Coursera Practical Machine Learning

Peer-graded Assignment: Prediction Assignment Writeup

Gerard van Meurs

January 2, 2019

Executive summary

The paper is the final report of a peer-graded assignment of the Coursera Practical Machine Learning course as part of the Data Science specialization. In this paper several Machine Learning Algorithms are applied to predict whether weight lift exercises were executed exactly conform the specifications of the exercise or whether these were executed to the specifications of some common mistakes. In order for the predictions to be as accurate as possible, some preparation ot the data was involved. Data was divided in een trainingset and a testset and five different models were fitted using crossvalidation. The Random Forest model turned out te be the most accurate of the five that were tested: .9983 accuracy on the testset. This final model was also used to predict the classe of the 20 extra testcases.

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).

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

If you want to read more: 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.

Reading and preparing the data

```
training <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv")
testing <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv")
```

Exploring the data

```
dim(training)
```

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

```
# str(training)
table(training$classe)
```

```
##
##    A    B    C    D    E
## 5580 3797 3422 3216 3607
```

The training dataset consists of 19622 observations on 160 variables. All observations have to do with weight lifting execises of 6 young adults: exactly according to the specifications (Classe = A), and four other manners corresponding to common misstakes (classe = B, C, D or E). The goal of this project is to predict the manner in which exercises were performed (classe). The distribution of classe is shown above.

The training dataset is split in a training-set and a validation-set: 70/30. The training-set is used to build models and the validation-set is used to validate the models. The testing-dataset will only be used to generate the quiz results.

```
inTrain <- createDataPartition(training$classe, p = 0.70, list = FALSE)
trainSet <- training[inTrain, ]
testSet <- training[-inTrain, ]
dim(trainSet)
```

```
## [1] 13737   160
```

```
dim(testSet)
```

```
## [1] 5885  160
```

```
table(trainSet$classe); prop.table(table(trainSet$classe))
```

```
##
##    A    B    C    D    E
## 3906 2658 2396 2252 2525
```

```
##
##         A         B         C         D         E
## 0.2843416 0.1934920 0.1744195 0.1639368 0.1838101
```

```
table(testSet$classe); prop.table(table(testSet$classe))
```

```
##
##    A    B    C    D    E
## 1674 1139 1026  964 1082
```

```
##
##         A         B         C         D         E
## 0.2844520 0.1935429 0.1743415 0.1638063 0.1838573
```

Selecting variables: removing identifying variables

In a first step all identifying variables are removed:

```
# removing identification variables (1:5)
trainSet <- trainSet[ , -(1:5)]
```

```
testSet <- testSet[ , -(1:5)]
dim(trainSet)
```

```
## [1] 13737   155
```

```
# dim(testSet)
```

In this step 5 variables are removed, leaving 155 of the 160 original variables in the dataframe.

Selecting variables: removing missing values

In a next step, missing values are evaluated. Although best practice would be to look at the underlying process of missingness, in this case only the amount of missingness is considered. All variables with 90% or more missing values are removed form the training- and the testset:

```
# removing variables with 90% or more missing values (NA)
nnaVars <- sapply(trainSet, function(x) mean(is.na(x))) > 0.90
nnaVars <- which(nnaVars)
trainSet <- trainSet[ , -nnaVars]
testSet <- testSet[, -nnaVars]
dim(trainSet)
```

```
## [1] 13737    88
```

```
# dim(testset)
```

This step removes 66 of the 155 remaining variables, so this results in a dataframe of 88 variables.

Selecting variables: removing (near) zero variance variables

In the next step variance of the predictors is considered. All variables with zero or near zero variance are removed from both the training- and the testset:

```
# removing variables with Near Zero Variance
nzVars <- nearZeroVar(trainSet)
trainSet <- trainSet[ , -nzVars]
testSet <- testSet[, -nzVars]
dim(trainSet)
```

```
## [1] 13737    54
```

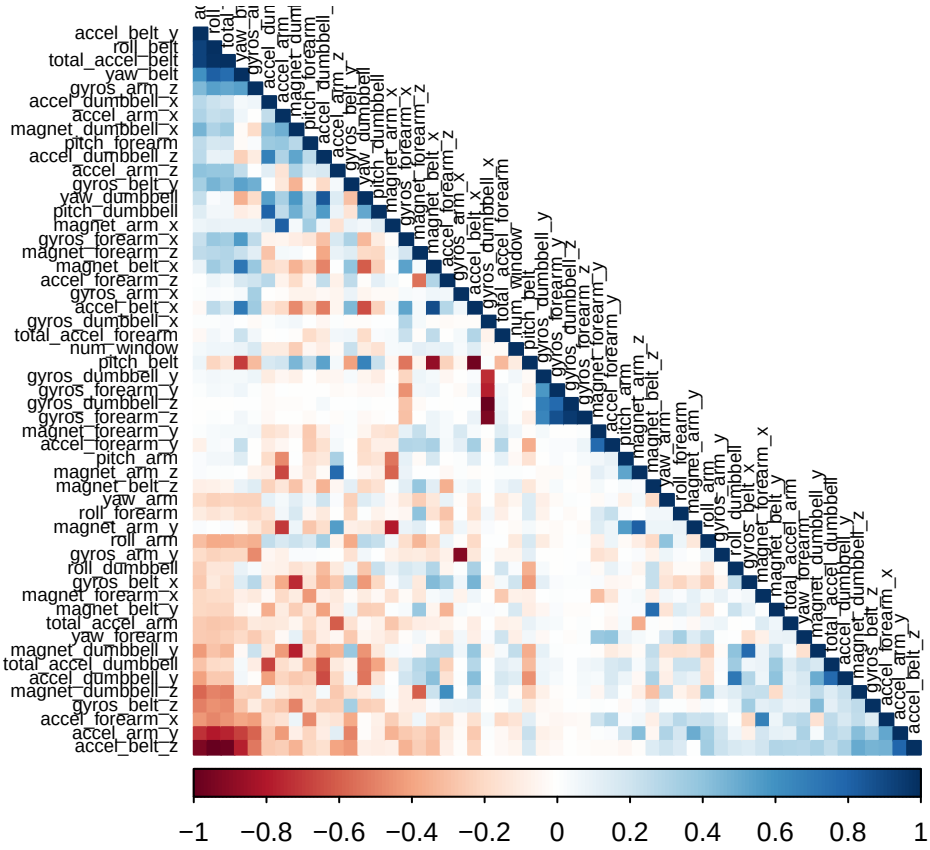
```
# dim(testSet)
```

This step removed 34 of the remaining 88 variables, resulting in a datafram of only 54 variables.

Selecting variables: removing highly correlating variables

After this, the mutual correlation between the remaining predictors is examined, to prevent potential problems associated with multicollinearity. First a visual presentation of the correlationmatrix is produced for visual inspection. After that a set of highly correlated variables (> 0.85) is selected by the findCorrelation() function of the caret-package. This function removes a minimum set of predictors to ensure that all pairwise correlations remain below a given threshold (in this case 0.85). This procedure removes another 9 predictors in both training- and testset:

```
# looking at the correlations between the numerical predictors
corMatrix <- cor(trainSet[, -54])
corrplot(corMatrix, order = "FPC", method = "color", type = "lower", tl.cex = 0.6, tl.col = rgb(0, 0, 0
```




```
# removing higly correlated predictors (cor > 0.85)
hcVar <- findCorrelation(corMatrix, cutoff = 0.85)
hcVar
```

```
## [1] 11  2 10  9  3 46 32 34 19
```

```
trainSet <- trainSet[ , -hcVar]
testSet <- testSet[, -hcVar]
dim(trainSet)
```

```
## [1] 13737    45
```

```
# dim(testSet)
```

This final dataprep-step removed another 9 variables, leaving a dataframe of 45 variables. After this data-preparation, a new training- and testset was computed with the preProcess() function of the caret-package (preProcess(trainSet, method = c("BoxCox", "center", "scale", "pca"))). Because the accuracy of the results with this transformed dataset was less than the corresponding accuracy with the raw data (with one exception for the knn-algorithm), the raw data is used in comparing the different algorithms.

Fitting prediction models

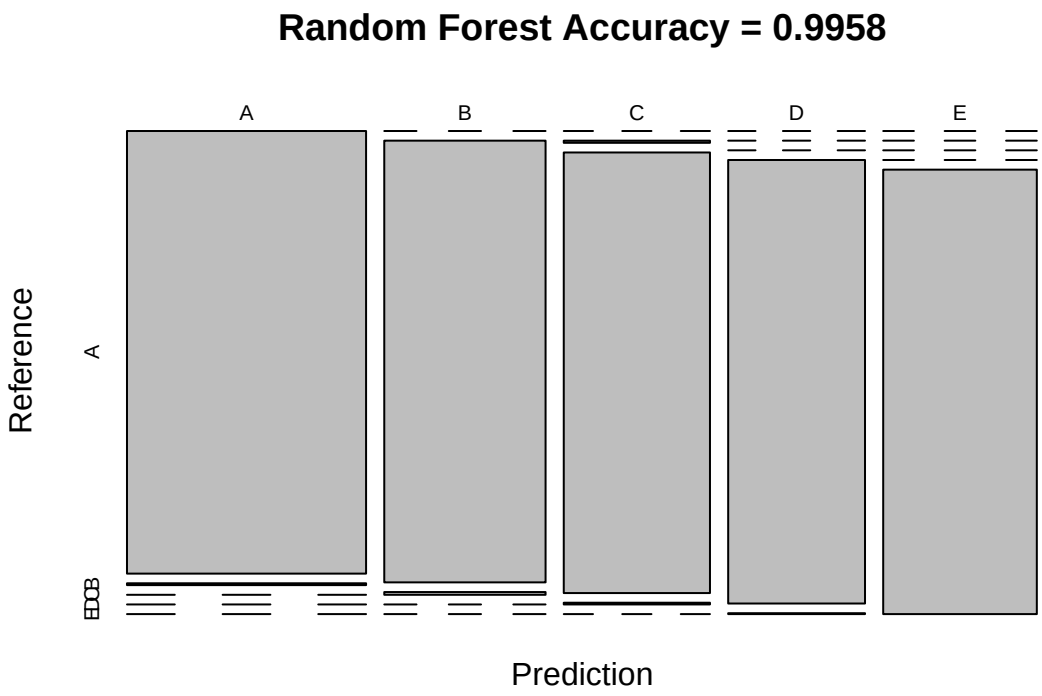
In this part five different models will be fitted to the trainingdata (and evaluated against the testdata).

Random Forest (RF)

```
# Random Forest
set.seed(131055)
trControl <- trainControl(method = "cv", number = 5, verboseIter = FALSE)
mod_rf <- train(classe ~ ., data = trainSet, method = "rf", trControl = trControl)
# mod_rf$finalModel; suppress: to much output
predict_rf <- predict(mod_rf, newdata = testSet)
confMat_rf <- confusionMatrix(predict_rf, testSet$classe)
confMat_rf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    7    0    0    0
##          B    0 1127    7    0    0
##          C    0    5 1019    4    0
##          D    0    0    0  960    2
##          E    0    0    0    0 1080
##
## Overall Statistics
##
##                Accuracy : 0.9958
##                  95% CI : (0.9937, 0.9972)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9946
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9895   0.9932   0.9959   0.9982
## Specificity            0.9983   0.9985   0.9981   0.9996   1.0000
## Pos Pred Value         0.9958   0.9938   0.9912   0.9979   1.0000
## Neg Pred Value         1.0000   0.9975   0.9986   0.9992   0.9996
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2845   0.1915   0.1732   0.1631   0.1835
## Detection Prevalence   0.2856   0.1927   0.1747   0.1635   0.1835
## Balanced Accuracy      0.9992   0.9940   0.9957   0.9977   0.9991
```

```
plot(confMat_rf$table, main = paste("Random Forest Accuracy =", round(confMat_rf$overall["Accuracy"], 4
```



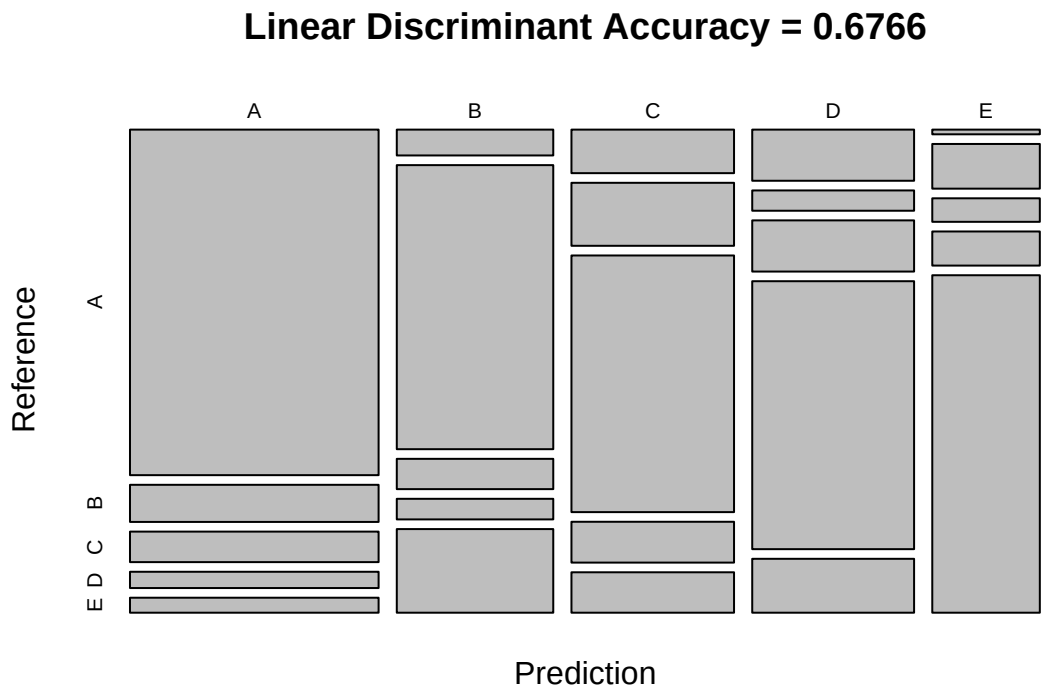
Linear Discriminant Analysis (LDA)

```
set.seed(131055)
trControl <- trainControl(method = "cv", number = 5, verboseIter = FALSE)
mod_lda <- train(classe ~ ., data = trainSet, method = "lda", trControl = trControl)
# mod_lda$finalModel; suppress: to much output
predict_lda <- predict(mod_lda, newdata = testSet)
confMat_lda <- confusionMatrix(predict_lda, testSet$classe)
confMat_lda
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1359  146  120   64   59
##          B   64  703   75   51  207
##          C  112  162  660  105  104
##          D  131   52  131  686  138
##          E    8   76   40   58  574
##
## Overall Statistics
##
##                Accuracy : 0.6766
##                  95% CI : (0.6645, 0.6886)
##     No Information Rate : 0.2845
```

```
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5908
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8118   0.6172   0.6433   0.7116  0.53050
## Specificity            0.9076   0.9164   0.9006   0.9081  0.96211
## Pos Pred Value         0.7775   0.6391   0.5774   0.6028  0.75926
## Neg Pred Value         0.9239   0.9089   0.9228   0.9414  0.90096
## Prevalence             0.2845   0.1935   0.1743   0.1638  0.18386
## Detection Rate         0.2309   0.1195   0.1121   0.1166  0.09754
## Detection Prevalence   0.2970   0.1869   0.1942   0.1934  0.12846
## Balanced Accuracy      0.8597   0.7668   0.7719   0.8099  0.74630
```

```
plot(confMat_lda$table, main = paste("Linear Discriminant Accuracy =", round(confMat_lda$overall["Accura
```



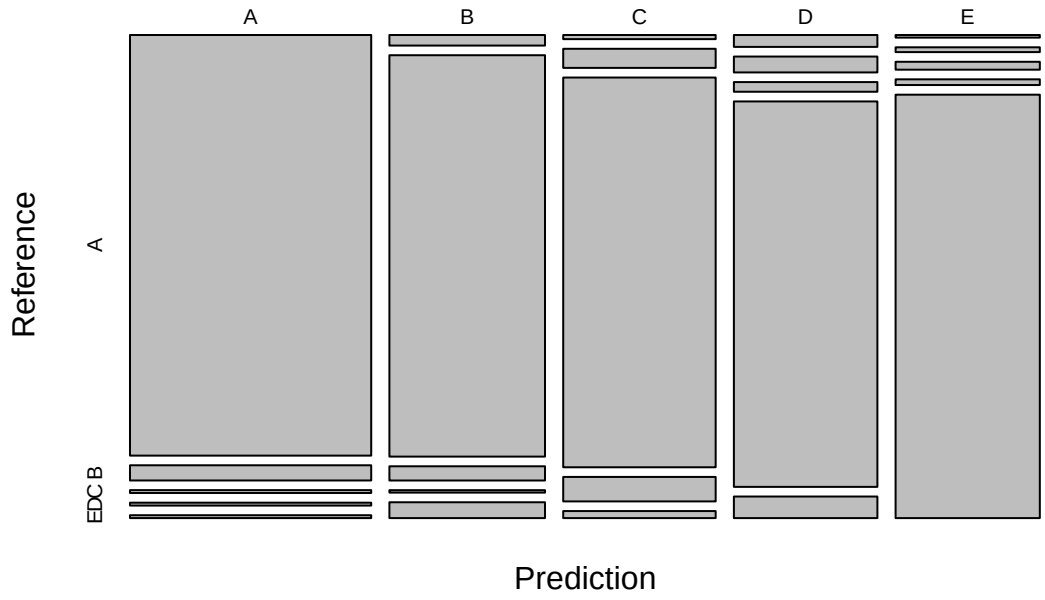
K Nearest Neighbor (KNN)

```
set.seed(131055)
trControl <- trainControl(method = "cv", number = 5, verboseIter = FALSE)
mod_knn <- train(classe ~ ., data = trainSet, method = "knn", trControl = trControl)
# mod_knn$finalModel; suppress: to much output
```

```r
predict_knn <- predict(mod_knn, newdata = testSet)
confMat_knn <- confusionMatrix(predict_knn, testSet$classe)
confMat_knn
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1605   58   11   11   11
##          B   26  988   35    6   39
##          C   10   46  940   59   17
##          D   27   36   22  875   49
##          E    6   11   18   13  966
##
## Overall Statistics
##
##                Accuracy : 0.9132
##                  95% CI : (0.9057, 0.9202)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8902
##  Mcnemar's Test P-Value : 2.29e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9588   0.8674   0.9162   0.9077   0.8928
## Specificity            0.9784   0.9777   0.9728   0.9728   0.9900
## Pos Pred Value         0.9463   0.9031   0.8769   0.8672   0.9527
## Neg Pred Value         0.9835   0.9685   0.9821   0.9817   0.9762
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2727   0.1679   0.1597   0.1487   0.1641
## Detection Prevalence   0.2882   0.1859   0.1822   0.1715   0.1723
## Balanced Accuracy      0.9686   0.9225   0.9445   0.9402   0.9414
```

```r
plot(confMat_knn$table, main = paste("K Nearest Neighbor Accuracy =", round(confMat_knn$overall["Accura
```



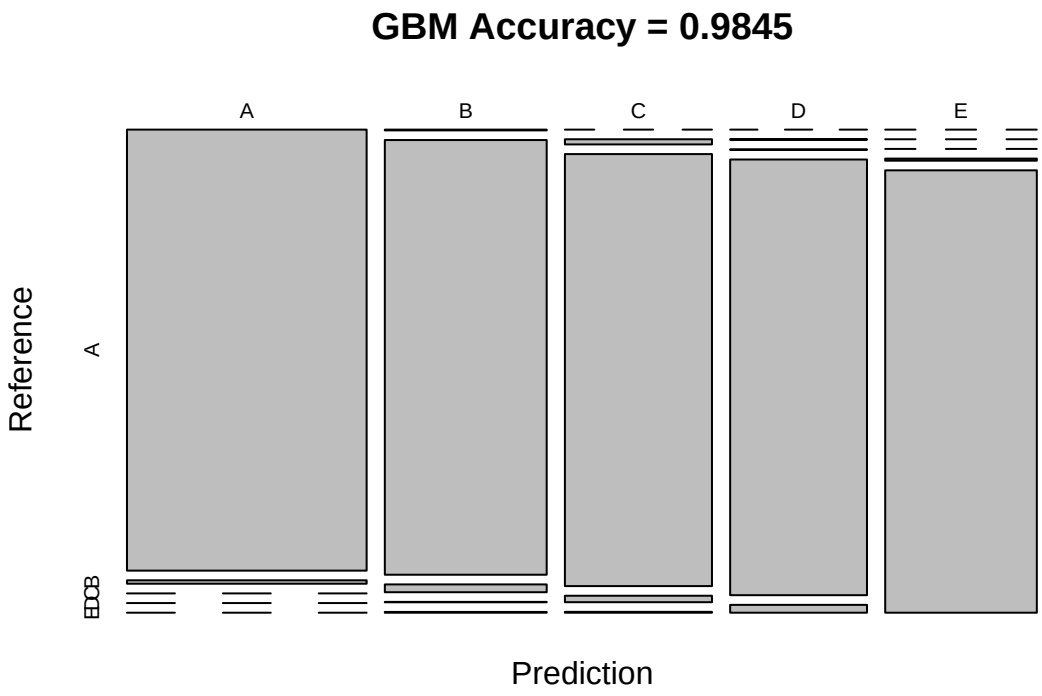
Gradient Boosting Model(GBM)

```
set.seed(131055)
trControl <- trainControl(method = "cv", number = 5, verboseIter = FALSE)
mod_gbm <- train(classe ~ ., data = trainSet, method = "gbm", trControl = trControl, verbose = FALSE)
# mod_gbm$finalModel; suppress: to much output
predict_gbm <- predict(mod_gbm, newdata = testSet)
confMat_gbm <- confusionMatrix(predict_gbm, testSet$classe)
confMat_gbm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1672   13    0    0    0
##          B    2 1113   20    1    2
##          C    0   12 1005   15    2
##          D    0    1    1  943   17
##          E    0    0    0    5 1061
##
## Overall Statistics
##
##                Accuracy : 0.9845
##                  95% CI : (0.981, 0.9875)
##     No Information Rate : 0.2845
```

```
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9804
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9988   0.9772   0.9795   0.9782   0.9806
## Specificity            0.9969   0.9947   0.9940   0.9961   0.9990
## Pos Pred Value         0.9923   0.9780   0.9720   0.9802   0.9953
## Neg Pred Value         0.9995   0.9945   0.9957   0.9957   0.9956
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2841   0.1891   0.1708   0.1602   0.1803
## Detection Prevalence   0.2863   0.1934   0.1757   0.1635   0.1811
## Balanced Accuracy      0.9979   0.9860   0.9868   0.9872   0.9898
```

```r
plot(confMat_gbm$table, main = paste("GBM Accuracy =", round(confMat_gbm$overall["Accuracy"], 4)))
```



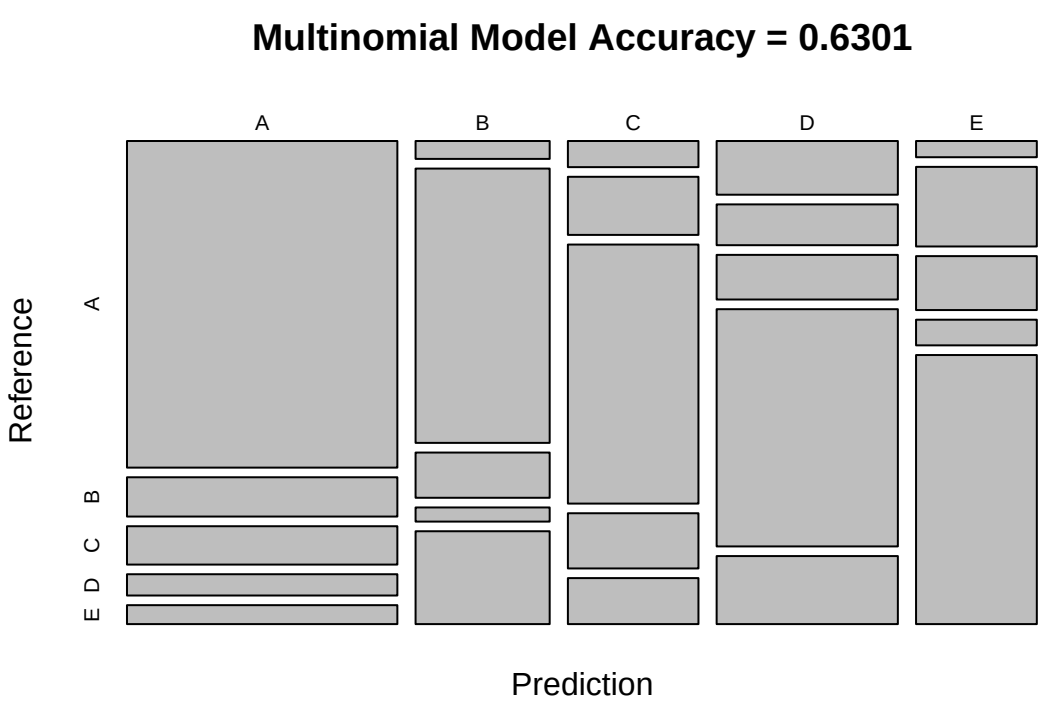
Multinominal Model (MULTINOM)

```r
set.seed(131055)
trControl <- trainControl(method = "cv", number = 5, verboseIter = FALSE)
mod_multinom <- train(classe ~ ., data = trainSet, method = "multinom", trControl = trControl, trace = F
# mod_multinom$finalModel; suppress: to much output
```

```r
predict_multinom <- predict(mod_multinom, newdata = testSet)
confMat_multinom <- confusionMatrix(predict_multinom, testSet$classe)
confMat_multinom
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1397  168  164   91   81
##          B   38  582   96   30  197
##          C   54  120  535  114   95
##          D  154  117  128  680  195
##          E   31  152  103   49  514
##
## Overall Statistics
##
##                Accuracy : 0.6301
##                  95% CI : (0.6176, 0.6424)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5305
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8345   0.5110  0.52144   0.7054  0.47505
## Specificity            0.8803   0.9239  0.92118   0.8793  0.93025
## Pos Pred Value         0.7349   0.6172  0.58279   0.5338  0.60542
## Neg Pred Value         0.9305   0.8873  0.90115   0.9384  0.88721
## Prevalence             0.2845   0.1935  0.17434   0.1638  0.18386
## Detection Rate         0.2374   0.0989  0.09091   0.1155  0.08734
## Detection Prevalence   0.3230   0.1602  0.15599   0.2165  0.14427
## Balanced Accuracy      0.8574   0.7175  0.72131   0.7923  0.70265
```

```r
plot(confMat_multinom$table, main = paste("Multinomial Model Accuracy =", round(confMat_multinom$overall
```



Final Prediction on the Test-set

All models were trained with crossvalidation (5 folds). Sorted on performance to predict the test data, the ranking of the different models is as follows:

1: 0.9958 (Random Forest)
2: 0.9845 (GBM)
3: 0.9132 (KNN)
4: 0.6766 (LDA)
*5: 0.6301 (Multinom)

On accuracy, Random Forest is a clear winner. And it is clear from the corresponding plot that all categories of classe are more or less equally predicted.

So for the final prediction on the extra test-set of 20 observations, the final Random Forest model is choosen.

```
predict_final <- predict(mod_rf, newdata = testing)
predict_final
```

```
##  [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
```