# Weight Lift Quality Prediction

Practical Machine Learning Course Project

*Le, Duc Anh*

*September, 2018*

**Abstract**

This project examines different models used for predicting the quality of exercises done by a group of 6 participants. Three different classification models including trees, random forest, and gradient boosted models. It is found that the random forest and stacked predictors would be the best approaches in qualifying the activities.

# Data Processing

## Getting Data

The training and testing data sets are downloaded from the respective URL.

```
##Loading training set
URL <-"https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
filename <-"pml-training.csv"
if (!file.exists(filename)) {
    download.file(URL,filename,method="curl")
}

##Loading testing set
URL <-"https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
filename <-"pml-testing.csv"
if (!file.exists(filename)) {
    download.file(URL,filename,method="curl")
}

training<-read.csv("pml-training.csv")
testing<-read.csv("pml-testing.csv")
```

The structure of the data collected are shown below:

```
str(training)
```

```
## 'data.frame':    19622 obs. of  160 variables:
##  $ X                       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ user_name               : Factor w/ 6 levels "adelmo","carlitos",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ raw_timestamp_part_1    : int  1323084231 1323084231 1323084231 1323084232 1323084232 1323084232 1
##  $ raw_timestamp_part_2    : int  788290 808298 820366 120339 196328 304277 368296 440390 484323 4844
##  $ cvtd_timestamp          : Factor w/ 20 levels "02/12/2011 13:32",..: 9 9 9 9 9 9 9 9 9 9 ...
##  $ new_window              : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ num_window              : int  11 11 11 12 12 12 12 12 12 12 ...
##  $ roll_belt               : num  1.41 1.41 1.42 1.48 1.48 1.45 1.42 1.42 1.43 1.45 ...
##  $ pitch_belt              : num  8.07 8.07 8.07 8.05 8.07 8.06 8.09 8.13 8.16 8.17 ...
##  $ yaw_belt                : num  -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 ...
##  $ total_accel_belt        : int  3 3 3 3 3 3 3 3 3 3 ...
##  $ kurtosis_roll_belt      : Factor w/ 397 levels "","-0.016850",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_picth_belt     : Factor w/ 317 levels "","-0.021887",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_yaw_belt       : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_belt      : Factor w/ 395 levels "","-0.003095",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_belt.1    : Factor w/ 338 levels "","-0.005928",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_yaw_belt       : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ max_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_belt          : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_belt            : Factor w/ 68 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ min_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_belt          : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_belt            : Factor w/ 68 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ amplitude_roll_belt     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_pitch_belt    : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_yaw_belt      : Factor w/ 4 levels "","#DIV/0!","0.00",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ var_total_accel_belt    : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_roll_belt        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_pitch_belt       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_yaw_belt         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ gyros_belt_x            : num  0 0.02 0 0.02 0.02 0.02 0.02 0.02 0.02 0.03 ...
##  $ gyros_belt_y            : num  0 0 0 0 0.02 0 0 0 0 0 ...
##  $ gyros_belt_z            : num  -0.02 -0.02 -0.02 -0.03 -0.02 -0.02 -0.02 -0.02 -0.02 0 ...
##  $ accel_belt_x            : int  -21 -22 -20 -22 -21 -21 -22 -22 -20 -21 ...
##  $ accel_belt_y            : int  4 4 5 3 2 4 3 4 2 4 ...
##  $ accel_belt_z            : int  22 22 23 21 24 21 21 21 24 22 ...
##  $ magnet_belt_x           : int  -3 -7 -2 -6 -6 0 -4 -2 1 -3 ...
##  $ magnet_belt_y           : int  599 608 600 604 600 603 599 603 602 609 ...
##  $ magnet_belt_z           : int  -313 -311 -305 -310 -302 -312 -311 -313 -312 -308 ...
##  $ roll_arm                : num  -128 -128 -128 -128 -128 -128 -128 -128 -128 -128 ...
##  $ pitch_arm               : num  22.5 22.5 22.5 22.1 22.1 22 21.9 21.8 21.7 21.6 ...
##  $ yaw_arm                 : num  -161 -161 -161 -161 -161 -161 -161 -161 -161 -161 ...
##  $ total_accel_arm         : int  34 34 34 34 34 34 34 34 34 34 ...
##  $ var_accel_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_roll_arm         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_pitch_arm        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_yaw_arm             : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_yaw_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_yaw_arm             : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ gyros_arm_x             : num  0 0.02 0.02 0.02 0 0.02 0 0.02 0.02 0.02 ...
##  $ gyros_arm_y             : num  0 -0.02 -0.02 -0.03 -0.03 -0.03 -0.03 -0.02 -0.03 -0.03 ...
##  $ gyros_arm_z             : num  -0.02 -0.02 -0.02 0.02 0 0 0 0 -0.02 -0.02 ...
##  $ accel_arm_x             : int  -288 -290 -289 -289 -289 -289 -289 -289 -288 -288 ...
##  $ accel_arm_y             : int  109 110 110 111 111 111 111 111 109 110 ...
##  $ accel_arm_z             : int  -123 -125 -126 -123 -123 -122 -125 -124 -122 -124 ...
##  $ magnet_arm_x            : int  -368 -369 -368 -372 -374 -369 -373 -372 -369 -376 ...
##  $ magnet_arm_y            : int  337 337 344 344 337 342 336 338 341 334 ...
##  $ magnet_arm_z            : int  516 513 513 512 506 513 509 510 518 516 ...
##  $ kurtosis_roll_arm       : Factor w/ 330 levels "","-0.02438",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_picth_arm      : Factor w/ 328 levels "","-0.00484",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_yaw_arm        : Factor w/ 395 levels "","-0.01548",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_arm       : Factor w/ 331 levels "","-0.00051",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_pitch_arm      : Factor w/ 328 levels "","-0.00184",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_yaw_arm        : Factor w/ 395 levels "","-0.00311",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ max_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_arm             : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_arm             : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_roll_arm      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_pitch_arm     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_yaw_arm       : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ roll_dumbbell           : num  13.1 13.1 12.9 13.4 13.4 ...
##  $ pitch_dumbbell          : num  -70.5 -70.6 -70.3 -70.4 -70.4 ...
##  $ yaw_dumbbell            : num  -84.9 -84.7 -85.1 -84.9 -84.9 ...
##  $ kurtosis_roll_dumbbell  : Factor w/ 398 levels "","-0.0035","-0.0073",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_picth_dumbbell : Factor w/ 401 levels "","-0.0163","-0.0233",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_yaw_dumbbell   : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_dumbbell  : Factor w/ 401 levels "","-0.0082","-0.0096",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_pitch_dumbbell : Factor w/ 402 levels "","-0.0053","-0.0084",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_yaw_dumbbell   : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ max_roll_dumbbell       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_dumbbell        : Factor w/ 73 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ min_roll_dumbbell       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_dumbbell        : Factor w/ 73 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ amplitude_roll_dumbbell : num  NA NA NA NA NA NA NA NA NA NA ...
##   [list output truncated]
```

## Cleaning Data

A few steps are done on the data sets in order to eliminate unnecessary information.

Firstly, the first 7 variables are eliminated since they only serve the purpose of identification and do not offer any prediction power:

```
training<-training[,-c(1:7)]
testing<-testing[,-c(1:7)]
```

Secondly, as noted from the structure of the, some of the varibles only contain mostly “NA” or " “, indicating that a large proportion of data is missing for these variables. As a results, these variables do not have sufficinet prediction power to be included for the modelling.

```
naMean<-colSums(is.na(training))/nrow(training)
training<-training[,naMean<0.5]
testing<-testing[,naMean<0.5]

emptyMean<-colSums(training=="")/nrow(training)
```

```
training<-training[,emptyMean<0.5]
testing<-testing[,emptyMean<0.5]
```

# Model Building

The following libraries are loaded into R for this analysis:

```
require(caret)
```

```
## Warning: package 'caret' was built under R version 3.4.4

## Warning: package 'ggplot2' was built under R version 3.4.4

## Warning in as.POSIXlt.POSIXct(Sys.time()): unknown timezone 'default/Asia/
## Ho_Chi_Minh'
```

```
require(rattle)
```

```
## Warning: package 'rattle' was built under R version 3.4.4
```

```
require(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.4.4
```

```
require(gbm)
require(rpart)
```

## Training and Cross-Validation

The training data is split into training and cross-validation sets. The trainData set is used for model building and the cvData is used for out-of-sample validation and selection of the models. For this study, 80% of the training set is assigned to trainData while the remaining samples are assigned to cvData.

```
set.seed(180792)
inTrain<-createDataPartition(y=training$classe,p=0.8,list=FALSE)
trainData<-training[inTrain,]
cvData<-training[-inTrain,]
```

## Model Building

For the purpose of this study, three primary models are constructed, including classfication tree, random forest and gradient boosted model. A fourth model is then derived as a stacked predictor based on the above-mentioned models. The accuracy of the four models are then evaluated against the cvData and the model with the best accuracy is used to evaluate the testing set (preTest).

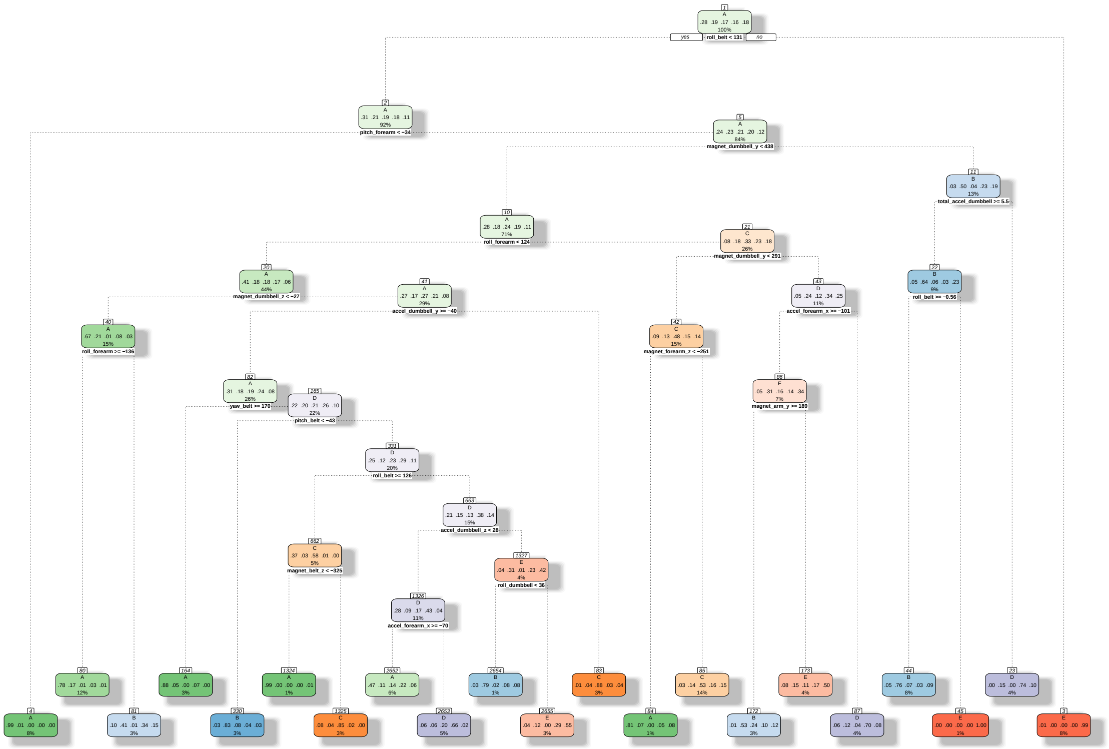
### Classification tree

```

Fit_rpart<-rpart(classe~.,
                 data=trainData)
```

The classification models are visualised as below:

```r
fancyRpartPlot(Fit_rpart)
```



Rattle 2018−Sep−03 06:33:10 adrianle

### Random forest

The resulted model created from the random forest method is shown below:

```r
Fit_rf<-randomForest(classe~.,
              data=trainData,
              verbose=FALSE)
print(Fit_rf)
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = trainData, verbose = FALSE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.39%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 4461    3    0    0    0 0.000672043
## B   10 3026    2    0    0 0.003949967
## C    0   15 2722    1    0 0.005843682
## D    0    0   22 2549    2 0.009327633
## E    0    0    1    6 2879 0.002425502
```

### Generalized Boosted Model

```
Fit_gbm<-train(classe~.,
               method="gbm",
               data=trainData,
               verbose=FALSE)
print(Fit_gbm)
```

```
## Stochastic Gradient Boosting
##
## 15699 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 15699, 15699, 15699, 15699, 15699, 15699, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7526795  0.6863592
##   1                  100      0.8203726  0.7725428
##   1                  150      0.8536314  0.8147017
##   2                   50      0.8550202  0.8162510
##   2                  100      0.9053658  0.8801676
##   2                  150      0.9283239  0.9092556
##   3                   50      0.8951310  0.8671522
##   3                  100      0.9394596  0.9233589
##   3                  150      0.9577954  0.9465819
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150,
##  interaction.depth = 3, shrinkage = 0.1 and n.minobsinnode = 10.
```

### Stacked Model

```
pred_rpart <- predict(Fit_rpart,newdata = trainData,type="class")
pred_rf <- predict(Fit_rf,newdata = trainData,type="class")
pred_gbm <- predict(Fit_gbm,newdata = trainData,n.trees=100)
data_stacked <- data.frame(classe=trainData$classe,
                           pred_rpart,
                           pred_rf,
                           pred_gbm)
Fit_Stacked <- randomForest(classe~.,data=data_stacked)
print(Fit_Stacked)
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = data_stacked)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 0.02%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 4464    0    0    0    0 0.0000000000
## B    0 3038    0    0    0 0.0000000000
## C    0    0 2737    0    1 0.0003652301
## D    1    1    0 2571    0 0.0007773028
## E    0    0    0    0 2886 0.0000000000
```

## Model Evaluation

```
test_rpart <- predict(Fit_rpart, newdata = cvData,type="class")
test_rf <- predict(Fit_rf, newdata = cvData,type="class")
test_gbm <- predict(Fit_gbm, newdata = cvData)
test_stacked <- data.frame(classe=cvData$classe,
                           test_rpart,
                           test_rf,
                           test_gbm)
names(test_stacked)<-names(data_stacked)
test_stacked <- predict(Fit_Stacked, newdata = test_stacked,type="class")
```

```
confusionMatrix(test_rpart,cvData$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A 985 122  42  67  17
##          B  37 452  61  56  54
##          C  39  88 521  93  88
##          D  33  59  37 342  32
##          E  22  38  23  85 530
##
## Overall Statistics
##
##                Accuracy : 0.7214
##                  95% CI : (0.7071, 0.7354)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6462
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8826   0.5955   0.7617  0.53188   0.7351
## Specificity            0.9116   0.9343   0.9049  0.95091   0.9475
## Pos Pred Value         0.7989   0.6848   0.6285  0.67992   0.7593
## Neg Pred Value         0.9513   0.9059   0.9473  0.91199   0.9408
## Prevalence             0.2845   0.1935   0.1744  0.16391   0.1838
## Detection Rate         0.2511   0.1152   0.1328  0.08718   0.1351
## Detection Prevalence   0.3143   0.1682   0.2113  0.12822   0.1779
## Balanced Accuracy      0.8971   0.7649   0.8333  0.74140   0.8413
```

```
confusionMatrix(test_rf,cvData$classe)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1115    5    0    0    0
##          B    0  753    5    0    0
##          C    0    1  677    4    0
##          D    0    0    2  639    1
##          E    1    0    0    0  720
## 
## Overall Statistics
## 
##                Accuracy : 0.9952
##                  95% CI : (0.9924, 0.9971)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
## 
##                   Kappa : 0.9939
##  Mcnemar's Test P-Value : NA
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9991   0.9921   0.9898   0.9938   0.9986
## Specificity            0.9982   0.9984   0.9985   0.9991   0.9997
## Pos Pred Value         0.9955   0.9934   0.9927   0.9953   0.9986
## Neg Pred Value         0.9996   0.9981   0.9978   0.9988   0.9997
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2842   0.1919   0.1726   0.1629   0.1835
## Detection Prevalence   0.2855   0.1932   0.1738   0.1637   0.1838
## Balanced Accuracy      0.9987   0.9953   0.9941   0.9964   0.9992
```

```
confusionMatrix(test_gbm,cvData$classe)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1096   19    0    0    0
##          B   10  720   26    4    5
##          C    7   17  651   21   10
##          D    3    2    7  609   11
##          E    0    1    0    9  695
## 
## Overall Statistics
## 
##                Accuracy : 0.9613
##                  95% CI : (0.9547, 0.9671)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.951
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9821   0.9486   0.9518   0.9471   0.9639
## Specificity            0.9932   0.9858   0.9830   0.9930   0.9969
## Pos Pred Value         0.9830   0.9412   0.9221   0.9636   0.9858
## Neg Pred Value         0.9929   0.9877   0.9897   0.9897   0.9919
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2794   0.1835   0.1659   0.1552   0.1772
## Detection Prevalence   0.2842   0.1950   0.1800   0.1611   0.1797
## Balanced Accuracy      0.9877   0.9672   0.9674   0.9701   0.9804
```

```r
confusionMatrix(test_stacked,cvData$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1115    5    0    0    0
##          B    0  752    5    0    0
##          C    0    1  677    4    0
##          D    0    0    2  639    1
##          E    1    1    0    0  720
##
## Overall Statistics
##
##                Accuracy : 0.9949
##                  95% CI : (0.9921, 0.9969)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9936
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9991   0.9908   0.9898   0.9938   0.9986
## Specificity            0.9982   0.9984   0.9985   0.9991   0.9994
## Pos Pred Value         0.9955   0.9934   0.9927   0.9953   0.9972
## Neg Pred Value         0.9996   0.9978   0.9978   0.9988   0.9997
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2842   0.1917   0.1726   0.1629   0.1835
## Detection Prevalence   0.2855   0.1930   0.1738   0.1637   0.1840
## Balanced Accuracy      0.9987   0.9946   0.9941   0.9964   0.9990
```

From the confusion matrix, the random forest and stacked predictor methods provide the best model for the validation set prediction with accuracy of 0.9952 and 0.9949 respectively. Hence these two models are used for predicting the testing data set.

## Testing Data Prediction

### Prediction using random forest model

```
testing_rf<-predict(Fit_rf, newdata = testing)
```

### Prediction using stacked predictor

```
final_rpart <- predict(Fit_rpart, newdata = testing,type="class")
final_rf <- predict(Fit_rf, newdata = testing,type="class")
final_gbm <- predict(Fit_gbm, newdata = testing)
final_stacked <- data.frame(final_rpart,
                            final_rf,
                            final_gbm)
names(final_stacked)<-names(data_stacked[,-1])
testing_stacked<-predict(Fit_Stacked, newdata = final_stacked,type="class")
```

### Prediction results

```
Test_Result<-data.frame(ID=1:20,
                        RandomForest = testing_rf,
                        Stacked = testing_stacked,
                        Matched = testing_rf == testing_stacked)
print(Test_Result)
```

```
##    ID RandomForest Stacked Matched
## 1   1            B       B    TRUE
## 2   2            A       A    TRUE
## 3   3            B       B    TRUE
## 4   4            A       A    TRUE
## 5   5            A       A    TRUE
## 6   6            E       E    TRUE
## 7   7            D       D    TRUE
## 8   8            B       B    TRUE
## 9   9            A       A    TRUE
## 10 10            A       A    TRUE
## 11 11            B       B    TRUE
## 12 12            C       C    TRUE
## 13 13            B       B    TRUE
## 14 14            A       A    TRUE
## 15 15            E       E    TRUE
## 16 16            E       E    TRUE
## 17 17            A       A    TRUE
## 18 18            B       B    TRUE
## 19 19            B       B    TRUE
## 20 20            B       B    TRUE
```

The two models seem to provide matched results on the prediction. We could hence use this result as the final prediction on the testing data.