# Practical Machine Learning Project

Anh

2/4/2022

## Background

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit 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.

## Data processing

### Importing data

```
# Load necessary packages
library(caret)
library(rattle)
library(rpart)
library(rpart.plot)
library(randomForest)
library(repmis)
```

```
# Load data locally
df_train = read.csv("pml-training.csv", na.strings = c("NA", ""))
df_test = read.csv("pml-testing.csv", na.strings = c("NA", ""))
dim(df_train)
```

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

```
dim(df_test)
```

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

From the dim functions, we can see that the training dataset has 19622 observations and 160 variables, and the testing data set contains 20 observations and the same variables as the training set. We are trying to predict the outcome of the variable `classe` in the training set.

### Cleaning data

```r
# delete columns of the training set that contain any missing values
train = df_train[, colSums(is.na(df_train)) == 0]
test = df_test[, colSums(is.na(df_test)) == 0]

# delete the first seven weak predictors
train = train[, -c(1:7)]
test = test[, -c(1:7)]
```

### Spliting data

To get out-of-sample errors, we split the cleaned training set `train` into a training set (`train_data`, 70%) for prediction and a validation set (`valid_data` 30%) to compute the out-of-sample errors.

```r
set.seed(7826)
inTrain <- createDataPartition(train$classe, p = 0.7, list = FALSE)
train_data <- train[inTrain, ]
valid_data <- train[-inTrain, ]
```

## Prediction algorithms

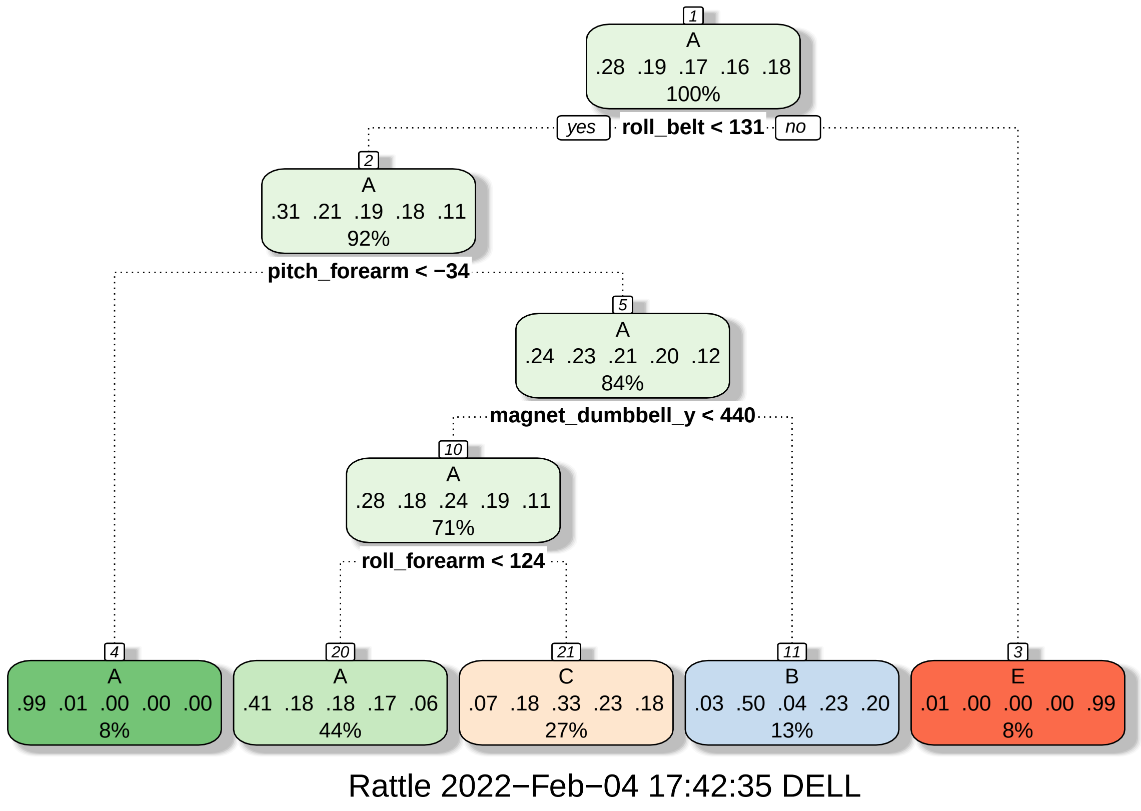
### Classification trees

Since data transformations are not necessarily important in non-linear models like classification trees, we do not transform any variables.

```r
control <- trainControl(method = "cv", number = 5)
fit_rpart <- train(classe ~ ., data = train_data, method = "rpart",
                   trControl = control)
print(fit_rpart, digits = 4)
```

```
## CART
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10990, 10990, 10990, 10989, 10989
## Resampling results across tuning parameters:
##
##   cp       Accuracy  Kappa
##   0.03102  0.5260    0.38038
##   0.05954  0.3935    0.16982
##   0.11586  0.3168    0.04946
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.03102.
```

```
fancyRpartPlot(fit_rpart$finalModel)
```



```
# predict outcomes using validation set
predict_rpart <- predict(fit_rpart, valid_data)
# Show prediction result
(conf_rpart <- confusionMatrix(as.factor(valid_data$classe), predict_rpart))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1510   27  134    0    3
##          B  464  410  265    0    0
##          C  467   37  522    0    0
##          D  432  163  369    0    0
##          E  164  144  293    0  481
##
## Overall Statistics
##
##                Accuracy : 0.4967
##                  95% CI : (0.4838, 0.5095)
##     No Information Rate : 0.5161
##     P-Value [Acc > NIR] : 0.9986
##
##                   Kappa : 0.3425
```

```
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.4972  0.52497   0.3298       NA  0.99380
## Specificity            0.9424  0.85717   0.8828   0.8362  0.88872
## Pos Pred Value         0.9020  0.35996   0.5088       NA  0.44455
## Neg Pred Value         0.6374  0.92183   0.7816       NA  0.99938
## Prevalence             0.5161  0.13271   0.2690   0.0000  0.08224
## Detection Rate         0.2566  0.06967   0.0887   0.0000  0.08173
## Detection Prevalence   0.2845  0.19354   0.1743   0.1638  0.18386
## Balanced Accuracy      0.7198  0.69107   0.6063       NA  0.94126
```

```
(accuracy_rpart <- conf_rpart$overall[1])
```

```
##  Accuracy
## 0.4966865
```

Accuracy rate is 0.5, and so the out-of-sample error rate is 0.5. Therefore, using classification tree does not predict the outcome `classe` very well.

## Random forests

```
fit_rf <- train(classe ~ ., data = train_data, method = "rf",
                   trControl = control)
print(fit_rf, digits = 4)
```

```
## Random Forest
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10991, 10990, 10990, 10988, 10989
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa
##    2    0.9913    0.9889
##   27    0.9921    0.9900
##   52    0.9840    0.9797
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 27.
```

```
# predict outcomes using validation set
predict_rf <- predict(fit_rf, valid_data)
# Show prediction result
(conf_rf <- confusionMatrix(as.factor(valid_data$classe), predict_rf))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1671    2    0    0    1
##          B    4 1134    1    0    0
##          C    0    8 1013    5    0
##          D    0    0   16  947    1
##          E    0    1    1    9 1071
##
## Overall Statistics
##
##                Accuracy : 0.9917
##                  95% CI : (0.989, 0.9938)
##     No Information Rate : 0.2846
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9895
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9976   0.9904   0.9825   0.9854   0.9981
## Specificity            0.9993   0.9989   0.9973   0.9965   0.9977
## Pos Pred Value         0.9982   0.9956   0.9873   0.9824   0.9898
## Neg Pred Value         0.9991   0.9977   0.9963   0.9972   0.9996
## Prevalence             0.2846   0.1946   0.1752   0.1633   0.1823
## Detection Rate         0.2839   0.1927   0.1721   0.1609   0.1820
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9984   0.9947   0.9899   0.9910   0.9979
```

```
(accuracy_rf <- conf_rf$overall[1])
```

```
##  Accuracy
## 0.9916737
```

The accuracy rate is 0.991, and so the out-of-sample error rate is 0.009. This may be due to the fact that many predictors in the dataset are highly correlated. Random forests choose a subset of predictors at each split and decorrelate the trees. This leads to high accuracy, although this algorithm is sometimes difficult to interpret and computationally inefficient.

## Prediction on test set

```
(predict(fit_rf, test))
```

```
##  [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
```