Choose Your Own Project

Ilse Tromp

2024-08-13

Introduction

This is the last graded project of the Capstone course of the Harvard Data Science series. For this project, I have chosen a data set to practice my machine learning knowledge. The data set that I will be using is the "Human Activity Recognition Using Smartphones" (HARUS) data set. The HARUS data set consists of the recordings from 30 participants who performed daily tasks while wearing a smartphone around their waist.

The participants were aged between 19 and 48 years of age and each participant performed 6 daily activities. These activities were: walking, walking downstairs, sitting, standing, and laying. The sampling rate of the recordings was 50Hz and stored as time series per dimension. So, six different signals were measured: 3 from the accelerometer and 3 from the gyroscope.

The data set has already been divided into training and test sets. Of the participants, 70% was randomly selected to form the training set, and 30% was selected to form the test set.

In this project, I will develop a model that determines the type of activity, based on data collected by smartphone sensors.

Before we start, we need to load the needed libraries.

```
if (!require(e1071)) install.packages("dplyr")
```

```
## Loading required package: e1071
```

```
if (!require(e1071)) install.packages("tidyverse")
if (!require(e1071)) install.packages("caret")
if (!require(e1071)) install.packages("ggplot2")
if (!require(e1071)) install.packages("rpart")
if (!require(e1071)) install.packages("rpart.plot")
if (!require(e1071)) install.packages("randomForest")
if (!require(e1071)) install.packages("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
```

```
library("tidyverse")
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v forcats   1.0.0     v readr     2.1.5
## v ggplot2   3.5.1     v stringr   1.5.1
## v lubridate 1.9.3     v tibble    3.2.1
## v purrr     1.0.2     v tidyr     1.3.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library("caret")
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library("ggplot2")
library("rpart")
library("rpart.plot")
library("randomForest")
```

```
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:ggplot2':
##
##     margin
##
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library("e1071")
```

Loading the data

The data has already been split into training and test sets. So, we can load the data immediately.

```r
# Loading the train data
X_train <- read.table("X_train.txt")
y_train <- read.table("y_train.txt")
subject_train <- read.table("subject_train.txt")

# Loading the test data
X_test <- read.table("X_test.txt")
y_test <- read.table("y_test.txt")
subject_test <- read.table("subject_test.txt")

# Loading features and activity labels
features <- read.table("features.txt")
feature_names <- as.character(features$V2)
activity_labels <- read.table("activity_labels.txt")

# Assigning an ID and name to activities
colnames(activity_labels) <- c("ActivityID", "ActivityName")

# Assigning feature names to the X variables
colnames(X_train) <- feature_names
colnames(X_test) <- feature_names

# Assigning column names to the y and subject data
colnames(y_train) <- "Activity"
colnames(subject_train) <- "Subject"
colnames(y_test) <- "Activity"
colnames(subject_test) <- "Subject"

# Combining train data into one
train_data <- cbind(subject_train, y_train, X_train)

# Combining test data into one
test_data <- cbind(subject_test, y_test, X_test)

# Checking for duplicate columns and rename if necessary
colnames(train_data) <- make.names(colnames(train_data), unique = TRUE)
colnames(test_data) <- make.names(colnames(test_data), unique = TRUE)

# Transforming activity into a categorical variable
train_data$Activity <- factor(train_data$Activity, levels = activity_labels$ActivityID, labels = activi
test_data$Activity <- factor(test_data$Activity, levels = activity_labels$ActivityID, labels = activity
```

Next we will combine the data into one set for possible data exploration.

```r
combined_data <- rbind(
  cbind(subject_train, y_train, X_train),
  cbind(subject_test, y_test, X_test)
)
```

Missing values

Let's make sure there are no missing values.

```r
# Checking for missing values
sum(is.na(combined_data))
```

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

Data exploration

First, let's examine how the activities are distributed.

```
##
##    1    2    3    4    5    6
## 1722 1544 1406 1777 1906 1944
```

Next, we will make sure there are no duplicate column names and make a barplot of the different activities.

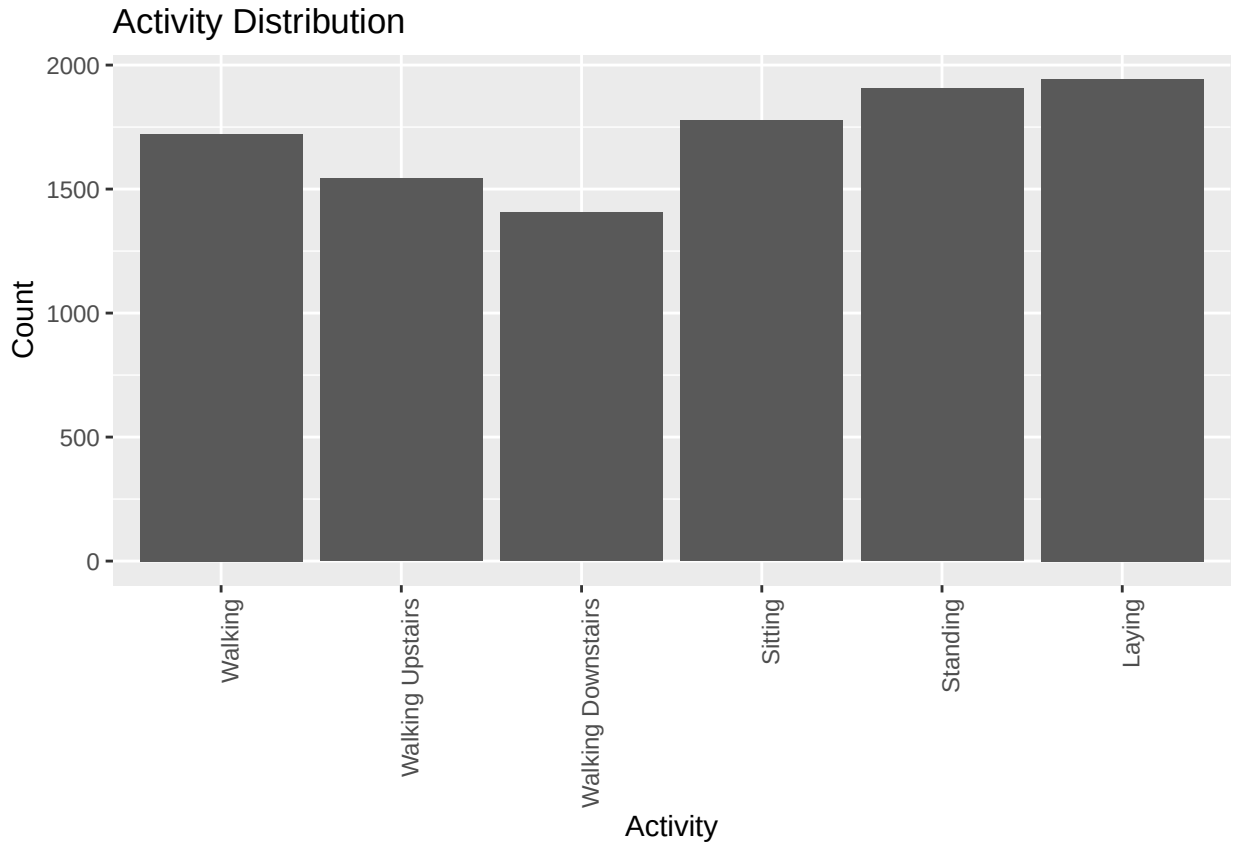
```r
# Checking for duplicate column names
duplicate_columns <- duplicated(colnames(combined_data))

# Showing duplicate column names
duplicate_column_names <- colnames(combined_data)[duplicate_columns]
duplicate_column_names
```

```
##  [1] "fBodyAcc-bandsEnergy()-1,8"       "fBodyAcc-bandsEnergy()-9,16"
##  [3] "fBodyAcc-bandsEnergy()-17,24"     "fBodyAcc-bandsEnergy()-25,32"
##  [5] "fBodyAcc-bandsEnergy()-33,40"     "fBodyAcc-bandsEnergy()-41,48"
##  [7] "fBodyAcc-bandsEnergy()-49,56"     "fBodyAcc-bandsEnergy()-57,64"
##  [9] "fBodyAcc-bandsEnergy()-1,16"      "fBodyAcc-bandsEnergy()-17,32"
## [11] "fBodyAcc-bandsEnergy()-33,48"     "fBodyAcc-bandsEnergy()-49,64"
## [13] "fBodyAcc-bandsEnergy()-1,24"      "fBodyAcc-bandsEnergy()-25,48"
## [15] "fBodyAcc-bandsEnergy()-1,8"       "fBodyAcc-bandsEnergy()-9,16"
## [17] "fBodyAcc-bandsEnergy()-17,24"     "fBodyAcc-bandsEnergy()-25,32"
## [19] "fBodyAcc-bandsEnergy()-33,40"     "fBodyAcc-bandsEnergy()-41,48"
## [21] "fBodyAcc-bandsEnergy()-49,56"     "fBodyAcc-bandsEnergy()-57,64"
## [23] "fBodyAcc-bandsEnergy()-1,16"      "fBodyAcc-bandsEnergy()-17,32"
## [25] "fBodyAcc-bandsEnergy()-33,48"     "fBodyAcc-bandsEnergy()-49,64"
## [27] "fBodyAcc-bandsEnergy()-1,24"      "fBodyAcc-bandsEnergy()-25,48"
## [29] "fBodyAccJerk-bandsEnergy()-1,8"   "fBodyAccJerk-bandsEnergy()-9,16"
## [31] "fBodyAccJerk-bandsEnergy()-17,24" "fBodyAccJerk-bandsEnergy()-25,32"
## [33] "fBodyAccJerk-bandsEnergy()-33,40" "fBodyAccJerk-bandsEnergy()-41,48"
## [35] "fBodyAccJerk-bandsEnergy()-49,56" "fBodyAccJerk-bandsEnergy()-57,64"
## [37] "fBodyAccJerk-bandsEnergy()-1,16"  "fBodyAccJerk-bandsEnergy()-17,32"
## [39] "fBodyAccJerk-bandsEnergy()-33,48" "fBodyAccJerk-bandsEnergy()-49,64"
## [41] "fBodyAccJerk-bandsEnergy()-1,24"  "fBodyAccJerk-bandsEnergy()-25,48"
## [43] "fBodyAccJerk-bandsEnergy()-1,8"   "fBodyAccJerk-bandsEnergy()-9,16"
## [45] "fBodyAccJerk-bandsEnergy()-17,24" "fBodyAccJerk-bandsEnergy()-25,32"
## [47] "fBodyAccJerk-bandsEnergy()-33,40" "fBodyAccJerk-bandsEnergy()-41,48"
## [49] "fBodyAccJerk-bandsEnergy()-49,56" "fBodyAccJerk-bandsEnergy()-57,64"
## [51] "fBodyAccJerk-bandsEnergy()-1,16"  "fBodyAccJerk-bandsEnergy()-17,32"
## [53] "fBodyAccJerk-bandsEnergy()-33,48" "fBodyAccJerk-bandsEnergy()-49,64"
## [55] "fBodyAccJerk-bandsEnergy()-1,24"  "fBodyAccJerk-bandsEnergy()-25,48"
## [57] "fBodyGyro-bandsEnergy()-1,8"      "fBodyGyro-bandsEnergy()-9,16"
```

```
## [59] "fBodyGyro-bandsEnergy()-17,24"    "fBodyGyro-bandsEnergy()-25,32"
## [61] "fBodyGyro-bandsEnergy()-33,40"    "fBodyGyro-bandsEnergy()-41,48"
## [63] "fBodyGyro-bandsEnergy()-49,56"    "fBodyGyro-bandsEnergy()-57,64"
## [65] "fBodyGyro-bandsEnergy()-1,16"     "fBodyGyro-bandsEnergy()-17,32"
## [67] "fBodyGyro-bandsEnergy()-33,48"    "fBodyGyro-bandsEnergy()-49,64"
## [69] "fBodyGyro-bandsEnergy()-1,24"     "fBodyGyro-bandsEnergy()-25,48"
## [71] "fBodyGyro-bandsEnergy()-1,8"      "fBodyGyro-bandsEnergy()-9,16"
## [73] "fBodyGyro-bandsEnergy()-17,24"    "fBodyGyro-bandsEnergy()-25,32"
## [75] "fBodyGyro-bandsEnergy()-33,40"    "fBodyGyro-bandsEnergy()-41,48"
## [77] "fBodyGyro-bandsEnergy()-49,56"    "fBodyGyro-bandsEnergy()-57,64"
## [79] "fBodyGyro-bandsEnergy()-1,16"     "fBodyGyro-bandsEnergy()-17,32"
## [81] "fBodyGyro-bandsEnergy()-33,48"    "fBodyGyro-bandsEnergy()-49,64"
## [83] "fBodyGyro-bandsEnergy()-1,24"     "fBodyGyro-bandsEnergy()-25,48"
```

```r
# Renaming duplicates
colnames(combined_data) <- make.names(colnames(combined_data), unique = TRUE)
```



Methods

To predict the right activity based on the smartphone sensor data, I will use several models and test the accuracy of each. First, I will develop a baseline model to compare the results to.

Baseline model

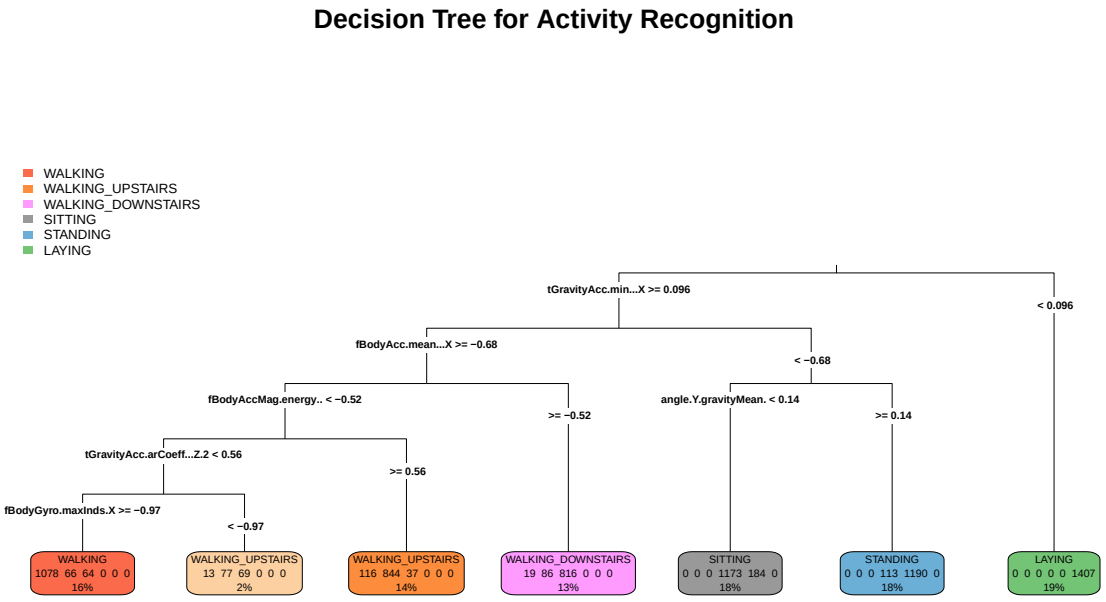
The baseline model I will use, is a decision tree. A decision tree splits the data its given into smaller sets at each decision node. This way, what's left in the end is hopefully the type of activity that corresponds to the data.

Decision Tree

We first start by fitting the model. We use the "class" method, because we want to classify activity type.

```
# Fitting the decision tree model
tree_model <- rpart(Activity ~ ., data = train_data, method = "class")
```

Next, we can see the decision tree.



We now use the decision tree model to predict activity type in the test set.

```
# Predicting on the test set
tree_predictions <- predict(tree_model, test_data, type = "class")
```

Finally, we calculate the confusion matrix and show the accuracy of this model.

```
## Confusion Matrix and Statistics
##
##                     Reference
```

```
## Prediction           WALKING WALKING_UPSTAIRS WALKING_DOWNSTAIRS SITTING
##   WALKING                430               61                40       0
##   WALKING_UPSTAIRS        55              402                98       0
##   WALKING_DOWNSTAIRS      11                8               282       0
##   SITTING                  0                0                 0     400
##   STANDING                 0                0                 0      91
##   LAYING                   0                0                 0       0
##                     Reference
## Prediction           STANDING LAYING
##   WALKING                   0      0
##   WALKING_UPSTAIRS          0      0
##   WALKING_DOWNSTAIRS        0      0
##   SITTING                 107      0
##   STANDING                425      0
##   LAYING                    0    537
##
## Overall Statistics
##
##                Accuracy : 0.8402
##                  95% CI : (0.8264, 0.8532)
##     No Information Rate : 0.1822
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8078
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: WALKING Class: WALKING_UPSTAIRS
## Sensitivity                  0.8669                  0.8535
## Specificity                  0.9588                  0.9382
## Pos Pred Value               0.8098                  0.7243
## Neg Pred Value               0.9727                  0.9712
## Prevalence                   0.1683                  0.1598
## Detection Rate               0.1459                  0.1364
## Detection Prevalence         0.1802                  0.1883
## Balanced Accuracy            0.9129                  0.8959
##                      Class: WALKING_DOWNSTAIRS Class: SITTING Class: STANDING
## Sensitivity                            0.67143         0.8147          0.7989
## Specificity                            0.99248         0.9564          0.9623
## Pos Pred Value                         0.93688         0.7890          0.8236
## Neg Pred Value                         0.94785         0.9627          0.9560
## Prevalence                             0.14252         0.1666          0.1805
## Detection Rate                         0.09569         0.1357          0.1442
## Detection Prevalence                   0.10214         0.1720          0.1751
## Balanced Accuracy                      0.83195         0.8855          0.8806
##                      Class: LAYING
## Sensitivity                 1.0000
## Specificity                 1.0000
## Pos Pred Value              1.0000
## Neg Pred Value              1.0000
## Prevalence                  0.1822
## Detection Rate              0.1822
```

```
## Detection Prevalence        0.1822
## Balanced Accuracy           1.0000

##           Model  Accuracy
## 1 Decision Tree 0.8401765
```

Random Forest

The first model I will test and compare to the baseline model, is random forests. Random forests is a model that is made up of several decision trees. In this case, the variable needs to be classified. I.E., the type of activity is determined. So, in the end the most frequent activity type over all decision trees will be predicted.

Let's first set a seed for reproducibility.

```
set.seed(123)
```

We again start by training the model on the training set.

```
# Training the Random Forest model
rf_model <- randomForest(Activity ~ ., data = train_data, ntree = 100, importance = TRUE)

rf_model
```

```
##
## Call:
##  randomForest(formula = Activity ~ ., data = train_data, ntree = 100,      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 23
##
##         OOB estimate of  error rate: 1.89%
## Confusion matrix:
##                     WALKING WALKING_UPSTAIRS WALKING_DOWNSTAIRS SITTING STANDING
## WALKING                1207               10                  9       0        0
## WALKING_UPSTAIRS          3             1064                  6       0        0
## WALKING_DOWNSTAIRS        4                9                973       0        0
## SITTING                   0                1                  0    1238       47
## STANDING                  0                0                  0      50     1324
## LAYING                    0                0                  0       0        0
##                     LAYING class.error
## WALKING                  0 0.015497553
## WALKING_UPSTAIRS         0 0.008387698
## WALKING_DOWNSTAIRS       0 0.013184584
## SITTING                  0 0.037325039
## STANDING                 0 0.036390102
## LAYING                1407 0.000000000
```

Next, we use this model to make predictions on the test set.

```
# Using the model to predict based on the test set
rf_predictions <- predict(rf_model, test_data)
```

Finally, we again calculate the confusion matrix and show the accuracy.

```
## Confusion Matrix and Statistics
##
##                     Reference
## Prediction           WALKING WALKING_UPSTAIRS WALKING_DOWNSTAIRS SITTING
##   WALKING                482               30                 19       0
##   WALKING_UPSTAIRS         7              433                 43       0
##   WALKING_DOWNSTAIRS       7                8                358       0
##   SITTING                  0                0                  0     431
##   STANDING                 0                0                  0      60
##   LAYING                   0                0                  0       0
##                     Reference
## Prediction           STANDING LAYING
##   WALKING                   0      0
##   WALKING_UPSTAIRS          0      0
##   WALKING_DOWNSTAIRS        0      0
##   SITTING                  37      0
##   STANDING                495      0
##   LAYING                    0    537
##
## Overall Statistics
##
##                Accuracy : 0.9284
##                  95% CI : (0.9185, 0.9375)
##     No Information Rate : 0.1822
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9139
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: WALKING Class: WALKING_UPSTAIRS
## Sensitivity                  0.9718                 0.9193
## Specificity                  0.9800                 0.9798
## Pos Pred Value               0.9077                 0.8965
## Neg Pred Value               0.9942                 0.9846
## Prevalence                   0.1683                 0.1598
## Detection Rate               0.1636                 0.1469
## Detection Prevalence         0.1802                 0.1639
## Balanced Accuracy            0.9759                 0.9496
##                      Class: WALKING_DOWNSTAIRS Class: SITTING Class: STANDING
## Sensitivity                             0.8524         0.8778          0.9305
## Specificity                             0.9941         0.9849          0.9752
## Pos Pred Value                          0.9598         0.9209          0.8919
## Neg Pred Value                          0.9759         0.9758          0.9845
## Prevalence                              0.1425         0.1666          0.1805
## Detection Rate                          0.1215         0.1463          0.1680
## Detection Prevalence                    0.1266         0.1588          0.1883
## Balanced Accuracy                       0.9232         0.9314          0.9528
##                      Class: LAYING
## Sensitivity                 1.0000
## Specificity                 1.0000
## Pos Pred Value              1.0000
```

```
## Neg Pred Value              1.0000
## Prevalence                  0.1822
## Detection Rate              0.1822
## Detection Prevalence        0.1822
## Balanced Accuracy           1.0000
```

```
## Accuracy:  0.9284018
```

```
##           Model  Accuracy
## 1 Decision Tree 0.8401765
## 2 Random Forest 0.9284018
```

Support Vector Machines

The next and last model we will test is the Support Vector Machines (SVM) model. SVM tries to find a hyperplane that separates the points of different classes. SVM then tries to find the hyperplane that has the largest distance between the points from the different classes.

We first start again by setting a seed for reproducibility.

```
set.seed(123)
```

Next, we train the SVM model, using the training set.

```
# Training the SVM model with a linear kernel
svm_model <- svm(Activity ~ ., data = train_data, kernel = "linear", scale = TRUE)

svm_model
```

```
##
## Call:
## svm(formula = Activity ~ ., data = train_data, kernel = "linear",
##     scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  799
```

Now, we predict activity type on the test set using the SVM model.

```
# Predicting on the test set
svm_predictions <- predict(svm_model, test_data)
```

And finally, we calculate the confusion matrix and show the accuracy.

```
## Confusion Matrix and Statistics
##
```

```
##                     Reference
## Prediction           WALKING WALKING_UPSTAIRS WALKING_DOWNSTAIRS SITTING
##   WALKING                495               16                  6       0
##   WALKING_UPSTAIRS         0              453                 15       1
##   WALKING_DOWNSTAIRS       1                2                399       0
##   SITTING                  0                0                  0     422
##   STANDING                 0                0                  0      67
##   LAYING                   0                0                  0       1
##                     Reference
## Prediction           STANDING LAYING
##   WALKING                   0      0
##   WALKING_UPSTAIRS          0      0
##   WALKING_DOWNSTAIRS        0      0
##   SITTING                  19      0
##   STANDING                513      0
##   LAYING                    0    537
##
## Overall Statistics
##
##                Accuracy : 0.9566
##                  95% CI : (0.9486, 0.9636)
##     No Information Rate : 0.1822
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9478
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: WALKING Class: WALKING_UPSTAIRS
## Sensitivity                  0.9980                  0.9618
## Specificity                  0.9910                  0.9935
## Pos Pred Value               0.9574                  0.9659
## Neg Pred Value               0.9996                  0.9927
## Prevalence                   0.1683                  0.1598
## Detection Rate               0.1680                  0.1537
## Detection Prevalence         0.1754                  0.1591
## Balanced Accuracy            0.9945                  0.9777
##                      Class: WALKING_DOWNSTAIRS Class: SITTING Class: STANDING
## Sensitivity                             0.9500         0.8595          0.9643
## Specificity                             0.9988         0.9923          0.9723
## Pos Pred Value                          0.9925         0.9569          0.8845
## Neg Pred Value                          0.9917         0.9725          0.9920
## Prevalence                              0.1425         0.1666          0.1805
## Detection Rate                          0.1354         0.1432          0.1741
## Detection Prevalence                    0.1364         0.1496          0.1968
## Balanced Accuracy                       0.9744         0.9259          0.9683
##                      Class: LAYING
## Sensitivity                 1.0000
## Specificity                 0.9996
## Pos Pred Value              0.9981
## Neg Pred Value              1.0000
## Prevalence                  0.1822
```

```
## Detection Rate              0.1822
## Detection Prevalence        0.1826
## Balanced Accuracy           0.9998
```

```
## Accuracy:  0.956566
```

```
##           Model  Accuracy
## 1 Decision Tree 0.8401765
## 2 Random Forest 0.9284018
## 3           SVM 0.9565660
```

Results

We can see that the baseline model (decision tree) has an accuracy of 0.840, Random Forest has an accuracy of 0.928, and SVM has an accurary of 0.957.

```
##           Model  Accuracy
## 1 Decision Tree 0.8401765
## 2 Random Forest 0.9284018
## 3           SVM 0.9565660
```

Conclusion

In this project I compared two models, Random Forest and SVM, to the baseline model, decision tree. I used these models to predict activity type, based on data collected by smartphone sensors.

The baseline model already predicted quite well, with an accuracy of 0.840. The RF model was a good improvement with an accuracy of 0.928. And the SVM model worked best, with an accuracy of 0.957.

This project has some limitations. First of all, the data was recorded from only 30 participants. This could cause, that, when these models would be used on "real world" data, It could be less precise in predicting activity type correctly. Furthermore, the activity types were recorded in a controlled setting. Again, when "real world" data would be used, It could be quite noisy and hard to predict. Lastly, although RF and SVM seem like the right models, these could still have overfitted the data. This would mean, that although these models seem to work good on the test data, they might not on real data.

Finally, for the future, it would be interesting to see how these models would do with "real world" data.

References

Reyes-Ortiz,Jorge, Anguita,Davide, Ghio,Alessandro, Oneto,Luca, and Parra,Xavier. (2012). Human Activity Recognition Using Smartphones. UCI Machine Learning Repository. https://doi.org/10.24432/C54S4K.

R.A. Irizarry. (2019). Introduction to Data Science.

IBM. "What Is Random Forest?" IBM, https://www.ibm.com/topics/random-forest#:~:text=Random%20forest%20is%20a%
Accessed 13 Aug. 2024.

GeeksforGeeks. "Support Vector Machine Algorithm." GeeksforGeeks, https://www.geeksforgeeks.org/support-vector-machine-algorithm/. Accessed 13 Aug. 2024.