Human Activity Recognition Using Machine Learning

David Owuor

2025-04-17

Introduction

This is a case study for Human Activity Recognition using data collected by smart devices (smartphones and smart watches). The experiments were done on volunteers within an age bracket of 19-48 years. Each person performed six different activities (WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING, LAYING) while wearing a smartphone on the waist. These smart devices contain two key types of sensors i.e. Accelerometer and Gyroscope capable of measuring the body orientation and motion in reference to the ground. Accelerometer measures triaxial acceleration and the estimated body acceleration while gyroscope measures triaxial angular velocity. The data for the experiment was collected by these smart devices and sent to remote cloud servers. The downloaded data was provided by New England College (USA) in their Machine Learning Course.

The aim is to reduce the dimensionality of the data, then train a classification model that can accurately identify the activity being performed.

```
# Load packages
suppressPackageStartupMessages(
  {
    library(tidyverse)
    library(caret)
    library(parallel)
    library(parallelMap)
  }
)

# Import data
HumanActivityRecognition <- read_csv("HumanActivityRecognition.csv")

## New names:
## Rows: 10297 Columns: 564
## ── Column specification
## ──────────────────────────────────────────────────────── Delimiter: ","
chr
## (1): Activity dbl (563): ...1, 1 tBodyAcc-mean()-X, 2 tBodyAcc-mean()-Y, 3
## tBodyAcc-mean()...
## ℹ Use `spec()` to retrieve the full column specification for this data. ℹ
```

```
## Specify the column types or set `show_col_types = FALSE` to quiet this
message.
## • `` -> `...1`
```

The data has 10297 observations of 564 variables. The target variable Activity is of character type, while the rest of the variables are all numeric.

```
# View the first few observations
head(HumanActivityRecognition)
```

```
## # A tibble: 6 × 564
##    ...1 `1 tBodyAcc-mean()-X` `2 tBodyAcc-mean()-Y` `3 tBodyAcc-mean()-Z`
##   <dbl>                 <dbl>                 <dbl>                 <dbl>
## 1     1                 0.278               -0.0164                -0.124
## 2     2                 0.28                -0.0195                -0.113
## 3     3                 0.279               -0.0262                -0.123
## 4     4                 0.277               -0.0166                -0.115
## 5     5                 0.277               -0.0101                -0.105
## 6     6                 0.279               -0.0196                -0.11
## # ℹ 560 more variables: `4 tBodyAcc-std()-X` <dbl>, `5 tBodyAcc-std()-Y`
<dbl>,
##   `6 tBodyAcc-std()-Z` <dbl>, `7 tBodyAcc-mad()-X` <dbl>,
##   `8 tBodyAcc-mad()-Y` <dbl>, `9 tBodyAcc-mad()-Z` <dbl>,
##   `10 tBodyAcc-max()-X` <dbl>, `11 tBodyAcc-max()-Y` <dbl>,
##   `12 tBodyAcc-max()-Z` <dbl>, `13 tBodyAcc-min()-X` <dbl>,
##   `14 tBodyAcc-min()-Y` <dbl>, `15 tBodyAcc-min()-Z` <dbl>,
##   `16 tBodyAcc-sma()` <dbl>, `17 tBodyAcc-energy()-X` <dbl>, …
```

The first column contains the row numbers. All the predictors are numeric and most have values ranging between -1 and 1. Feature scaling won't be very necessary, but will be done because the variables have different variances.

Data Cleaning

Data cleaning will involve assessing the data for missing values and duplicated entries.

```
# Check for missing values
sum(is.na(HumanActivityRecognition))
```

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

The data has no missing values.

```
# Check for duplicated observations
sum(duplicated(HumanActivityRecognition))
```

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

There are no duplicated observations in the data as well. I'll now convert the target variable into a nominal factor as needed for classification algorithms.

```
# Convert the target variable Activity into a factor
HumanActivityRecognition$Activity <-
factor(HumanActivityRecognition$Activity)

# Table frequencies of the target class labels
table(HumanActivityRecognition$Activity)

##
##             LAYING            SITTING           STANDING
WALKING
##               1923               1775               1902
1769
## WALKING_DOWNSTAIRS   WALKING_UPSTAIRS
##               1397               1531
```

Laying had the most activities, followed by standing, sitting, walking, walking upstairs and walking downstairs respectively. There is class imbalance in the data, but not so high. The data is so large, making it difficult to perform EDA. So I'll go directly to dimensionality reduction and model training.

Dimensionality Reduction

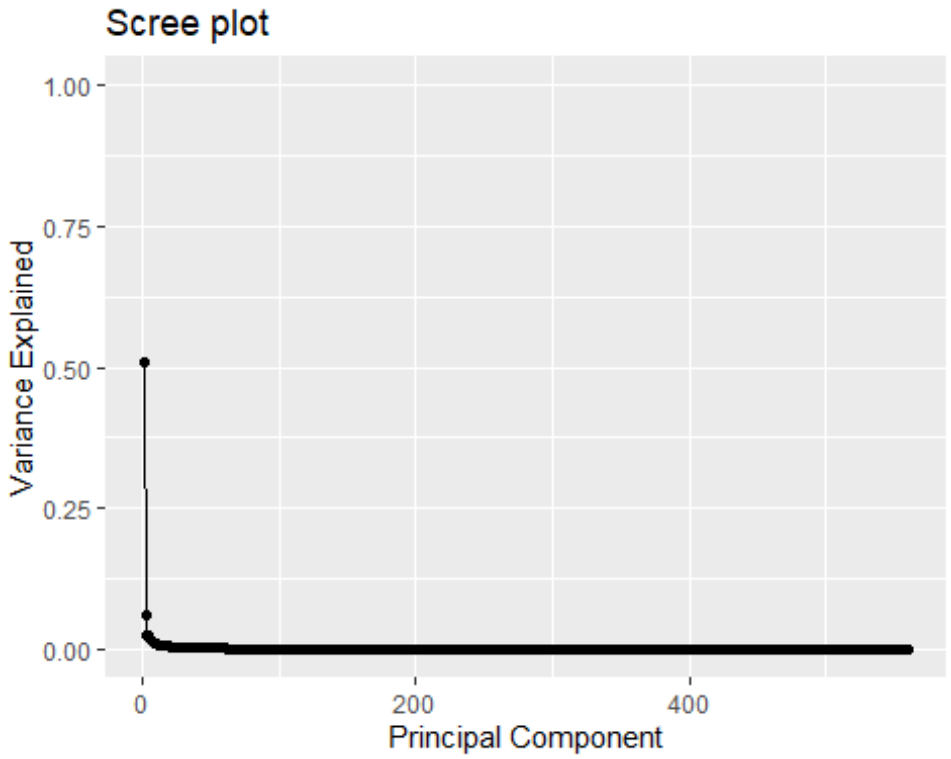
I'll use PCA for dimensionality reduction. PCA requires all the variable to be numeric, so I'll omit the target variable Activity from PCA. I'll also omit the first variable containing the row numbers, and the second last variable (y) which contains value labels for the target. PCA also requires feature scaling. If features are not scaled, those with larger variances will dominate the principal components.

```
# Perform variable selection for PCA (Omit the first variable, y and
Activity)
x <- HumanActivityRecognition[, -c(1,563,564)]
# Scale the data
x_scaled <- scale(x, center = TRUE, scale = TRUE)

# Perform PCA
pca <- princomp(x_scaled)
```

Principal Components are contained in the object named scores, and can be accessed by *$scores*.

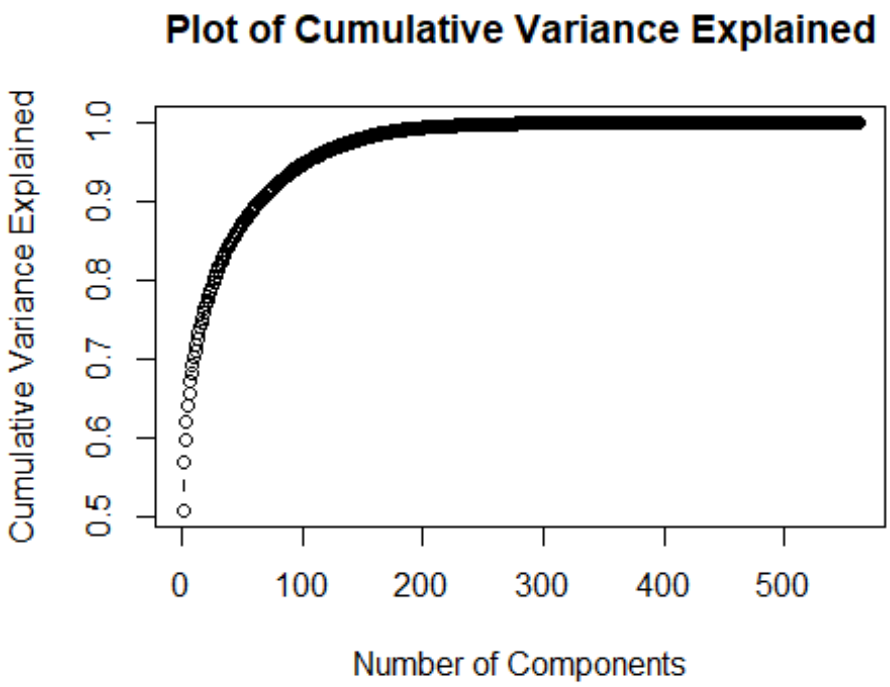
```
# Get the variance explained by each principal component
variance_explained <- ((pca$sdev ^ 2)/(sum(pca$sdev ^ 2)))

# Create a scree plot of the variance explained by each Principal Component
qplot(c(1:561), variance_explained) + geom_line() + ylim(0,1) +
  labs(title = "Scree plot", x = "Principal Component",
       y = "Variance Explained")
```

```
## Warning: `qplot()` was deprecated in ggplot2 3.4.0.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



The first principal component explains the maximum variance in the data (about 50%), followed by the second, third and so on.. From the plot, most of the principal components have variance explained close to zero. I'll compute the cumulative variance explained to help me decide on the number of principal components to retain.

```r
# Get the cumulative variance explained
cumulative_var_explained <- cumsum(variance_explained)

# Plot the cumulative variance explained
plot(cumulative_var_explained, type = "b",
     xlab = "Number of Components",
     ylab = "Cumulative Variance Explained",
     main = "Plot of Cumulative Variance Explained")
```



The first 100 principal components seem to have a cumulative variance explained of about 90%, implying that these 100 PCs explains most of the variability in the data. I'll retain principal components with explains at most 95% of the variance.

```
## Get the number of PCs to retain, based on cumulative variance explained

# Retain components with cumulative variance <= 0.95
num_components <- which(cumulative_var_explained <= 0.95)

# Extract the retained Principal Components from PCA results
selectedPC_scores <- pca$scores[ ,num_components]

# Get the dimension of the extracted Principal Components data
dim(selectedPC_scores)

## [1] 10297   103
```

103 principal components that explain 95% of the total variability in the data have been retained. PCA has indeed helped in reducing the dimension of the data from 561 to 103 features. I'll now use these retained PCs to train my classification models.

```
# Convert the selected Principal Component scores into a data frame
selectedPC_scores <- as.data.frame(selectedPC_scores)
# Add a column for the target variable
selectedPC_scores <- selectedPC_scores |>
  mutate(Activity = HumanActivityRecognition$Activity)
```

Data Partitioning

The data is large and I'll partition it into into training and test sets, using 70/30 split. That is, 70% of the data will be allocated for model training, and the remaining 30% will be used for model evaluation.

```
## Partition the data into training and test sets

# Set random seed for reproducibility
set.seed(105)
# partition the data
train_index <- createDataPartition(selectedPC_scores$Activity, p = 0.70, list
= FALSE)
# Assign 70% to training set
training_data <- selectedPC_scores[train_index,]
# Assign training set the remaining 30%
test_data <- selectedPC_scores[-train_index,]
```

Training set has 7211 instances, while test set has 3086 instances.

Model Training

I'll try 6 algorithms (QDA, KNN, RF, SVM, Neural Net and XGBoost). Even if the dimensionality of the data was reduced by PCA, the data is still large and I will not tune all the hyperparameters for each and every model due to the limited computational resources I have. I'll utilize all the available CPU cores to perform parallel processing in order to speed up the model training process.

```
# Set-up the test harness to use 7-fold cross validation
train_control <- trainControl(method = "cv", number = 7)

# Begin parallelization
parallelStartSocket(cpus = detectCores())

## Starting parallelization in mode=socket with cpus=8.

## Train the models

# QDA
set.seed(105) # Random seed number for reproducibility
# train the model
fit.qda <- train(Activity ~ ., data = training_data, method = "qda",
                 trControl = train_control) # QDA has no hyperparameter to
tune

# KNN
set.seed(105)
fit.knn <- train(Activity ~ ., data = training_data, method = "knn",
```

```
                 trControl = train_control, tuneLength = 10)

## Random Forest
set.seed(105)
fit.rf <- train(Activity ~ ., data = training_data, method = "rf",
                trControl = train_control, tuneLength = 5)

## SVM
set.seed(105)
fit.svm <- train(Activity ~ ., data = training_data, method = "svmRadial",
                 trControl = train_control, tuneLength = 5)

## Neural Net
set.seed(105)
fit.nnet <- train(Activity ~ ., data = training_data, method = "nnet",
                  trControl = train_control, tuneLength = 5, trace = FALSE)

## XGBoost
set.seed(105)
fit.xgb <- train(Activity ~ ., data = training_data, method = "xgbTree",
                  trControl = train_control)

## [23:33:05] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated,
use `iteration_range` instead.
```

I used the same seed number to ensure that the results can be directly compared.

```
# stop parallelization
parallelStop()

## Stopped parallelization. All cleaned up.
```

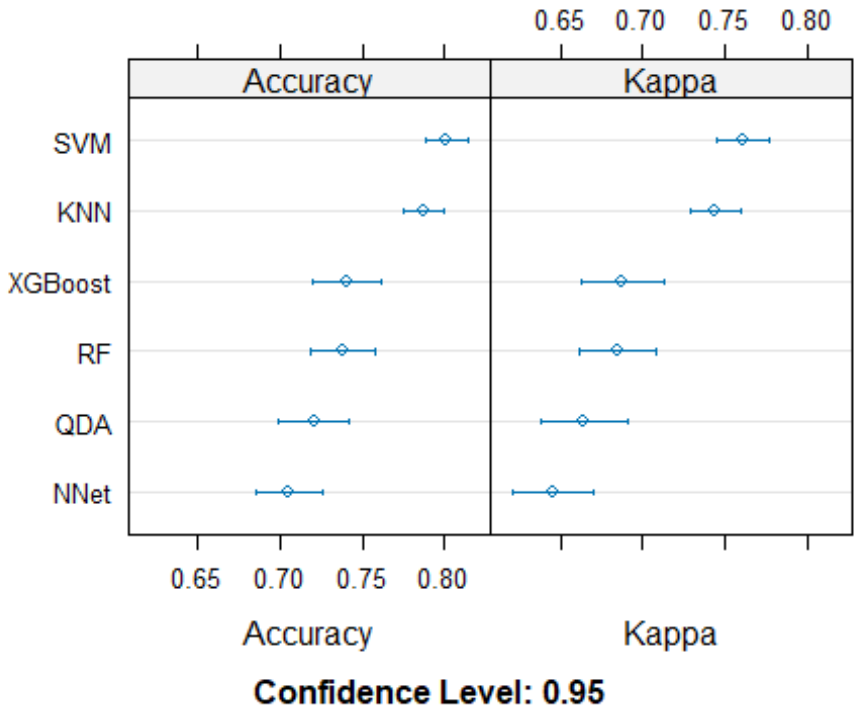
Model Evaluation

```
# Put the fitted models into a list
results <- resamples(list(QDA = fit.qda, KNN = fit.knn, RF = fit.rf,
                          SVM = fit.svm, NNet = fit.nnet, XGBoost = fit.xgb))
# Generate summary statistics of accuracy across each model
summary(results)

##
## Call:
## summary.resamples(object = results)
##
## Models: QDA, KNN, RF, SVM, NNet, XGBoost
## Number of resamples: 7
##
## Accuracy
##              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## QDA     0.6951456 0.7061696 0.7109602 0.7205688 0.7340097 0.7575170    0
## KNN     0.7737864 0.7775619 0.7895247 0.7876797 0.7907750 0.8137730    0
```

```
## RF      0.7135922 0.7225992 0.7376093 0.7383205 0.7498887 0.7720660    0
## SVM     0.7893204 0.7942714 0.7970874 0.8015477 0.8023259 0.8312318    0
## NNet    0.6776699 0.6914128 0.7025194 0.7054561 0.7190148 0.7371484    0
## XGBoost 0.7106796 0.7274488 0.7410281 0.7403971 0.7472046 0.7817653    0
##
## Kappa
##              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## QDA     0.6338773 0.6465944 0.6525789 0.6640587 0.6801970 0.7083718    0
## KNN     0.7275915 0.7320673 0.7466317 0.7442638 0.7478902 0.7757086    0
## RF      0.6554465 0.6658951 0.6841384 0.6849837 0.6989395 0.7256317    0
## SVM     0.7461608 0.7521468 0.7557223 0.7610258 0.7621065 0.7967909    0
## NNet    0.6121874 0.6280860 0.6415203 0.6452470 0.6616680 0.6835131    0
## XGBoost 0.6520018 0.6716129 0.6883080 0.6874871 0.6957447 0.7373849    0
```

SVM performs best with a training accuracy of 80.15%.

```
# Plot results to compare accuracy of models
dotplot(results)
```



SVM and KNN produced more stable results across the cross-validation folds as compared to the other models.

Model Validation

I'll use the best performing model (which is SVM) to make predictions on test set and evaluate how the model performs on new data.

```r
# Make predictions on test dataset
svm_preds <- predict(fit.svm, newdata = test_data)

# Create a confusion matrix to see how the model performs on new data
confusionMatrix(test_data$Activity, svm_preds)
```

```
## Confusion Matrix and Statistics
##
##                     Reference
## Prediction           LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS
##   LAYING                473      32       30      16                  13
##   SITTING                22     440       47       5                  11
##   STANDING               18      47      474       8                  11
##   WALKING                33      33       18     428                  11
##   WALKING_DOWNSTAIRS     21      26       21      13                 324
##   WALKING_UPSTAIRS       10      17       29       7                  11
##                     Reference
## Prediction           WALKING_UPSTAIRS
##   LAYING                          12
##   SITTING                          7
##   STANDING                        12
##   WALKING                          7
##   WALKING_DOWNSTAIRS              14
##   WALKING_UPSTAIRS               385
##
## Overall Statistics
##
##                Accuracy : 0.8179
##                  95% CI : (0.8038, 0.8314)
##     No Information Rate : 0.2006
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7807
##
##  Mcnemar's Test P-Value : 5.049e-07
##
## Statistics by Class:
##
##                      Class: LAYING Class: SITTING Class: STANDING
## Sensitivity                 0.8198         0.7395          0.7658
## Specificity                 0.9589         0.9631          0.9611
## Pos Pred Value              0.8212         0.8271          0.8316
## Neg Pred Value              0.9586         0.9393          0.9424
## Prevalence                  0.1870         0.1928          0.2006
## Detection Rate              0.1533         0.1426          0.1536
## Detection Prevalence        0.1866         0.1724          0.1847
## Balanced Accuracy           0.8894         0.8513          0.8634
##                      Class: WALKING Class: WALKING_DOWNSTAIRS
## Sensitivity                  0.8973                    0.8504
## Specificity                  0.9609                    0.9649
```

```
## Pos Pred Value               0.8075                   0.7733
## Neg Pred Value               0.9808                   0.9786
## Prevalence                   0.1546                   0.1235
## Detection Rate               0.1387                   0.1050
## Detection Prevalence         0.1717                   0.1358
## Balanced Accuracy            0.9291                   0.9076
##                      Class: WALKING_UPSTAIRS
## Sensitivity                           0.8810
## Specificity                           0.9721
## Pos Pred Value                        0.8388
## Neg Pred Value                        0.9802
## Prevalence                            0.1416
## Detection Rate                        0.1248
## Detection Prevalence                  0.1487
## Balanced Accuracy                     0.9265
```

The model has a validation accuracy of 81.79%, which is good. The model also has high Precision for five of the classes i.e. LAYING (0.82), SITTING (0.83), STANDING (0.83), WALKING (0.81), and WALKING UPSTAIRS (0.84), which is impressive.

Real-World Applications of the Human Activity Recognition Models

The Human Activity Recognition (HAR) model, trained on sensor data such as accelerometers and gyroscopes, have numerous practical applications across various domains;

- In Fitness and Health Tracking to monitor daily activity and estimate energy expenditure.
- In Elderly Care to detect sudden falls or abnormal inactivity, and alert caregivers or emergency services, enabling timely intervention and improved elderly safety.
- In Rehabilitation and Remote Health Monitoring to monitor prescribed physical therapy exercises and assist physiotherapists in evaluating patients remotely.
- In Sports to track and analyze athletic movement patterns and help in preventing injuries.
- In Industrial settings to detect unsafe behaviors and help to reduce risk and improve safety compliance.

Limitations of this Analysis

- The model is unable to train with real time data, and will need to be updated when new data comes in.
- I did not tune all the model hyperparameters due to the limited computational resources, and maybe I did not obtain the optimal results. The model can still be improved upon.

- Also, some information may have been lost during PCA.