CodeBook

Bryan

10/9/2021

Codebook

Signals

Note that '-XYZ' is used to denote 3-axial signals in the X, Y, and Z directions.

1. tBodyAcc-XYZ: Separated body acceleration signal via a low pass Butterworth filter with a corner frequency of 0.3 Hz
2. tGravityAcc-XYZ: Separated gravity acceleration signal via a low pass Butterworth filter with a corner frequency of 0.3 Hz
3. tBodyAccJerk-XYZ: The derived body linear acceleration with respect to time
4. tBodyGyro-XYZ: The angular velocity signal of the body
5. tBodyGyroJerk-XYZ: The derived angular velocity with respect to time
6. tBodyAccMag: The magnitude of the three-dimensional signals of tBodyAcc using the Euclidean norm
7. tGravityAccMag: The magnitude of the three-dimensional signals of tGravityAcc using the Euclidean norm
8. tBodyAccJerkMag: The magnitude of the three-dimensional signals of tBodyAccJerk using the Euclidean norm
9. tBodyGyroMag: The magnitude of the three-dimensional signals of tBodyGyro using the Euclidean norm
10. tBodyGyroJerkMag: The magnitude of the three-dimensional signals of tBodyGyroJerk using the Euclidean norm
11. fBodyAcc-XYZ: Fast Fourier Transform (FFT) applied to the tBodyAcc signals
12. fBodyAccJerk-XYZ: Fast Fourier Transform (FFT) applied to the tBodyAccJerk signals
13. fBodyGyro-XYZ: Fast Fourier Transform (FFT) applied to the tBodyGyro signals
14. fBodyAccMag: Fast Fourier Transform (FFT) applied to the tBodyAccMag signals
15. fBodyAccJerkMag: Fast Fourier Transform (FFT) applied to the tBodyAccJerkMag signals
16. fBodyGyroMag: Fast Fourier Transform (FFT) applied to the tBodyGyroMag signals
17. fBodyGyroJerkMag: Fast Fourier Transform (FFT) applied to the tBodyGyroJerkMag signals

Variables

The following are the set of variables that were estimated from these signals.

1. mean(): Mean value
2. std(): Standard deviation
3. mad(): Median absolute deviation
4. max(): Largest value in array
5. min(): Smallest value in array

6. sma(): Signal magnitude area
7. energy(): Energy measure. Sum of the squares divided by the 8. number of values.
8. iqr(): Interquartile range
9. entropy(): Signal entropy
10. arCoeff(): Autorregresion coefficients with Burg order equal to 4
11. correlation(): correlation coefficient between two signals
12. maxInds(): index of the frequency component with largest magnitude
13. meanFreq(): Weighted average of the frequency components to obtain a mean frequency
14. skewness(): skewness of the frequency domain signal
15. kurtosis(): kurtosis of the frequency domain signal
16. bandsEnergy(): Energy of a frequency interval within the 64 bins of the FFT of each window.
17. angle(): Angle between to vectors.

Data Transformation

I started by firstly loading in the dplyr package as well as the data sets to be put together and cleaned.

```
library(dplyr)

# Load in data sets
x_test <- read.table("./UCI HAR Dataset/test/X_test.txt")
y_test <- read.table("./UCI HAR Dataset/test/Y_test.txt")
x_train <- read.table("./UCI HAR Dataset/train/X_train.txt")
y_train <- read.table("./UCI HAR Dataset/train/y_train.txt")
features <- read.table("./UCI HAR Dataset/features.txt")
labels <- read.table("./UCI HAR Dataset/activity_labels.txt")
subjects_train <- read.table("./UCI HAR Dataset/train/subject_train.txt")
subjects_test <- read.table("./UCI HAR Dataset/test/subject_test.txt")
```

Once all the data sets were loaded in, I labelled the column names properly and merged the training and test sets

```
# Place in labels for each x data set
headers <- features[,2]
colnames(x_test) <- headers
colnames(x_train) <- headers

# Label columns in y data sets
y_label <- c("Activity")
colnames(y_test) <- y_label
colnames(y_train) <- y_label

# Merge training data set with test data set and create final merged data set
x_tt <- rbind(x_train, x_test)
y_tt <- rbind(y_train, y_test)
merged_data <- cbind(x_tt, y_tt)
```

To extract the mean and standard deviation of each measurement, I made a function and called it once to get the mean and another time to get the standard deviation.

```
# Create function to extract standard deviation and mean
extractData <- function(data) {
```

```
        reg_ex <- paste(data, "\\(\\)", sep="")
        filter <- grep(reg_ex, colnames(merged_data))
        new_data <- merged_data[, filter]
}

# Extract mean and standard deviations of each measurement
mean_measures <- extractData("mean")
std_measures <- extractData("std")
```

To descriptively label the activities, I used the numbers of the "Activity" column as indices to a vector that contained the descriptive activity names.

```
# Create function to assign activity labels to numbers
assignCode <- function() {
        acts <- labels$V2
        code <- lapply(merged_data$Activity, function(i){acts[i]})
}

# Replace numbers with actual activity labels
merged_data$Activity <- assignCode()
```

In order to group by both activity and subject, we first need to add the subjects to the data set since it isn't initially a part of it.

```
# Create function to merge training subjects and test subjects
assignSubjects <- function() {
        subjects <- rbind(subjects_train, subjects_test)
        colnames(subjects) <- c("Subjects")
        subjects
}

# Add subjects to data set
merged_data$Subjects <- assignSubjects()
```

The data set we now have is currently incapable of being grouped because there are duplicate column names. Upon further inspection, I noticed that each column that had a duplicate always had two duplicates, meaning that columns of the same name occurred 3 times. These were supposed to be measurements for the X, Y, and Z components, respectively. So, to handle these duplicates, I scanned through the data set for them and added an "X" to the first instance, a "Y" to the second instance, and a "Z" to the third instance.

```
manageDuplicates <- function() {

        # Store column names for data set
        data_columns <- colnames(merged_data)

        # Create X, Y, and Z characters to assign to duplicate column names
        dims <- c("X", "Y", "Z")

        # Create index to iterate through dims vector
        index <- 0

        # Locate names of duplicated columns
```

```
        duplicate_columns <- merged_data[,duplicated(data_columns)]

        # Go through each column name that's duplicated
        for (elem in colnames(duplicate_columns)) {

                # Find indices of duplicated column name in data_columns
                instances <- which(data_columns %in% elem)
                for (e in instances) {

                        # Assign an X, Y, or Z character to the end of said
                        # column name
                        index <- (index %% 3) + 1
                        data_columns[e] <- paste(data_columns[e], dims[index]
                                                , sep=" ")
                }
        }

        colnames(merged_data) <<- data_columns
}

# Manage duplicate columns before grouping them by activity and subject
manageDuplicates()
```

With the duplicates managed, we can now sort by group and activity and create a new data set containing the mean of all variables for each activity and subject.

```
# Manage duplicate columns before grouping them by activity and subject
manageDuplicates()

createSummary <- function() {
        grp <- merged_data %>% group_by(Activity, Subjects)
        summary <- summarise_all(grp, mean)
        summary
}

# Create second data set with average of each variable by activity and subject
s <- createSummary()
```