Machine Learning Project

Sebastian Lewis

24/06/2021

Load libraries and datasets

```r
library(ggplot2)
library(caret)
library(caretEnsemble)
library(tidyverse)
library(visdat)
training <- read.csv("pml-training.csv")
testing <- read.csv("pml-testing.csv")
```

Spilt the training data into a train and verify dataset

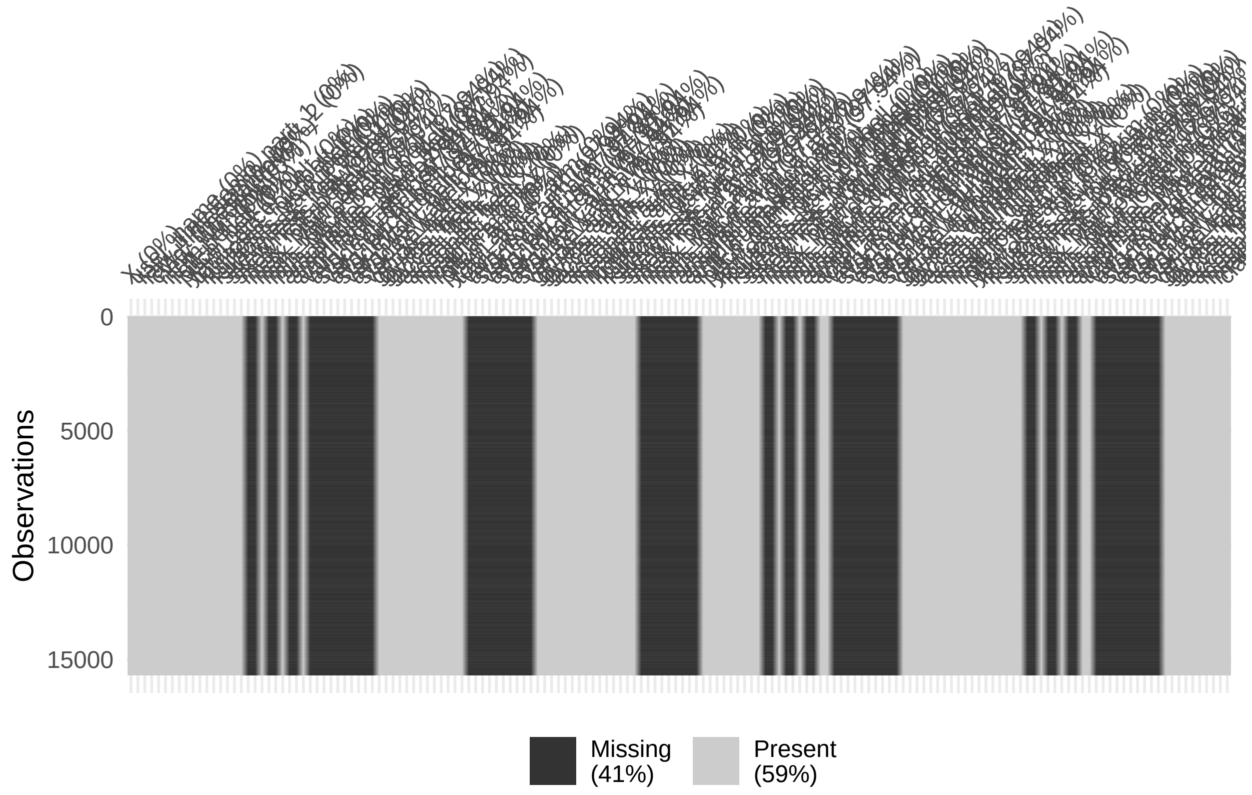
```r
inTrain <- createDataPartition(y=training$classe,p=0.8,list=FALSE)
train <- training[inTrain,]
verify <- training[-inTrain,]
```

Explore the train data

```r
dim(train);dim(verify); dim(testing)
head(train)
```

The data contains 160 variables. The test set has nearly 14,000 cases. Lets conduct a high level analysis of the data to see if any is missing.

```r
vis_miss(train,warn_large_data = FALSE)
```



### Pre-process data

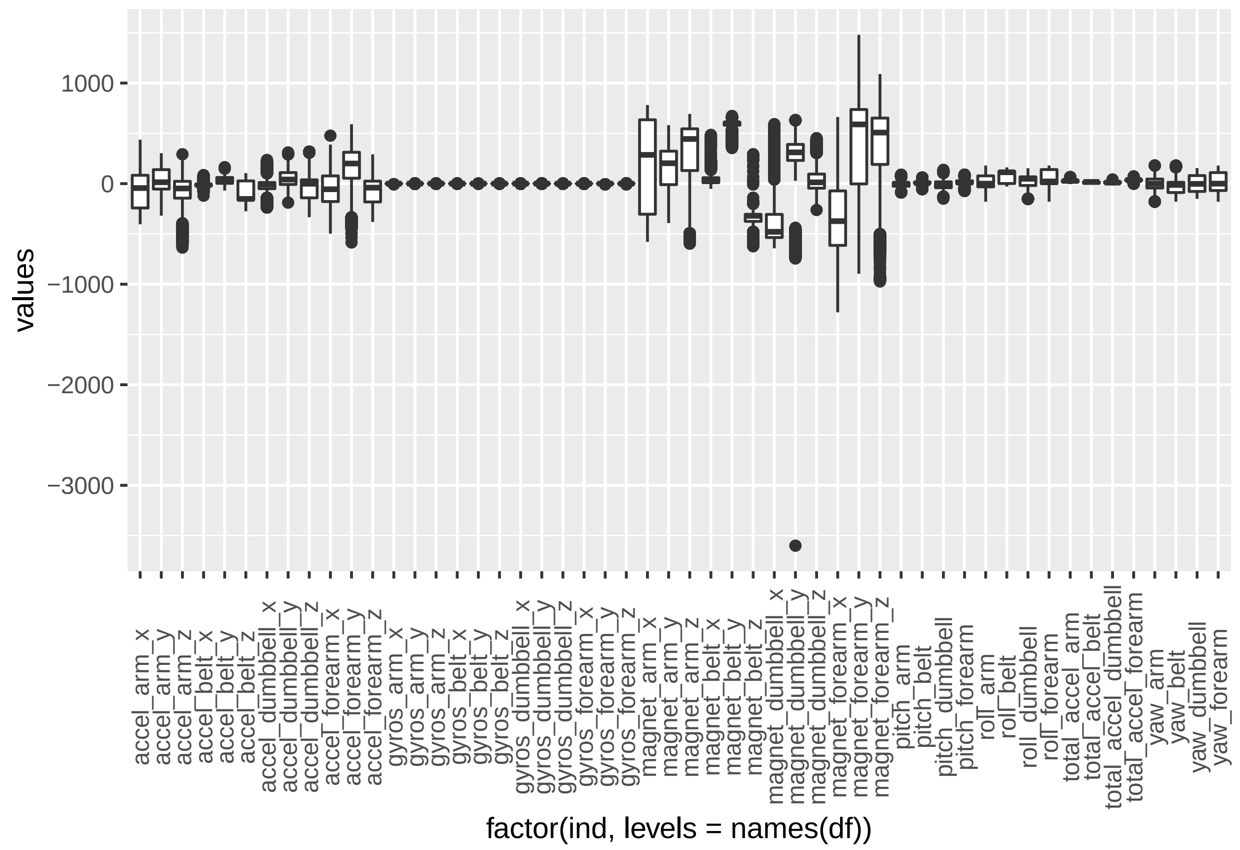
41% of the data is missing. This appears to be concentrated in a number of columns. Lets clean all the datasets, removing the missing data as well as variables like like user name and timestamp and window number which do not give information related to the accelerometers.

```
dfs <- list(train,verify,testing)
sets <- lapply(dfs,function(x){
  x[x==""] <- NA
  x <- x[, colSums(is.na(x)) == 0]
  x <- select(x,-c(X,user_name,raw_timestamp_part_1,raw_timestamp_part_2,cvtd_timestamp,new_window,num_
  return (x)
})
train <- sets[[1]]
dim(train)
```

Removing the columns leaves 53 predictor variables that hold data on the movement and orientation of the accelerometers on the belt, arm, dumbbell, forearm. There are variables for roll, pitch, yaw and for the x,y and z orientation of the gyros, accel and magnet sensors.

Explore predictor variables

```
df <- select(train,-c(classe)) # remove classe for plotting
df <- df[,order(names(df))] # order alphabetically
ggplot(stack(df), aes(x = factor(ind, levels = names(df)), y = values)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5))
```



The box plot shows there is considerable variance between the groups of variables. Accel and magnet variables show more dispersion than other groups of variables, notably gyros, which appears to have the least variance. Normalizing or standardizing our data might improve the model.

Build model

A random forests algorithm was used to predict `classe`. Random forests are very good at classification problems and are good handling datasets that contain large numbers of cases and variables.

I tried both the `rf` and `ranger` implementation of random forests. I decided on using the `ranger` algorithm which is computationally faster and generally gave slightly higher in sample accuracy than `rf`.

I standardised the data by centring and scaling it using the `preProcess` function, which gave a slightly better in sample error. I experimented with tuning the `ranger` settings like `mtry` but any improvements in the in sample accuracy were very slight, and it it proved hard to find a reliable test to show these improvements were statistically significant. I tried two different resampling methods to cross validate the model using the `trainControl` function:

- Repeated cross validation using `repeatedcv` with 5 repetitions of 2 fold and 5 repetitions of 5 folds.
- Using the out-of-bag `oob` error estimate. This is estimated internally by the random forests algorithm using data that is left out of the bootstrap samples that it uses in the construction of the trees.

I decided to use the `oob` error estimate to cross validate my model. Trials showed that it gave a very similar in sample error estimate to `repeatedcv` but is computationally much faster than `repeatedcv` which takes very long a very long time to complete.

The final model is shown below:

```
set.seed(3355)
ctrloob = trainControl( method = "oob") #out-of-bag (oob) error estimation
system.time(Mdl <- train(classe~.,data=train, method='ranger',preProcess = c("center", "scale"),
              trControl = ctrloob)) #$ranger2
```

```
##    user  system elapsed
## 346.180   2.151  52.683
```

```
Mdl
```

```
## Random Forest
##
## 15699 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing: centered (52), scaled (52)
## Resampling results across tuning parameters:
##
##   mtry  splitrule   Accuracy   Kappa
##    2    gini        0.9947130  0.9933121
##    2    extratrees  0.9932480  0.9914590
##   27    gini        0.9946493  0.9932318
##   27    extratrees  0.9963692  0.9954074
##   52    gini        0.9867507  0.9832371
##   52    extratrees  0.9964329  0.9954880
##
## Tuning parameter 'min.node.size' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 52, splitrule = extratrees
##  and min.node.size = 1.
```

```
#in sample error
getMetric(Mdl)
```

```
## [1] 0.9964329
```

Estimate out of sample error

I estimated the out of sample error by testing the model on the `verify` data set that was not used to train the model. The result is given in the confusion matrix below, with `Accuracy` representing the out of sample error.

```
pred <- predict(Mdl,verify)
confusionMatrix(factor(verify$classe),pred)$table
```

```
##            Reference
## Prediction    A    B    C    D    E
##          A 1115    0    0    0    1
##          B    2  757    0    0    0
##          C    0    1  681    2    0
##          D    0    0    9  634    0
##          E    0    0    0    1  720
```

```
#out of sample error
confusionMatrix(factor(verify$classe),pred)$overall['Accuracy']
```

```
##  Accuracy
## 0.9959215
```

The out of sample error is slightly lower than the in sample error. This is to be expected as the model tends to the tune itself to the training set and overfit to the training data.