEQ()
```

Seems that `randomforests` gives the best performance as expected, even with noticably greater computing time.

- Accuracy = 0.9995
- Out of sample error ~ 0.0007 (based on validation set accuracy).

So we'll proceed to final stage of our project - predicting 20 cases from test set.

```
predict(mod_rf, test_clean)
```

```
##  [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
```

Part III

Conclusion

After fitting three models, I would say that having accuracy of of 0.9995 is more than enough. So either `randomforests` would suffice our purpose of predicting the quality of performed exercises, however we have

to take into account the time it takes to fit a model.

Moving forward, I found Principal Component Analysis to be a good approach to reduce dimensionality of data and still be able to fit high performing models

***References*:**

1. 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.