pml

2025-04-08

Setup

packages

```
library(caret)
```

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

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

Reading in data and exploratory data analysis, and cleaning and splitting

```
training = read.csv("pml-training.csv") # read in training set
testing = read.csv("pml-testing.csv") # read in testing set
names(training) # display variable names to remove some insignificant ones for lighter data
```

```
##   [1] "X"                       "user_name"
##   [3] "raw_timestamp_part_1"    "raw_timestamp_part_2"
##   [5] "cvtd_timestamp"          "new_window"
##   [7] "num_window"              "roll_belt"
##   [9] "pitch_belt"              "yaw_belt"
##  [11] "total_accel_belt"        "kurtosis_roll_belt"
##  [13] "kurtosis_picth_belt"     "kurtosis_yaw_belt"
##  [15] "skewness_roll_belt"      "skewness_roll_belt.1"
##  [17] "skewness_yaw_belt"       "max_roll_belt"
##  [19] "max_picth_belt"          "max_yaw_belt"
##  [21] "min_roll_belt"           "min_pitch_belt"
##  [23] "min_yaw_belt"            "amplitude_roll_belt"
##  [25] "amplitude_pitch_belt"    "amplitude_yaw_belt"
##  [27] "var_total_accel_belt"    "avg_roll_belt"
##  [29] "stddev_roll_belt"        "var_roll_belt"
##  [31] "avg_pitch_belt"          "stddev_pitch_belt"
##  [33] "var_pitch_belt"          "avg_yaw_belt"
##  [35] "stddev_yaw_belt"         "var_yaw_belt"
##  [37] "gyros_belt_x"            "gyros_belt_y"
##  [39] "gyros_belt_z"            "accel_belt_x"
##  [41] "accel_belt_y"            "accel_belt_z"
##  [43] "magnet_belt_x"           "magnet_belt_y"
##  [45] "magnet_belt_z"           "roll_arm"
##  [47] "pitch_arm"               "yaw_arm"
##  [49] "total_accel_arm"         "var_accel_arm"
##  [51] "avg_roll_arm"            "stddev_roll_arm"
##  [53] "var_roll_arm"            "avg_pitch_arm"
##  [55] "stddev_pitch_arm"        "var_pitch_arm"
##  [57] "avg_yaw_arm"             "stddev_yaw_arm"
##  [59] "var_yaw_arm"             "gyros_arm_x"
##  [61] "gyros_arm_y"             "gyros_arm_z"
##  [63] "accel_arm_x"             "accel_arm_y"
##  [65] "accel_arm_z"             "magnet_arm_x"
##  [67] "magnet_arm_y"            "magnet_arm_z"
##  [69] "kurtosis_roll_arm"       "kurtosis_picth_arm"
##  [71] "kurtosis_yaw_arm"        "skewness_roll_arm"
##  [73] "skewness_pitch_arm"      "skewness_yaw_arm"
##  [75] "max_roll_arm"            "max_picth_arm"
##  [77] "max_yaw_arm"             "min_roll_arm"
##  [79] "min_pitch_arm"           "min_yaw_arm"
##  [81] "amplitude_roll_arm"      "amplitude_pitch_arm"
##  [83] "amplitude_yaw_arm"       "roll_dumbbell"
##  [85] "pitch_dumbbell"          "yaw_dumbbell"
##  [87] "kurtosis_roll_dumbbell"  "kurtosis_picth_dumbbell"
##  [89] "kurtosis_yaw_dumbbell"   "skewness_roll_dumbbell"
##  [91] "skewness_pitch_dumbbell" "skewness_yaw_dumbbell"
##  [93] "max_roll_dumbbell"       "max_picth_dumbbell"
##  [95] "max_yaw_dumbbell"        "min_roll_dumbbell"
##  [97] "min_pitch_dumbbell"      "min_yaw_dumbbell"
##  [99] "amplitude_roll_dumbbell" "amplitude_pitch_dumbbell"
## [101] "amplitude_yaw_dumbbell"  "total_accel_dumbbell"
## [103] "var_accel_dumbbell"      "avg_roll_dumbbell"
## [105] "stddev_roll_dumbbell"    "var_roll_dumbbell"
## [107] "avg_pitch_dumbbell"      "stddev_pitch_dumbbell"
## [109] "var_pitch_dumbbell"      "avg_yaw_dumbbell"
## [111] "stddev_yaw_dumbbell"     "var_yaw_dumbbell"
## [113] "gyros_dumbbell_x"        "gyros_dumbbell_y"
## [115] "gyros_dumbbell_z"        "accel_dumbbell_x"
## [117] "accel_dumbbell_y"        "accel_dumbbell_z"
## [119] "magnet_dumbbell_x"       "magnet_dumbbell_y"
## [121] "magnet_dumbbell_z"       "roll_forearm"
## [123] "pitch_forearm"           "yaw_forearm"
## [125] "kurtosis_roll_forearm"   "kurtosis_picth_forearm"
## [127] "kurtosis_yaw_forearm"    "skewness_roll_forearm"
## [129] "skewness_pitch_forearm"  "skewness_yaw_forearm"
## [131] "max_roll_forearm"        "max_picth_forearm"
## [133] "max_yaw_forearm"         "min_roll_forearm"
## [135] "min_pitch_forearm"       "min_yaw_forearm"
## [137] "amplitude_roll_forearm"  "amplitude_pitch_forearm"
```

```
## [139] "amplitude_yaw_forearm"    "total_accel_forearm"
## [141] "var_accel_forearm"        "avg_roll_forearm"
## [143] "stddev_roll_forearm"      "var_roll_forearm"
## [145] "avg_pitch_forearm"        "stddev_pitch_forearm"
## [147] "var_pitch_forearm"        "avg_yaw_forearm"
## [149] "stddev_yaw_forearm"       "var_yaw_forearm"
## [151] "gyros_forearm_x"          "gyros_forearm_y"
## [153] "gyros_forearm_z"          "accel_forearm_x"
## [155] "accel_forearm_y"          "accel_forearm_z"
## [157] "magnet_forearm_x"         "magnet_forearm_y"
## [159] "magnet_forearm_z"         "classe"
```

```
table(training$classe) # check that classes are not terribly skewed
```

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

```
dim(testing);dim(training) # sizes of data
```

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

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

```
sum(!complete.cases(training));sum(!complete.cases(testing)) # checking for missing data
```

```
## [1] 19216
```

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

```
removeable = c(grep("^(skewness|avg|var|stddev|amplitude|kurtosis|min|max)",names(training)),1,2,3,4,6,7) # the grep f
unction picks up variables that can be represented by other variables in the data (see codebook in question)
training = subset(training,select = -removeable)
testing = subset(testing,select = -removeable)
sum(!complete.cases(training));sum(!complete.cases(testing)) # rechecking for missing data after removing insignifican
t variables
```

```
## [1] 0
```

```
## [1] 0
```

```
set.seed(143) # for randomness in createDataPartition()
trainInd = createDataPartition(training$classe,p = 0.7,list = FALSE)# train/test split
train2 = training[trainInd,]
test2 = training[-trainInd,]
```

Evaluating models

We begin with reviewing a simple rpart decision tree on the data, as it is inexpensive to train, making it highly scalable

```
# rpart decision tree
set.seed(197) # for randomness in train()
modelrpart = train(classe~.,method="rpart",data=train2,trControl = trainControl(method = "cv"))
table(predict(modelrpart,test2[-54]))
```

```
##
##    A    B    C    D    E
## 2734  798 1817    0  536
```

rpart was unable to classify into D, making it unsuitable. An ensemble of randomised decision trees (ie random forest) may work better, as random variables are included. Before trying that, training data will, again, be split by train/split to prevent overfitting to in sample errors. As this is simply part of feature selection in cross validation, a small ensemble of 5 trees with 3 folds will be used.

```
#random forest
set.seed(1819) # for randomness in createDataPartition() and train()
trainInd = createDataPartition(train2$classe,p = 0.7,list=FALSE) # further train/test split
train3 = train2[trainInd,]
test3 = train2[-trainInd,]
startTime = Sys.time()
modelrf = train(classe~.,method="rf",data=train3,trControl = trainControl(method="cv",number = 3),ntree=5)
Sys.time()-startTime # prints time taken for fitting
confusionMatrix(as.factor(test3$classe),predict(modelrf,test3[,-54]))$table
confusionMatrix(as.factor(test3$classe),predict(modelrf,test3[,-54]))$overall
```

```
## Time difference of 2.989816 secs
##           Reference
## Prediction    A    B    C    D    E
##          A 1165    5    1    0    0
##          B   14  771    8    3    1
##          C    1   18  698    1    0
##          D    1    0    8  664    2
##          E    0    0    2    5  750
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9834871      0.9791090      0.9791124      0.9871549      0.2865469
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

This shows how random forest works a lot better, even with only 3 fold cross validation and ensembling 5 trees producing a very high accuracy of 0.983. Preprocesisng looks unnecessary. The short time frame for model fitting also means we can afford to increase ntree to 50 and number of folds to 10 for less bias, training on train2 and validating with test2.

```
set.seed(2025)
startTime = Sys.time()
modelfinal = train(classe~.,method="rf",data=train2,trControl = trainControl(method="cv",number = 10),ntree=50)
Sys.time()-startTime # prints time taken for fitting
confusionMatrix(as.factor(test2$classe),predict(modelfinal,test2[,-54]))$table
confusionMatrix(as.factor(test2$classe),predict(modelfinal,test2[,-54]))$overall
```

```
## Time difference of 2.281408 mins
##           Reference
## Prediction    A    B    C    D    E
##          A 1673    0    1    0    0
##          B    8 1126    5    0    0
##          C    0    7 1016    3    0
##          D    0    0    4  958    2
##          E    0    0    0    0 1082
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9947324      0.9933363      0.9925313      0.9964182      0.2854715
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

Accuracy is even higher at 99.5% and only takes about 2.5 minutes to fit, still making it possible to scale. As model selection for modelfinal is based on metrics from a subsample, we can expect that the model is not overfitted onto the data, and will be representative of the population sample. Hence, an error rate of around 0.5% can be expected out of sample.

The model's prediction of the 20 test cases in testing

```
predict(modelfinal,testing)
```

```
##  [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
```