Classe Predictor

Nazar Mammedov

2025-01-12

Loading data

Create training and test partitions. We create internal train and test partitions from the data provided in "pml-training.csv" because we want to be able to test our model.

```
library(caret)
```

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

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

```
library(ggplot2)
library(e1071)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
data = read.csv("pml-training.csv")
external_test = read.csv("pml-testing.csv")
```

Cleaning data

Create a new dataset with proper columns both for training and external test. We drop columns with a lot of NAs because they are not going to be useful anyway, and also drop some variables such as timestamps.

```r
# Drop columns with any NA values
data_no_na <- data %>% select_if(~ !any(is.na(.)))
test_no_na <- external_test %>% select_if(~ !any(is.na(.)))

#Select only numeric columns
data_numeric <- data_no_na %>% select_if(is.numeric) %>% bind_cols(data_no_na %>% select(classe))
test_numeric <- test_no_na %>% select_if(is.numeric)

# drop first 4 columns X, timestamps, and num_window
# because I don't think they are good predictors anyway
new_data <- data_numeric[, -c(1:4)]
new_data$classe <- as.factor(new_data$classe)
new_test_data <- test_numeric[, -c(1:4)]
new_test_data <- new_test_data %>% select(-problem_id)
```

Basic modelling

Now given this data set. We train the model with SVM linear. It gives .7856 accuracy level which is not good.

```r
# Split the data into training and testing sets (70/30 split)
set.seed(123)
inTrain <- createDataPartition(new_data$classe, p = 0.7, list = FALSE)
train_data <- new_data[inTrain, ]
test_data <- new_data[-inTrain, ]

model_all <- svm(classe ~ ., data = train_data, kernel = "linear")

predictions_all <- predict(model_all, test_data)
cm <- confusionMatrix(predictions_all, test_data$classe)
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1557  149  109   63   60
##          B   28  837   83   42  155
##          C   34   65  796  112   85
##          D   43   18   23  703   52
##          E   12   70   15   44  730
##
## Overall Statistics
##
##                Accuracy : 0.7856
##                  95% CI : (0.7748, 0.796)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7271
##
##  Mcnemar's Test P-Value : < 2.2e-16
```

```
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9301   0.7349   0.7758   0.7293   0.6747
## Specificity            0.9095   0.9351   0.9391   0.9724   0.9706
## Pos Pred Value         0.8034   0.7310   0.7289   0.8379   0.8381
## Neg Pred Value         0.9704   0.9363   0.9520   0.9483   0.9298
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2646   0.1422   0.1353   0.1195   0.1240
## Detection Prevalence   0.3293   0.1946   0.1856   0.1426   0.1480
## Balanced Accuracy      0.9198   0.8350   0.8575   0.8508   0.8227
```

Random forests approach

We try Random forests approach. And we don't have to train it on a full training partition. We can use a smaller subset. Otherwise we have to wait for zillion hours for the training to complete because we don't have a supercomputer at hand. It still gives higher than 90% results.

```r
# Sample 2000 observations randomly and create a smaller subset
sample_indices <- sample(seq_len(nrow(train_data)), size = 4000)
small_train <- train_data[sample_indices, ]

control = trainControl(method="cv", number=5)
model_rf <- train(classe~., data=small_train, method='rf', trControl = control)

predictions_rf <- predict(model_rf, test_data)
cmrf <- confusionMatrix(predictions_rf, test_data$classe)
cmrf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1656   27    0    0    1
##          B   12 1087   22    2    1
##          C    5   24  993   26    8
##          D    0    1   11  929    7
##          E    1    0    0    7 1065
##
## Overall Statistics
##
##                Accuracy : 0.9737
##                  95% CI : (0.9692, 0.9776)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9667
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
```

```
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9892   0.9543   0.9678   0.9637   0.9843
## Specificity            0.9934   0.9922   0.9870   0.9961   0.9983
## Pos Pred Value         0.9834   0.9671   0.9403   0.9800   0.9925
## Neg Pred Value         0.9957   0.9891   0.9932   0.9929   0.9965
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2814   0.1847   0.1687   0.1579   0.1810
## Detection Prevalence   0.2862   0.1910   0.1794   0.1611   0.1823
## Balanced Accuracy      0.9913   0.9733   0.9774   0.9799   0.9913
```

Random Forest gives better Accuracy.

Out of sample error rate

Out of sample error rate is (1-Accuracy rate), but can be calculated as below.

```
conf_matrix <- cmrf$table
misclassifications <- sum(conf_matrix) - sum(diag(conf_matrix))
total_observations <- sum(conf_matrix)
misclassification_rate <- misclassifications / total_observations
print(misclassification_rate)
```

```
## [1] 0.02633815
```

Predicting 20 cases

Now we predict "classe" in the external test data which was provided in "pml-testing.csv". As predicted this gives about 94% of accuracy, which was also the expected Quiz result.

```
predictions_test <- predict(model_rf, new_test_data)
print(predictions_test)
```

```
##  [1] B A A A A E D D A A B C B A E E A B B B
## Levels: A B C D E
```