Week 4 project - Getting and Cleaning Data

Camilla Jensen

October 21, 2016

This codebook explains step by step how the data was created and binds together all the files and scripts that were used to complete the project assignment.

## It is therefore in principle the combination of all the other submitted files and the readme file. The list of variables, their explanations and measurement is reported at the end of the codebook. The scripts are embedded in the codebook as well. In some sense the readme file becomes superfluous. Hence my readme file is really short and mainly serves to give proper reference to the data source.

1.SCRIPT WITH EXPLANATIONS

First we need to download and unzip the dataset (must be connected to Internet for this step to work):

download.file("https://d396qusza40orc.cloudfront.net/getdata%2Fprojectfiles%2FUCI%20HAR%20Dataset.zip", destfile="HARUSD.zip")

Close all connections and unzip the folder:

unzip(zipfile="HARUSD.zip")

Now we can see the same dataset in the local environment (my documents folder) as we can see online or by just downloading it to the PC and it got into the right archive or folder on its own and it looks exactly the way it should and is in the folder named "UCI HAR Dataset".

From this 'raw' dataset we have to read in any relevant part to solve the assignment.

The README file contains a lot of information about this. We learn that test and train are subsets of the full subject sample, e.g. 30/70 division of trial subjects into these two groups. This means that we need to stack the two subsets into one file rather than merging them via any common id per se. In other words the subjectids are not overlapping in this case so some of the 30 subjects are tried out in the train part of the dataset and the others in the test part. So for this procedure we have to use the rbind command (e.g. to stack the data).

First we need to read in all the files as data tables. X\_train and X\_test are really the essential files as they contain all the data, while y\_train and y\_test hold the key to the activity codes of which there are 6 and link these codes to the actual descriptors. Similarly, subject\_train and subject\_test contain the keys to the subject ids of which there are 30.

First I read in all the labelling files using the following commands:

headers <- read.table ("UCI HAR Dataset/features.txt")  
activity <- read.table ("UCI HAR Dataset/activity\_labels.txt")  
subjectid\_train <- read.table("UCI HAR Dataset/train/subject\_train.txt")  
subjectid\_test <- read.table("UCI HAR Dataset/test/subject\_test.txt")

Now I read in the actual data files:

X\_train <- read.table("UCI HAR Dataset/train/X\_train.txt")  
X\_test <- read.table("UCI HAR Dataset/test/X\_test.txt")

And finally the activity id's that connect the dots:

y\_train <- read.table("UCI HAR Dataset/train/y\_train.txt")  
y\_test <- read.table("UCI HAR Dataset/test/y\_test.txt")

We have read in a totality of 8 files and it can also be verified under Data in the Global Environment. Here we now see the following data read into r-studio:

activity, 6 obs of 2 vars

headers, 561 obs of 2 vars

subjectid\_test, 2947 obs of 1 var

subjectid\_train, 7352 obs of 1 var

X\_test, 2947 obs of 561 vars

X\_train, 7352 obs of 561 vars

y\_test, 2947 obs of 1 var

y\_train, 7352 obs of 1 var

These datafiles must be merged to get to the full dataset. We start by assigning the 561 obs in header onto the X\_test and X\_train datasets as these 561 obs are the key to understanding what lies under the 561 vars in actual datasets:

colnames(X\_train) <- headers[ ,2]  
colnames(X\_test) <- headers[ ,2]

Then we have to assign the header activitycode to the 2947 and 7352 obs in the two datasets. This is the 1 var in the y\_test and y\_train datatset and we have to assign the header subjectid to the 1 var in the subject\_id test and the subject\_id train datatables:

colnames(y\_train) <- "activitycode"  
colnames(y\_test) <- "activitycode"  
colnames(subjectid\_test) <- "subjectid"  
colnames(subjectid\_train) <- "subjectid"

Finally we need to assign the same header to the activitylabels file and thereby get access to the descriptor for the 6 activity groups (but it will only be attached to the full dataset as the last step before delivering the data because it is string and will disappear in the later aggregations):

colnames(activity) <- c("activitycode", "activitytype")

Now we can first merge all the info for the train and test groups respectively, and then finally combine them into a full datast using rbind:

train <- cbind(y\_train, subjectid\_train, X\_train)  
test <- cbind(y\_test, subjectid\_test, X\_test)  
fullset <- rbind(train, test)

Looking at the Data files on the right hand side in R studio we can see that train and test now have 563 vars instead of the original 561 vars as we have added the activitycodes and subjectids to the actual data. Finally the fullset combines the two subsamples into one and has the correct sum of observations e.g. 2,947+7,352=10,299 obs. Checking with head and tail commands everything looks normal in the dataset and we can see the activitycodes and subjectids followed by all the other variables.

The next step is to select only a portion of the fullset, namely all variables that contain variable means m and standard deviations sd (and assumedly not include the meanFreq as it is a different kind of stats var). We also want to keep the activitycodes and subjectids. We can use the grepl function which can search for patterns in the text, hence we want to extract anything from the fullset where the following is true and afterwards apply this selectdata function to retrieve the relevant portion of the fullset to get the fullset\_ms (selective part only containing mean and sd values):

COLS <- colnames(fullset)  
selectdata <- (grepl("activitycode", COLS) | grepl("subjectid", COLS) | grepl("mean()..", COLS) & !grepl("-meanFreq..", COLS) | grepl("std()...", COLS))  
fullset\_ms <-fullset[ ,selectdata==TRUE]

Finally we need to make the summary dataset from the fullset\_ms file where this data is now summarised in a more condensed form by subjectid and activitycodes.

We also have to make sure the data is properly labelled with the description of the activitycodes using the string variable. Here the problem might be that summarizing the data returns numeric values for the string labels. Hence we wait to add the labels to the very last step before writing the final dataset as a textfile:

finaldata <- aggregate(. ~subjectid + activitycode, fullset\_ms, mean)  
finaldata <- merge (activity, finaldata, by="activitycode")  
write.table(finaldata, "finaldata.txt", row.names=FALSE)

The final dataset has 180 observations (e.g. 6 activities x 30 subjects = 180 obs) and there is now a totality of only 60 variables (as described next with the metadata file).

1. METADATA

First we can create a list of the vars for the metadata to copy over to the text:

str(finaldata)

## 'data.frame': 180 obs. of 60 variables:  
## $ activitycode : int 1 1 1 1 1 1 1 1 1 1 ...  
## $ activitytype : Factor w/ 6 levels "LAYING","SITTING",..: 4 4 4 4 4 4 4 4 4 4 ...  
## $ subjectid : int 1 2 3 4 5 6 7 8 9 10 ...  
## $ tBodyAcc-mean()-X : num 0.277 0.276 0.276 0.279 0.278 ...  
## $ tBodyAcc-mean()-Y : num -0.0174 -0.0186 -0.0172 -0.0148 -0.0173 ...  
## $ tBodyAcc-mean()-Z : num -0.111 -0.106 -0.113 -0.111 -0.108 ...  
## $ tBodyAcc-std()-X : num -0.284 -0.424 -0.36 -0.441 -0.294 ...  
## $ tBodyAcc-std()-Y : num 0.1145 -0.0781 -0.0699 -0.0788 0.0767 ...  
## $ tBodyAcc-std()-Z : num -0.26 -0.425 -0.387 -0.586 -0.457 ...  
## $ tGravityAcc-mean()-X : num 0.935 0.913 0.937 0.964 0.973 ...  
## $ tGravityAcc-mean()-Y : num -0.2822 -0.3466 -0.262 -0.0859 -0.1004 ...  
## $ tGravityAcc-mean()-Z : num -0.0681 0.08473 -0.13811 0.12776 0.00248 ...  
## $ tGravityAcc-std()-X : num -0.977 -0.973 -0.978 -0.984 -0.979 ...  
## $ tGravityAcc-std()-Y : num -0.971 -0.972 -0.962 -0.968 -0.962 ...  
## $ tGravityAcc-std()-Z : num -0.948 -0.972 -0.952 -0.963 -0.965 ...  
## $ tBodyAccJerk-mean()-X : num 0.074 0.0618 0.0815 0.0784 0.0846 ...  
## $ tBodyAccJerk-mean()-Y : num 0.02827 0.01825 0.01006 0.00296 -0.01632 ...  
## $ tBodyAccJerk-mean()-Z : num -4.17e-03 7.90e-03 -5.62e-03 -7.68e-04 8.32e-05 ...  
## $ tBodyAccJerk-std()-X : num -0.114 -0.278 -0.269 -0.297 -0.303 ...  
## $ tBodyAccJerk-std()-Y : num 0.067 -0.0166 -0.045 -0.2212 -0.091 ...  
## $ tBodyAccJerk-std()-Z : num -0.503 -0.586 -0.529 -0.751 -0.613 ...  
## $ tBodyGyro-mean()-X : num -0.0418 -0.053 -0.0256 -0.0318 -0.0489 ...  
## $ tBodyGyro-mean()-Y : num -0.0695 -0.0482 -0.0779 -0.0727 -0.069 ...  
## $ tBodyGyro-mean()-Z : num 0.0849 0.0828 0.0813 0.0806 0.0815 ...  
## $ tBodyGyro-std()-X : num -0.474 -0.562 -0.572 -0.501 -0.491 ...  
## $ tBodyGyro-std()-Y : num -0.0546 -0.5385 -0.5638 -0.6654 -0.5046 ...  
## $ tBodyGyro-std()-Z : num -0.344 -0.481 -0.477 -0.663 -0.319 ...  
## $ tBodyGyroJerk-mean()-X : num -0.09 -0.0819 -0.0952 -0.1153 -0.0888 ...  
## $ tBodyGyroJerk-mean()-Y : num -0.0398 -0.0538 -0.0388 -0.0393 -0.045 ...  
## $ tBodyGyroJerk-mean()-Z : num -0.0461 -0.0515 -0.0504 -0.0551 -0.0483 ...  
## $ tBodyGyroJerk-std()-X : num -0.207 -0.39 -0.386 -0.492 -0.358 ...  
## $ tBodyGyroJerk-std()-Y : num -0.304 -0.634 -0.639 -0.807 -0.571 ...  
## $ tBodyGyroJerk-std()-Z : num -0.404 -0.435 -0.537 -0.64 -0.158 ...  
## $ tBodyAccMag-mean() : num -0.137 -0.29 -0.255 -0.312 -0.158 ...  
## $ tGravityAccMag-mean() : num -0.137 -0.29 -0.255 -0.312 -0.158 ...  
## $ tBodyAccJerkMag-mean() : num -0.141 -0.281 -0.28 -0.367 -0.288 ...  
## $ tBodyGyroMag-mean() : num -0.161 -0.447 -0.466 -0.498 -0.356 ...  
## $ tBodyGyroJerkMag-mean() : num -0.299 -0.548 -0.566 -0.681 -0.445 ...  
## $ fBodyAcc-mean()-X : num -0.203 -0.346 -0.317 -0.427 -0.288 ...  
## $ fBodyAcc-mean()-Y : num 0.08971 -0.0219 -0.0813 -0.1494 0.00946 ...  
## $ fBodyAcc-mean()-Z : num -0.332 -0.454 -0.412 -0.631 -0.49 ...  
## $ fBodyAcc-std()-X : num -0.319 -0.458 -0.379 -0.447 -0.298 ...  
## $ fBodyAcc-std()-Y : num 0.056 -0.1692 -0.124 -0.1018 0.0426 ...  
## $ fBodyAcc-std()-Z : num -0.28 -0.455 -0.423 -0.594 -0.483 ...  
## $ fBodyAccJerk-mean()-X : num -0.171 -0.305 -0.305 -0.359 -0.345 ...  
## $ fBodyAccJerk-mean()-Y : num -0.0352 -0.0788 -0.1405 -0.2796 -0.1811 ...  
## $ fBodyAccJerk-mean()-Z : num -0.469 -0.555 -0.514 -0.729 -0.59 ...  
## $ fBodyAccJerk-std()-X : num -0.134 -0.314 -0.297 -0.297 -0.321 ...  
## $ fBodyAccJerk-std()-Y : num 0.10674 -0.01533 -0.00561 -0.2099 -0.05452 ...  
## $ fBodyAccJerk-std()-Z : num -0.535 -0.616 -0.544 -0.772 -0.633 ...  
## $ fBodyGyro-mean()-X : num -0.339 -0.43 -0.438 -0.373 -0.373 ...  
## $ fBodyGyro-mean()-Y : num -0.103 -0.555 -0.562 -0.688 -0.514 ...  
## $ fBodyGyro-mean()-Z : num -0.256 -0.397 -0.418 -0.601 -0.213 ...  
## $ fBodyGyro-std()-X : num -0.517 -0.604 -0.615 -0.543 -0.529 ...  
## $ fBodyGyro-std()-Y : num -0.0335 -0.533 -0.5689 -0.6547 -0.5027 ...  
## $ fBodyGyro-std()-Z : num -0.437 -0.56 -0.546 -0.716 -0.42 ...  
## $ fBodyAccMag-mean() : num -0.129 -0.324 -0.29 -0.451 -0.305 ...  
## $ fBodyBodyAccJerkMag-mean() : num -0.0571 -0.1691 -0.1868 -0.3186 -0.2695 ...  
## $ fBodyBodyGyroMag-mean() : num -0.199 -0.531 -0.57 -0.609 -0.484 ...  
## $ fBodyBodyGyroJerkMag-mean(): num -0.319 -0.583 -0.608 -0.724 -0.548 ...

The trimmed output is copied here below (for exact definition of the vars with respect to more technical features of the dataset see also the Feature selection file from the authors which is copied in towards the end of the file). It is not entirely clear from the documentation given what is the unit of measurement neither of time nor frequency variables, so I guess for time it is seconds, but for frequency it could be either 50, 20 or 0.3 hz as I have no full insight into how those filters work:

Variable: Class and units:

activitycode Integer w/ 6 levels 1 through 6

activitytype Factor w/ 6 levels "LAYING","SITTING"etc.

subjectid Integer w/ 30 subjects 1 through 30

tBodyAcc-mean()-X Numeric, sec (time)

tBodyAcc-mean()-Y Numeric, sec

tBodyAcc-mean()-Z Numeric, sec

tBodyAcc-std()-X Numeric, sec

tBodyAcc-std()-Y Numeric, sec

tBodyAcc-std()-Z Numeric, sec

tGravityAcc-mean()-X Numeric, sec

tGravityAcc-mean()-Y Numeric, sec

tGravityAcc-mean()-Z Numeric, sec

tGravityAcc-std()-X Numeric, sec

tGravityAcc-std()-Y Numeric, sec

tGravityAcc-std()-Z Numeric, sec

tBodyAccJerk-mean()-X Numeric, sec

tBodyAccJerk-mean()-Y Numeric, sec

tBodyAccJerk-mean()-Z Numeric, sec

tBodyAccJerk-std()-X Numeric, sec

tBodyAccJerk-std()-Y Numeric, sec

tBodyAccJerk-std()-Z Numeric, sec

tBodyGyro-mean()-X Numeric, sec

tBodyGyro-mean()-Y Numeric, sec

tBodyGyro-mean()-Z Numeric, sec

tBodyGyro-std()-X Numeric, sec

tBodyGyro-std()-Y Numeric, sec

tBodyGyro-std()-Z Numeric, sec

tBodyGyroJerk-mean()-X Numeric, sec

tBodyGyroJerk-mean()-Y Numeric, sec

tBodyGyroJerk-mean()-Z Numeric, sec

tBodyGyroJerk-std()-X Numeric, sec

tBodyGyroJerk-std()-Y Numeric, sec

tBodyGyroJerk-std()-Z Numeric, sec

tBodyAccMag-mean() Numeric, sec

tGravityAccMag-mean() Numeric, sec

tBodyAccJerkMag-mean() Numeric, sec

tBodyGyroMag-mean() Numeric, sec

tBodyGyroJerkMag-mean() Numeric, sec

fBodyAcc-mean()-X Numeric, hz (frequency, either 50, 20 or 0.3??) fBodyAcc-mean()-Y Numeric, hz

fBodyAcc-mean()-Z Numeric, hz

fBodyAcc-std()-X Numeric, hz

fBodyAcc-std()-Y Numeric, hz

fBodyAcc-std()-Z Numeric, hz

fBodyAccJerk-mean()-X Numeric, hz

fBodyAccJerk-mean()-Y Numeric, hz

fBodyAccJerk-mean()-Z Numeric, hz

fBodyAccJerk-std()-X Numeric, hz

fBodyAccJerk-std()-Y Numeric, hz

fBodyAccJerk-std()-Z Numeric, hz

fBodyGyro-mean()-X Numeric, hz

fBodyGyro-mean()-Y Numeric, hz

fBodyGyro-mean()-Z Numeric, hz

fBodyGyro-std()-X Numeric, hz

fBodyGyro-std()-Y Numeric, hz

fBodyGyro-std()-Z Numeric, hz

fBodyAccMag-mean() Numeric, hz

fBodyBodyAccJerkMag-mean() Numeric, hz

fBodyBodyGyroMag-mean() Numeric, hz

fBodyBodyGyroJerkMag-mean() Numeric, hz

# Feature Selection

The features selected for this database come from the accelerometer and gyroscope 3-axial raw signals tAcc-XYZ and tGyro-XYZ. These time domain signals (prefix 't' to denote time) were captured at a constant rate of 50 Hz. Then they were filtered using a median filter and a 3rd order low pass Butterworth filter with a corner frequency of 20 Hz to remove noise. Similarly, the acceleration signal was then separated into body and gravity acceleration signals (tBodyAcc-XYZ and tGravityAcc-XYZ) using another low pass Butterworth filter with a corner frequency of 0.3 Hz.

Subsequently, the body linear acceleration and angular velocity were derived in time to obtain Jerk signals (tBodyAccJerk-XYZ and tBodyGyroJerk-XYZ). Also the magnitude of these three-dimensional signals were calculated using the Euclidean norm (tBodyAccMag, tGravityAccMag, tBodyAccJerkMag, tBodyGyroMag, tBodyGyroJerkMag).

Finally a Fast Fourier Transform (FFT) was applied to some of these signals producing fBodyAcc-XYZ, fBodyAccJerk-XYZ, fBodyGyro-XYZ, fBodyAccJerkMag, fBodyGyroMag, fBodyGyroJerkMag. (Note the 'f' to indicate frequency domain signals).

These signals were used to estimate variables of the feature vector for each pattern:  
'-XYZ' is used to denote 3-axial signals in the X, Y and Z directions.

tBodyAcc-XYZ tGravityAcc-XYZ tBodyAccJerk-XYZ tBodyGyro-XYZ tBodyGyroJerk-XYZ tBodyAccMag tGravityAccMag tBodyAccJerkMag tBodyGyroMag tBodyGyroJerkMag fBodyAcc-XYZ fBodyAccJerk-XYZ fBodyGyro-XYZ fBodyAccMag fBodyAccJerkMag fBodyGyroMag fBodyGyroJerkMag

The set of variables that were estimated from these signals are:

mean(): Mean value std(): Standard deviation mad(): Median absolute deviation max(): Largest value in array min(): Smallest value in array sma(): Signal magnitude area energy(): Energy measure. Sum of the squares divided by the number of values. iqr(): Interquartile range entropy(): Signal entropy arCoeff(): Autorregresion coefficients with Burg order equal to 4 correlation(): correlation coefficient between two signals maxInds(): index of the frequency component with largest magnitude meanFreq(): Weighted average of the frequency components to obtain a mean frequency skewness(): skewness of the frequency domain signal kurtosis(): kurtosis of the frequency domain signal bandsEnergy(): Energy of a frequency interval within the 64 bins of the FFT of each window. angle(): Angle between to vectors.

Additional vectors obtained by averaging the signals in a signal window sample. These are used on the angle() variable:

gravityMean tBodyAccMean tBodyAccJerkMean tBodyGyroMean tBodyGyroJerkMean

The complete list of variables of each feature vector is available in 'features.txt'