Practical Machine Learning – Course Project

Phuoc NguyenNguyen

23 October, 2025

1 1. Synopsis

The goal of this project is to use data from accelerometers on the belt, forearm, arm, and dumbell to predict the manner in which participants performed a barbell lift exercise. The prediction target is the `classe` variable in the training dataset. This report details the steps taken to build a predictive model, including data preprocessing, model training, and evaluation.

2 2. Loading and Preprocessing Data

2.1 2.1. Setting up the Environment

First, we load the required R libraries for this analysis. We will use `caret` for model training and evaluation, `dplyr` for data manipulation, `randomForest` for our prediction model, and `rpart` for decision tree visualization if needed.

```
library(caret)
library(dplyr)
library(randomForest)
library(rpart)
```

2.2 2.2. Loading the Data

Next, we load the training and testing datasets from the provided CSV files, which are available in the project directory.

```
# Load the training data
pml.training <- read.csv("pml-training.csv", na.strings=c("NA","#DIV/0!", ""))

# Load the testing data
pml.testing <- read.csv("pml-testing.csv", na.strings=c("NA","#DIV/0!", ""))
```

We can take a look at the dimensions of the loaded datasets.

```
dim(pml.training)
```

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

```
dim(pml.testing)
```

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

2.3 2.3. Data Cleaning

Before building the model, we need to clean the data. This involves removing irrelevant columns and columns with a large number of missing values (NAs).

First, let's identify and remove columns that consist mostly of NAs. We'll set a threshold and remove any column where more than 90% of the values are missing.

```
# Identify columns with a high percentage of missing values
na_cols <- which(colSums(is.na(pml.training)) > 0.9 * nrow(pml.training))
# Remove these columns
training.cleaned <- pml.training[, -na_cols]
```

Next, we remove identifier columns and metadata that are not useful for predicting the `classe`. These include row numbers, user names, timestamps, and window identifiers.

```
# Identify and remove irrelevant columns
identifier_cols <- c("X", "user_name", "raw_timestamp_part_1", "raw_timestamp_part_2", "cv
td_timestamp", "new_window", "num_window")
training.cleaned <- training.cleaned %>%
  select(-one_of(identifier_cols))
```

After cleaning, let's check the new dimensions of our training data.

```
dim(training.cleaned)
```

```
## [1] 19622    53
```

Now the data is much cleaner, with fewer columns, and is ready for partitioning and model training.

3 3. Model Building

3.1 3.1. Data Partitioning

To evaluate our model's performance, we need to split the cleaned training data into two parts: a training set and a validation set. We will use 70% of the data for training the model and the remaining 30% for validation. This allows us to estimate the out-of-sample error.

We'll use the `createDataPartition` function from the `caret` package to create a balanced split based on the `classe` outcome variable.

```
set.seed(12345) # for reproducibility
inTrain <- createDataPartition(y = training.cleaned$classe, p = 0.7, list = FALSE)
trainSet <- training.cleaned[inTrain, ]
validationSet <- training.cleaned[-inTrain, ]

# Check dimensions of the partitioned data
dim(trainSet)
```

```
## [1] 13737    53
```

```
dim(validationSet)
```

```
## [1] 5885   53
```

3.2 3.2. Model Training

We will use the Random Forest algorithm to train our prediction model. Random Forest is a strong classifier, well-suited for this type of problem as it can handle a large number of predictors and is generally robust to overfitting.

We will use the `train` function from the `caret` package, specifying the method as `"rf"` for Random Forest. We'll also use 5-fold cross-validation to ensure our model's performance is stable and to get a reliable estimate of its accuracy.

```
# Set up cross-validation with 5 folds
fitControl <- trainControl(method = "cv", number = 5)

# Train the Random Forest model
# The formula classe ~ . means we use all other columns as predictors
modelFit <- train(classe ~ ., data = trainSet, method = "rf", trControl = fitControl)

# Print the model summary
modelFit
```

```
## Random Forest
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10987, 10990, 10990, 10991, 10990
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9911914  0.9888567
##   27    0.9902453  0.9876601
##   52    0.9846405  0.9805695
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

4 4. Model Evaluation

4.1 4.1. Performance on Validation Set

Now that we have trained our model, we will evaluate its performance on the validation set, which the model has not seen before. This will give us an estimate of the out-of-sample error.

We use the `predict` function with our trained model (`modelFit`) to make predictions on the `validationSet` . Then, we use the `confusionMatrix` function to compare the predicted outcomes with the actual `classe` values.

```
# Make predictions on the validation set
predictions <- predict(modelFit, newdata = validationSet)

# Get the confusion matrix
confMatrix <- confusionMatrix(predictions, as.factor(validationSet$classe))
confMatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1673    4    0    0    0
##          B    1 1135    8    0    0
##          C    0    0 1017   22    0
##          D    0    0    1  940    1
##          E    0    0    0    2 1081
##
## Overall Statistics
##
##                Accuracy : 0.9934
##                  95% CI : (0.991, 0.9953)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9916
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9994   0.9965   0.9912   0.9751   0.9991
## Specificity            0.9991   0.9981   0.9955   0.9996   0.9996
## Pos Pred Value         0.9976   0.9921   0.9788   0.9979   0.9982
## Neg Pred Value         0.9998   0.9992   0.9981   0.9951   0.9998
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2843   0.1929   0.1728   0.1597   0.1837
## Detection Prevalence   0.2850   0.1944   0.1766   0.1601   0.1840
## Balanced Accuracy      0.9992   0.9973   0.9934   0.9873   0.9993
```

The accuracy on the validation set is very high, which suggests that our model generalizes well to new data. The out-of-sample error can be estimated as `1 - Accuracy`.

5 5. Prediction on Test Data

Finally, we will apply our trained model to the `pml.testing` dataset to predict the `classe` for the 20 provided test cases. Before making predictions, we must apply the same preprocessing steps to the test data as we did to the training data.

```
# Apply the same column removal to the test set
# First, identify columns with a high percentage of missing values in the ORIGINAL trainin
g data
# This ensures we use the same set of columns as removed from the training set
na_cols_test <- which(colSums(is.na(pml.training)) > 0.9 * nrow(pml.training))
testing.cleaned <- pml.testing[, -na_cols_test]

# Remove the same identifier columns from the test set
identifier_cols_test <- c("X", "user_name", "raw_timestamp_part_1", "raw_timestamp_part_
2", "cvtd_timestamp", "new_window", "num_window")
testing.cleaned <- testing.cleaned %>%
  select(-one_of(identifier_cols_test))

# Check dimensions of the cleaned test data
dim(testing.cleaned)
```

```
## [1] 20 53
```

Now, we make predictions using our `modelFit` .

```
predictions_test <- predict(modelFit, newdata = testing.cleaned)
predictions_test
```

```
##  [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
```

These are the predictions for the 20 test cases. These predictions should be submitted to the Course Project Prediction Quiz.

6 6. Conclusion

In this project, we successfully built a predictive model using accelerometer data to classify the manner in which participants performed a barbell lift exercise. We followed a structured approach:

1. **Data Loading and Preprocessing**: We loaded the training and testing datasets, identified and removed columns with a high percentage of missing values, and discarded irrelevant identifier columns.
2. **Data Partitioning**: The cleaned training data was split into a training set (70%) and a validation set (30%) to allow for robust model evaluation.
3. **Model Training**: A Random Forest model was trained on the training set using 5-fold cross-validation. Random Forest was chosen for its ability to handle complex relationships and its robustness.
4. **Model Evaluation**: The trained model achieved high accuracy on the validation set, indicating good generalization capability and a low estimated out-of-sample error.
5. **Prediction**: Finally, the model was applied to the unseen test data to predict the `classe` for 20 new cases.

This project demonstrates the effectiveness of machine learning techniques, particularly Random Forest, in classifying human activities based on sensor data. The high accuracy achieved suggests that the model is well-suited for this prediction task.