Predictive Model

Jasmine

2025-02-19

Weight Lifting Exercises Dataset

```
library(dplyr)
library(caret)
library(corrplot)
library(xgboost)
library(Matrix)
library(e1071)
library(kernlab)
```

Data Preprocessing and feature selection

We select necessary features for predictive modeling and remove columns with missing values, non-predictive columns, columns with near zero variance, and highly correlated features. We also standardize the data. All selected features are related to the accelerometers and gyroscopes of the devices worn by the participants.

- **Gyroscope (`gyros_*`)**: Measures angular velocity (rotation).
- **Accelerometer (`accel_*`)**: Measures linear acceleration.
- **Magnetometer (`magnet_*`)**: Measures orientation relative to Earth's magnetic field.
- **Euler Angles (`roll_*`, `pitch_*`, `yaw_*`)**: Describes rotational movements.
- **Total Acceleration (`total_accel_*`)**: Combined acceleration magnitude.

```
# read the data, replace missing values with NA, remove columns with missing values and drop non-predic

test <- read.csv("pml-testing.csv", na.strings = c("NA","",'#DIV/0!'))
train <- read.csv("pml-training.csv", na.strings = c("NA","",'#DIV/0!'))
test <- test[,colSums(is.na(test)) == 0]
test <- na.omit(test)
train <- train[, colSums(is.na(train)) == 0]
train <- na.omit(train)
test <- test[, !colnames(test) %in% c("X", "problem_id", "raw_timestamp_part_1", "raw_timestamp_part_2"
train <- train[, !colnames(train) %in% c("X", "problem_id", "raw_timestamp_part_1", "raw_timestamp_part_

# check the dimensions and names of the datasets
# dim(train)
# dim(test)
# names(train)
# names(test)

# data encode (factorize) the classes
train$classe <- factor(train$classe)
```

```r
# data exploration
table(train$classe)
```

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

```r
# summary(train)

# feature selection
# remove columns with near zero variance and highly correlated features
# nzv_features <- nearZeroVar(train[, sapply(train, is.numeric)]) # no near zero variance features
cor_matrix <- cor(train[, sapply(train, is.numeric)])
correlated_features <- findCorrelation(cor_matrix, cutoff = 0.8)
train <- train[, -correlated_features]
test <- test[, -correlated_features]

# standardize the data
preProc_tr<- preProcess(train[,-41], method = c("center", "scale"))
train_scaled <- predict(preProc_tr, train[,-41])
train_scaled$classe <- train$classe
preProc_te <- preProcess(test, method = c("center", "scale"))
test_scaled <- predict(preProc_te, test)
```

Slice the train data into training and testing datasets

The scaled train dataset is split into 70% training and 30% testing datasets.

```r
set.seed(123)
trainIndex <- createDataPartition(train_scaled$classe, p = 0.7, list = FALSE)
train_data <- train_scaled[trainIndex,]
test_data <- train_scaled[-trainIndex,]
```

Random Forest Model

We train a random forest model with 5-fold cross validation and 100 trees (to save time). The model is evaluated using the confusion matrix.

```r
# train the model and tune the hyperparameters using built-in cross-validation
tune_grid <- expand.grid(
  mtry = c(2, 5, 10,15,20)) # number of variables randomly sampled as candidates at each split
control <- trainControl(method = "cv", number = 5)
# Fit the model
model_rf <- train(classe ~ ., data = train_data, method = "rf", trControl = control, tuneGrid = tune_gr
model_rf
```

```
## Random Forest
##
## 13737 samples
##    40 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
```

```
## Summary of sample sizes: 10990, 10990, 10989, 10988, 10991
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9904640  0.9879359
##    5    0.9920654  0.9899629
##   10    0.9925024  0.9905155
##   15    0.9921382  0.9900550
##   20    0.9905369  0.9880285
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 10.
```

```
# model evaluation for the test dataset
confusionMatrix(predict(model_rf, test_data), test_data$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1673    4    0    0    0
##          B    0 1129    4    0    1
##          C    0    6 1022    7    4
##          D    0    0    0  957    4
##          E    1    0    0    0 1073
##
## Overall Statistics
##
##                Accuracy : 0.9947
##                  95% CI : (0.9925, 0.9964)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9933
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9994   0.9912   0.9961   0.9927   0.9917
## Specificity            0.9991   0.9989   0.9965   0.9992   0.9998
## Pos Pred Value         0.9976   0.9956   0.9836   0.9958   0.9991
## Neg Pred Value         0.9998   0.9979   0.9992   0.9986   0.9981
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2843   0.1918   0.1737   0.1626   0.1823
## Detection Prevalence   0.2850   0.1927   0.1766   0.1633   0.1825
## Balanced Accuracy      0.9992   0.9951   0.9963   0.9960   0.9957
```

```
# Predict the classe for the 20 sample size test_scaled dataset
predict(model_rf, test_scaled)
```

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

XGBoost Model

The XGBoost model is configured for multi-class classification with `"multi:softmax"` as the objective function, a maximum tree depth of 10, a learning rate (`eta`) of 0.3, and early stopping after 5 rounds if `mlogloss` does not improve. A 5-fold cross-validation (`xgb.cv`) determines the optimal number of boosting rounds (`best_iter`), enhancing model efficiency while preventing overfitting. The final model is trained using `xgb.train()` with the best iteration count. Predictions on the test dataset are evaluated with `confusionMatrix()`.

```
# prepare Matrix for xgboost
# train data
train_data1 <- data.matrix(train_data[,-41])
train_data1 <- Matrix(train_data1, sparse = TRUE) # convert to sparse matrix
train_y <- as.numeric(train_data$classe)-1 # convert classe to numeric-1, as xgboost requires labels to
traindata <- list(data = train_data1, label = train_y)
dtrain <- xgb.DMatrix(data = traindata$data, label = traindata$label)
# test data
test_data1 <- data.matrix(test_data[,-41])
test_data1 <- Matrix(test_data1, sparse = TRUE) # convert to sparse matrix
test_y <- as.numeric(test_data$classe)-1 # convert classe to numeric-1, as xgboost requires labels to s
testdata <- list(data = test_data1, label = test_y)
dtest <- xgb.DMatrix(data = testdata$data, label = testdata$label)
# test scaled data
test_scaled1 <- data.matrix(test_scaled)
test_scaled1 <- Matrix(test_scaled1, sparse = TRUE)
dtest_scaled <- xgb.DMatrix(data = test_scaled1)


# set the parameters and build the model

param <- list(
  objective = "multi:softmax",
  booster = "gbtree",
  max_depth = 10,
  eta = 0.3,
  nrounds = 500,  # Number of boosting iterations
  num_class = 5,  # Number of classes for multi-class classification
  verbose = T, # Report performance
  early_stopping_rounds = 5,  # Stop early if no improvement in 10 rounds
  eval_metric = "mlogloss",
  nfolds = 5,
  maximize = FALSE
)

cv_results <- xgb.cv(
  params = param,
  data = dtrain,
  nfold = 5,
  nrounds = 500,
  verbose = F,
  early_stopping_rounds = 5,
  maximize = FALSE
)

# Select the best iteration
```

```r
best_iter <- cv_results$best_iteration
# Train the model
xgb_model <- xgb.train(
  params = param,
  data = dtrain,
  nrounds = best_iter
)
```

```
## [23:03:56] WARNING: src/learner.cc:767:
## Parameters: { "early_stopping_rounds", "maximize", "nfolds", "nrounds", "verbose" } are not used.
```

```r
# model evaluation for the test dataset
pred <- predict(xgb_model, dtest)
xgb.cf <- confusionMatrix(as.factor(pred), as.factor(test_y))
xgb.cf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1    2    3    4
##          0 1672    0    0    0    0
##          1    0 1132    4    0    0
##          2    0    6 1021    1    4
##          3    0    0    1  960    5
##          4    2    1    0    3 1073
##
## Overall Statistics
##
##                Accuracy : 0.9954
##                  95% CI : (0.9933, 0.997)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9942
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity            0.9988   0.9939   0.9951   0.9959   0.9917
## Specificity            1.0000   0.9992   0.9977   0.9988   0.9988
## Pos Pred Value         1.0000   0.9965   0.9893   0.9938   0.9944
## Neg Pred Value         0.9995   0.9985   0.9990   0.9992   0.9981
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2841   0.1924   0.1735   0.1631   0.1823
## Detection Prevalence   0.2841   0.1930   0.1754   0.1641   0.1833
## Balanced Accuracy      0.9994   0.9965   0.9964   0.9973   0.9952
```

```r
# predict on the test dataset
pred <-predict(xgb_model, dtest_scaled)
# transform the predictions to the original classe labels
class_labels <- c("A", "B", "C", "D", "E")
class_labels[pred + 1]
```

```
##  [1] "E" "A" "B" "E" "A" "E" "D" "D" "A" "E" "A" "C" "D" "A" "E" "D" "E" "B" "E"
## [20] "D"
```

SVM Model

The Support Vector Machine (SVM) model is trained using the `svm()` function with a radial basis function (RBF) kernel. The model is evaluated using 5 fold cross-validation.

```
# Define 5-fold cross-validation settings
cv_control <- trainControl(method = "cv", number = 5)

# Train SVM model with cross-validation
svm_cv_model <- train(
  classe ~ .,
  data = train_data,
  method = "svmRadial",
  trControl = cv_control,
  tuneGrid = expand.grid(sigma = 0.1, C = 0.5)  # Equivalent to gamma and cost
)

svm_cv_model
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 13737 samples
##    40 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10989, 10990, 10989, 10989, 10991
## Resampling results:
##
##   Accuracy   Kappa
##   0.9485341  0.934828
##
## Tuning parameter 'sigma' was held constant at a value of 0.1
## Tuning
##  parameter 'C' was held constant at a value of 0.5
```

```
# Predict on the test dataset
predict(svm_cv_model, test_scaled)
```

```
##  [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
```