Smartwatch Activity Recognition

HarvardX - PH125.9x Capstone Project Submitted in Partial Fullfilment of the Requirements for the Data Science Professional Certificate

Phil Girurugwiro

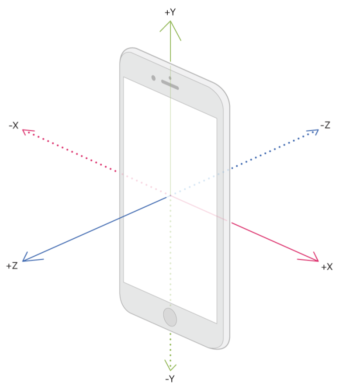
2/27/2020

Contents

Introduction

Project Overview

The objective of this project is to build a model that would accurately recognize the physical activity being performed by an individual based on the data collected by the smartwatch's accelerometer. An accelerometer is an electromechanical device built into a mobile device that measures the dynamic acceleration forces, commonly known as g-forces. Along with a gyroscope, the accelerometer allows the mobile device to track its movements. The accelerometer collects the g-forces by measuring the change in velocity in the three-dimensional space, i.e. along the x, y, and z axes and reports the values in increments of the gravitational acceleration in a given direction, x, y, or z. A value of 1.0 represents 9.81 meters per second per second. Below is a visual sketch (image from apple.com) of the accelerator.



Based on accelerometer readings, can we accurately recognize the physical activity being performed among these four activities: A= walking, B= jogging, C= climbing stairs, or P= dribbling a basketball?

Dataset

The dataset that is used in this project was taken from UCI Machine Learning Repository archive (https://archive.ics.uci.edu/ml/machine-learning-databases/00507/) and was simplified for this project. The overall dataset contains high speed (20Hz) time series data from both a smartphone and a smartwatch from 50 test subjects as they perform 18 activities for 3 minutes per activity. In addition, for each device, the gyroscope and the accelerometer were both used to collect raw data. There are therefore 4 different directories of data. For this project, a subset of this data was used:

- Collection method: Smartwatch accelerometer.
- Activity: Only 4 activities were considered for this study (A, B, C, & P)
- Logging: Instead of using the entire 20Hz data, a 10-second window average was used for this project, i.e. 200 datapoints per window were averaged to provide one datapoint. This reduced the total number of datapoints to 18 per user per activity.

The folder containing this subset is uploaded here: https://github.com/pgrugwiro/Activity_Monitor/raw/master/watch_accelerometer.zip.

Data Overview

Download and Save Data

Download the Smartwatch accelerometer dataset from the GitHub link. This is the data that will be used in building an activity recognition model.

Install packages if necessary and download data to object *dl*:

```r
dl <- tempfile()
download.file("https://github.com/pgrugwiro/Activity_Monitor/raw/master/watch_accelerometer.zip", dl)
filenames <- unzip(dl)
```

The folder contains 50 files, one file per test subject (user). Each file contains 1 second average, as well as 10 second average data for each accelerometer measurement for all 18 activities. It also contains other statistics such as correlation between coordinates.

The following script reads data from all 50 files, skipping rows that do not contain useful data, and writes the combined data into one large dataset *dataset_watch_accel*

```r
#write a function that reads csv files after skipping rows.
ez.read = function(file, ..., skip.rows=NULL){
  if (!is.null(skip.rows)) {
    tmp = readLines(file)
    tmp = tmp[-(skip.rows)]
    tmpFile = tempfile()
    on.exit(unlink(tmpFile))
    writeLines(tmp,tmpFile)
    file = tmpFile
  }
  result = read.csv(file, ...)

}

dataset_watch_accel <- do.call("rbind",lapply(filenames, FUN=function(files){
  ez.read(paste(files, sep = ","),
          skip.rows = c(1:97), sep = ",", header = FALSE)
}))
```

The following code further subsets the dataset by selecting the desired values: 10-second window averages for coordinates x, y, and z for each of the 50 users and selects activities A, B, C, & P out of the 18 measured activities.

```r
#select activity, average x, y, z, user
watch_accelerometer <- dataset_watch_accel %>% select(V1, V2, V33, V34, V93)
#name columns
columns <- c("Activity", "x", "y", "z", "user")
colnames(watch_accelerometer) <- columns

#select activities A, B, C and P only
activity_index <- which(watch_accelerometer$Activity %in% c("A", "B", "C", "P"))
watch_accelerometer <- watch_accelerometer %>% slice(activity_index)
watch_accelerometer$user <- as.factor(watch_accelerometer$user)
```

```r
#drop other factor levels
watch_accelerometer$Activity <- factor(watch_accelerometer$Activity)

#remove objects to conserve memory.
rm(dataset_watch_accel, columns, dl, filenames, ez.read, activity_index)
```

Exploratory Data Analysis

Review of data structure:

```r
str(watch_accelerometer)
```

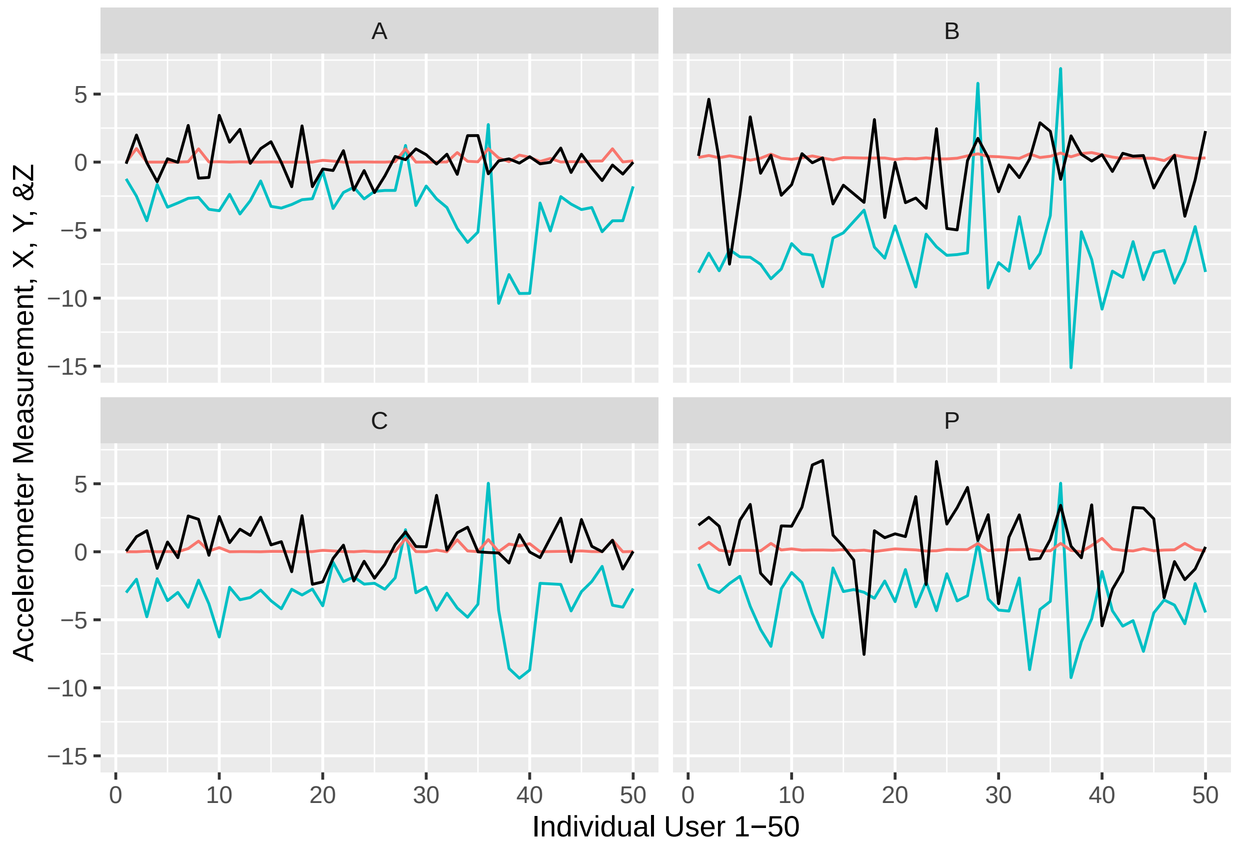
```
## 'data.frame':    4028 obs. of  5 variables:
##  $ Activity: Factor w/ 4 levels "A","B","C","P": 1 1 1 1 1 1 1 1 1 1 ...
##  $ x       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ y       : num  -1.14 -1.75 -1.54 -1.31 -1.04 ...
##  $ z       : num  -0.0229 -1.3276 -0.9722 -0.5052 0.2407 ...
##  $ user    : Factor w/ 50 levels "1600","1601",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```r
head(watch_accelerometer) %>% knitr::kable()
```

Activity	x	y	z	user
A	0	-1.13570	-0.0228586	1600
A	0	-1.75208	-1.3276100	1600
A	0	-1.53875	-0.9722430	1600
A	0	-1.31171	-0.5051590	1600
A	0	-1.03888	0.2406710	1600
A	0	-1.47259	1.0458300	1600

The dataset is indeed a dataframe with 4028 observations of 5 variables: Activity, User, X, Y, Z (XYZ are variables for accelerometer values in each of the 3 coordinates). There are 4 activities, A, B, C, & P and 50 test subjects (users).

The following plot of the predictors x, y, & z averages per users confirms individuality of users:



The following code calculates the correlation between the predictors x, y, and z for each user & activity combination. This allows to decide whether to drop predictors if they're highly correlated:

It can be seen that the correlation between the predictors is not high enough to drop them. They will all be kept in the building of the model. NA values are where the standard deviation is zero.

Due to the limited number of datapoints in our dataset, we expect high variations that may have a negative impact on the overall accuracy of the models. Here's an example of the mean and standard error of the accelerator measurements for user 1615 performing activity P.

coordinate	mean	se
x	0.1563889	0.0142504
y	-2.9231329	0.2568437
z	0.3873386	0.7732373

Building The Model

The model will be build using a train-test approach. The dataset will be split into two sets: a training set (75% of the data) and a testing set (25% of the data). Different algorithms will be tested and the accuracy of each algorithm will be calculated. Four algorithms will be used:

- Random guess of activity
- K-Nearest Neighbors algorithm
- Decision trees and
- Random Forests

Algorithms for Activity Recognition

Begin by splitting the dataset into the training set and the testing set. Each algorithm will be trained on the training set and evaluated on the test set for accuracy. The training set contains 75% of the data and the test set 25%.

```r
set.seed(1, sample.kind = "Rounding") #IF USING OLDER VERSION OF R, REMOVE "sample.kind..."
test_index <- createDataPartition(y = watch_accelerometer$x, times = 1, p = 0.25, list = FALSE)

train_set <- watch_accelerometer %>% slice(-test_index)
test_set <- watch_accelerometer %>% slice(test_index)
```

Activity Recognition by Guessing

This code generates a vector of random activities taken from the 4 activities, A, B, C & P of similar length as the test set data. It then compares the generated random list of activities to the actual list of activities and calculates the accuracy of the prediction.

```r
set.seed(2, sample.kind = "Rounding")
guess_activity <- sample(rep(c("A", "B", "C", "P"),17),length(test_set$Activity), replace = T)
guessing <- confusionMatrix(test_set$Activity, as.factor(guess_activity))


prediction_accuracy <- data_frame(Method="Guessing",
                                  Accuracy = guessing$overall[1])
```

Here is the 4x4 table of the predicted activity versus actual (confusion matrix) and the overall accuracy:

	A	B	C	P
A	64	60	67	75
B	61	66	65	55
C	53	66	63	49
P	61	63	73	67

Method	Accuracy
Guessing	0.2579365

By just guessing the physical activity being performed by the user, we only achieve the accuracy that is less than 25%. This is nothing more than just the prevalence of each activity in the dataset. We can certainly do better with advanced machine learning algorithms.

Activity Recognition with KNN Algorithm

The k-Nearest Neighbors, KNN, estimates the conditional probabilities $p(x_1, x_2)$ for any point (x_1, x_2) by averaging the k nearest points. It first calculates the distance between two predictors 1 & 2 for all pairs of predictors in the data frame: $dist(1,2) = \sqrt{\sum_{i=1}^{N}(x_{i,1} - x_{1,2})^2}$ and computes the conditional probability for a point by using the nearest points (the neighborhood). We use this algorithm found in the caret package to train the model as follows:

```r
## Fit with KNN model
fit_knn <- train(Activity~x+y+z+user, method = "knn",
                 data = train_set)

predicted_activity_knn <- predict(fit_knn, test_set)
knn <- confusionMatrix(predicted_activity_knn, test_set$Activity)

prediction_accuracy <- bind_rows(prediction_accuracy,
                                 data_frame(Method="KNN Model",
                                            Accuracy = knn$overall[1]))
```

The number of neighbors k used in the model is 5, which is the default value. Here is the 4x4 table of the predicted activity versus actual (confusion matrix) and the new overall accuracy from the knn method:

	A	B	C	P
A	211	3	61	6
B	9	229	9	14
C	33	11	135	17
P	13	4	26	227

Method	Accuracy
KNN Model	0.7956349
Guessing	0.2579365

With the kNN machine learning algorithm, we are able to achieve an accuracy that is close to 80% by using 5 neighbors. This is a good result considering the size of the data and the variations. Next, we try a different algorithm to see if it does any better, the decision trees algorithm.

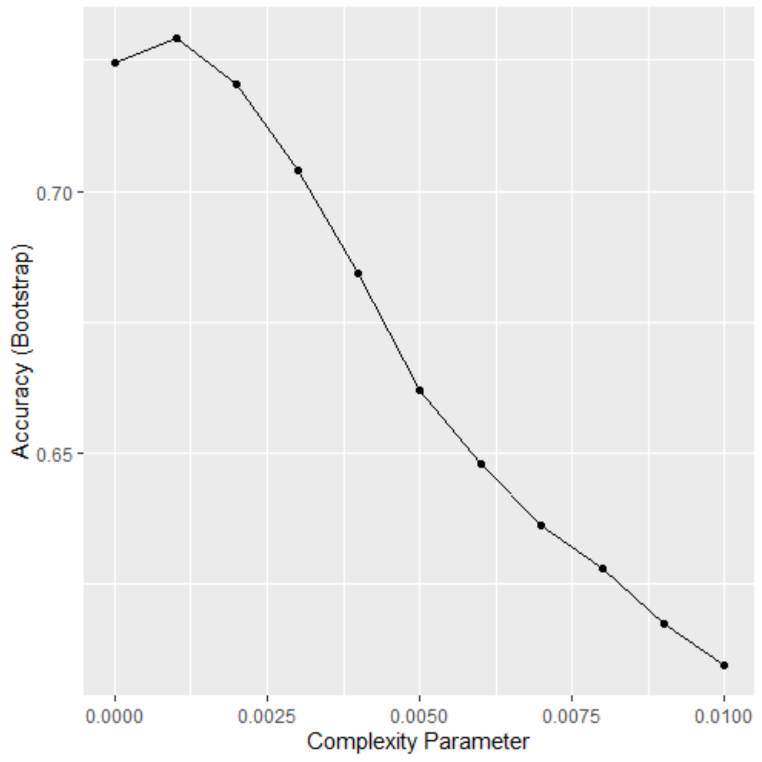
Activity Recognition with Decision Trees Algorithm (CART)

The general idea of the classification of regresssion trees (CART) is to build a decision tree and, at the end of each node, obtain a prediction $\hat{y}$. A mathematical way to describe this is to say that we are partitioning the predictor space into J non-overlapping regions, $R_1, R_2, ..., R_J$, and then for any predictor x that falls within region R_j , estimate $f(x)$ with the average of the training observations y_i for which the associated predictor x_i is also in R_j. (Data Science Book by Rafael Irizarry). This algorithm is included in the *caret* package under the name $RPART$ and is applied to our data as follows:

```r
## Fit with RPART
fit_rpart <- train(Activity ~ .,
                   method = "rpart",
                   tuneGrid = data.frame(cp = 0.001),
                   data = train_set)
predicted_activity_rpart <- predict(fit_rpart, test_set)
rpart <- confusionMatrix(predicted_activity_rpart, test_set$Activity)

prediction_accuracy <- bind_rows(prediction_accuracy,
                                 data_frame(Method="RPART",
                                            Accuracy = rpart$overall[1]))
```

The complexity parameter *cp* that was used here is 0.001, which provides the highest accuracy in the training set between the values of 0 and 0.01 (see figure below). The *cp* determines when to stop partitioning the data space. The lower the *cp* value is, the more flexible the algorithm. If *cp* is 0, then each point in the data is its own partition, which would result in overtraining.



Here is the 4x4 table of the predicted activity versus actual (confusion matrix) and the new overall accuracy from the decision trees method:

	A	B	C	P
A	147	6	49	6
B	10	224	12	21
C	84	4	141	19
P	25	13	29	218

Method	Accuracy
KNN Model	0.7956349
RPART	0.7242063
Guessing	0.2579365

The overall accuracy from the decision trees underperforms in comparison to the previous algorithm, kNN. We can improve on the decision trees algorithm by applying the random forests algorithm and see if the results are better:

Activity Recognition with Random Forests Algorithm

Random forests are a very popular machine learning approach that addresses the shortcomings of decision trees. The goal is to improve prediction performance and reduce instability by averaging multiple decision trees (a forest of trees constructed with randomness). It has two features that help accomplish this. The first step is bootstrap aggregation or bagging. The idea is to generate many predictors, each using regression or classification trees, and then forming a final prediction based on the average prediction of all these trees. To assure that the individual trees are not the same, we use the bootstrap to induce randomness. These two features combined explain the name: the bootstrap makes the individual trees randomly different, and the combination of trees is the forest. (Data Science Book by Rafael Irizarry)

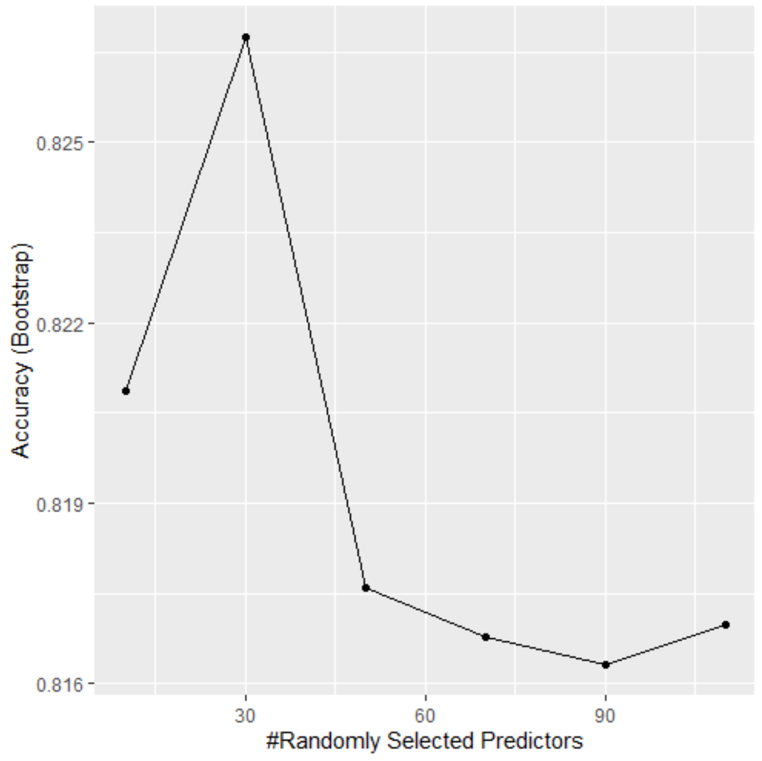
We apply the Random Forests algorithm which is included in the *randomForest* package to our training dataset.

```
fit_rforest <- train(Activity~. ,method ="rf",
                     tuneGrid = data.frame(mtry = 30),
                     data = train_set)


predicted_activity_rforest <- predict(fit_rforest, test_set)
rfor <- confusionMatrix(predicted_activity_rforest, test_set$Activity)

prediction_accuracy <- bind_rows(prediction_accuracy,
                                 data_frame(Method="RFOREST",
                                            Accuracy = rfor$overall[1]))
```

The tuning parameter *mtry* is used to pick a randomly selected subset of predictors that provide the best fit. Every tree has a different random selection of features. This reduces the correlation between features and improves prediction accuracy. In the model, the value for *mtry* is 30, which provides the highest accuracy from a range of values as shown in the figure below.



Here is the 4x4 table of the predicted activity versus actual (confusion matrix) and the new overall accuracy from the random forests method:

	A	B	C	P
A	199	0	48	1
B	4	236	4	9
C	50	5	158	6
P	13	6	21	248

Method	Accuracy
RFOREST	0.8343254
KNN Model	0.7956349
RPART	0.7242063
Guessing	0.2579365

We achieve an accuracy of 83%, the highest of all the algorithms explored. This level of accuracy is certainly acceptable for the scope of this project and considering the variation in our dataset.

Results

The results show that with our dataset, the random forests model provides the highest accuracy of 83% among all the four studied models:

- randomly guessing the activity
- predicting activity by means of the k-nearest neighbors
- predicting activity by means of decision trees
- predicting activity by means of random forests.

While it is expected that random forests would be an improvement upon decision trees, no attempt is made at understanding why it did better than the knn. This undertaking is beyond the scope of this project. Given the size of the dataset and expected variations in the data, the accuracy of 83% is satisfactory. It is worth noting that the activities studied in this project are defined as follows:

- A: Walking
- B: Jogging
- C: Climbing Stairs
- P: Dribbling a basketball. They were picked out of a list of 18 activities that included other activities such as sitting, eating soup, eating pasta, etc…

Method	Accuracy
RFOREST	0.8343254
KNN Model	0.7956349
RPART	0.7242063
Guessing	0.2579365

Conclusion

In the introduction section, layed out the goal of this project and the steps taken to arrive to the dataset that was used. The complete dataset comprised of four directories of data: phone accelerometer, phone gyroscope, smartwatch accelerometer, and smartwatch gyroscope measurements. In each of the four directories, there were 50 files, one file per one test subject. And finally, in each file, there was high speed (20Hz) data for 18 physical activities performed by the test subject, 3 minutes per activity. We set out to only use a small subset of this dataset, which limited the number of activities to just 4: Walking, Jogging, Climbing stairs, and Dribbling a ball. In addition, instead of using 20Hz data, we used 10s window averages (i.e. 200 point averages), which reduced the number of data points per activity per user from 3600 to just 18 datapoints. Finally, we only considered the smartwartch accelerometer data. With this aggressive subsetting, we were able to train the dataset with various models and achieved an accuracy of 83% when evaluating the model on the test set. We beleive we can achieve an even higher accuracy if we used 1s averages instead of 10s window averages, and an even greater accuracy if we used the entires 20Hz data, but this would require significant computational power. In addition, we can recommend exploring other algorithms beyond what was considered in this project to attempt achieving an even higher accuracy.

References

1. Introduction to Data Science - Data Analysis and Prediction Algorithms with R by Rafael A. Irizarry
2. Phone Accelerometer Sketch: https://developer.apple.com/documentation/coremotion/getting_raw_accelerometer_events
3. Opensource Dataset: https://archive.ics.uci.edu/ml/datasets/WISDM+Smartphone+and+Smartwatch+Activity+and+Biometrics+Dataset+#