Getting and Cleaning Data- Course Project Assignment write up

Summary

This document summarizes the approach followed in preparing the data obtained from Human Activity Recognition activity measurements through sensors. The data from training and test sets are merged with their corresponding subject and activity sources. Descriptive names are then added to each of the parameters measured. Data is then filtered for only mean or standard deviation parameters and then summarized by each subject and for each activity.

Objective

Objective of the exercise is to combine all the data available in the source and making it tidy data set and summarizing for each subject and for each activity.

Source

The data set comes from the Human Activity Recognition Using Smartphones Dataset by Jorge L. Reyes-Ortiz, Davide Anguita, Alessandro Ghio, Luca Oneto. Smartlab - Non Linear Complex Systems Laboratory DITEN - Universit? degli Studi di Genova. Via Opera Pia 11A, I-16145, Genoa, Italy. activityrecognition@smartlab.ws www.smartlab.ws

Details

The experiments have been carried out with a group of 30 volunteers within an age bracket of 19-48 years. Each person performed six activities (WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING, LAYING) wearing a smartphone (Samsung Galaxy S II) on the waist. Using sensor signals (accelerometer and gyroscope) data were collected and filtered thru time window and a vector of features was obtained by calculating variables from the time and frequency domain. Out of the 561 features available, only mean and standard deviations are collected and archived.

Input

First data set is unzipped on local working directory that results in the following folder structure for the data:

.\readme.txt

.\activity_labels.txt

.\features.txt

.\train\X_train.txt

.\train\y_train.txt

.\train\subject_train.txt

.\test\X_test.txt

.\test\y_test.txt

.\test\subject_test.txt

Approach

1. First all the input data sets are read in R as described in code_analysis.R
2. Observe X training sets and X test sets to make sure they have equal number of columns and no missing values (column comparison)
3. Observe the number of readings (total number of data) for X and y and subjects match) match for training and test sets separately.
4. Luckily this is already a good tidy dataset by all means!
5. Training and test data sets are merged for each of the above measurements. Rowbind command in R used to append the test sets at the end of training sets for each paramters.

   This results in X_traintest, y_traintest, subject_traintest.
6. Let us make it little more easy to read the columns
7. Feature names are read from features.txt that results in 561 parameter names corresponding 561 columns available in X_traintest
8. We will add column headers for each of the data sets for easy reading and filtering.
9. For X_traintest, we assign parameter headings from second column of features dataframe.
10. Similarly single numeric vector y_traintest is assigned the column header "activity". subject_traintest is assigned column header "subject". Activity_labels dataset is assigned column header c("activity","activity_label").
11. Now we will filter only those columns from X_traintest containing mean() or words in their column header. We will use grepl command for this purpose.

    colPattern1 ← "mean() | std()"

```
X_final <- X_traintest[,grepl(colPattern1,colnames(X_traintest))][,]
```

This results in 79 columns after filtering.

12. Now we will merge this data with subject_traintest and y_traintest by using column binding in R

```
Xy_final <- cbind(subject_traintest,y_traintest,X_final)
```

13. Now we will add activity labels for each observation. This is done thru merging activity_labels with X_final dataset we got above. We use merge command in R for this

```
Xy_finalact <- merge(activity_labels,Xy_final,by.x="activity",by.y="activity")
```

14. At this time data is tidy with easily readable column headers.

15. Next task is to summarize the data by taking mean of each parameters for each subject and for each activity. Groupwise summarizing is done thru use of package plyr in R.

```
library(plyr)

Xy_tidySummary <-
ddply(Xy_finalact,c("subject","activity","activity_label"),function(df)
colMeans(df[,4:82]))
```

16. This results in 180 rows with 82 columns of parameters.

17. We will write to this to a text file on the local hard disk by using write.table command in R

```
write.table(Xy_tidySummary,"har_meanstdsum.txt",row.name=FALSE)
```

18. We may want to clean all the memory by removing the R objects from memory (we have indeed lot of data and made several copies along the way) if we want to move on to other projects.

```
##rm(list=ls(all=TRUE))
```

Usage

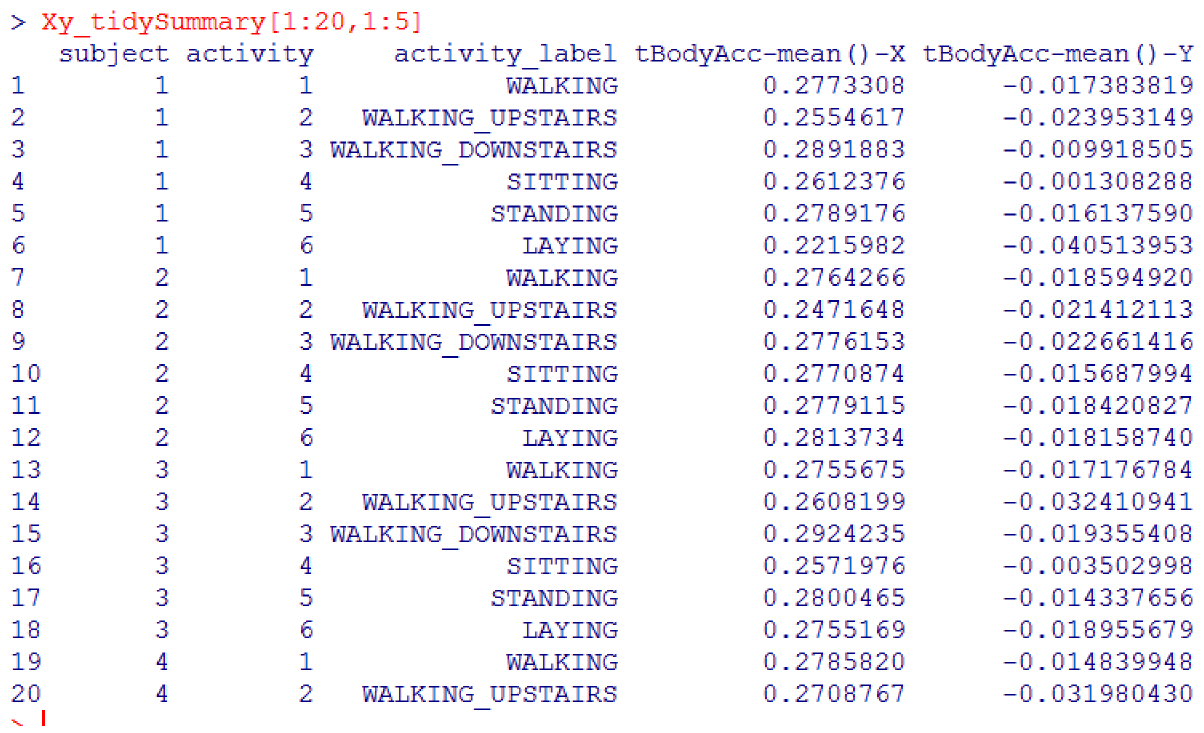
All the above code is contained in the R source file run_analysis.R and can be run in R studio:

```
source("run_analysis.R")
```

Output

A data frame with 180 observations on the following 82 variables.

A sample of the output is shown below:



```
> Xy_tidySummary[1:20,1:5]
   subject activity     activity_label tBodyAcc-mean()-X tBodyAcc-mean()-Y
1        1        1            WALKING         0.2773308      -0.017383819
2        1        2   WALKING_UPSTAIRS         0.2554617      -0.023953149
3        1        3 WALKING_DOWNSTAIRS         0.2891883      -0.009918505
4        1        4            SITTING         0.2612376      -0.001308288
5        1        5           STANDING         0.2789176      -0.016137590
6        1        6             LAYING         0.2215982      -0.040513953
7        2        1            WALKING         0.2764266      -0.018594920
8        2        2   WALKING_UPSTAIRS         0.2471648      -0.021412113
9        2        3 WALKING_DOWNSTAIRS         0.2776153      -0.022661416
10       2        4            SITTING         0.2770874      -0.015687994
11       2        5           STANDING         0.2779115      -0.018420827
12       2        6             LAYING         0.2813734      -0.018158740
13       3        1            WALKING         0.2755675      -0.017176784
14       3        2   WALKING_UPSTAIRS         0.2608199      -0.032410941
15       3        3 WALKING_DOWNSTAIRS         0.2924235      -0.019355408
16       3        4            SITTING         0.2571976      -0.003502998
17       3        5           STANDING         0.2800465      -0.014337656
18       3        6             LAYING         0.2755169      -0.018955679
19       4        1            WALKING         0.2785820      -0.014839948
20       4        2   WALKING_UPSTAIRS         0.2708767      -0.031980430
```

Description of each columns

`subject`

a numeric vector ranging from 1 to 30 representing the subject/volunteer identifier

`activity`

a numeric vector ranging from 1 to 6 representing the activity of the subject measured

`activity_label`

a character vector describing the activity measured above.

Following is the mapping between these two columns

activity	**activity_label**
1	**WALKING**
2	**WALKING_UPSTAIRS**

3	**WALKING_DOWNSTAIRS**
4	**SITTING**
5	**STANDING**
6	**LAYING**

```
tBodyAcc-mean()-X
..
fBodyBodyGyroJerkMag-meanFreq()
```

Columns 4 thru 82 a numeric vector representing the mean of the each of separate parameters measured for each subject and for each activity. They are categorized into various Body Acceleration or Body Jerks etc. Here is the full list:

```
4. tBodyAcc-mean()-X
5. tBodyAcc-mean()-Y
6. tBodyAcc-mean()-Z
7. tBodyAcc-std()-X
8. tBodyAcc-std()-Y
9. tBodyAcc-std()-Z
10. tGravityAcc-mean()-X
11. tGravityAcc-mean()-Y
12. tGravityAcc-mean()-Z
13. tGravityAcc-std()-X
14. tGravityAcc-std()-Y
15. tGravityAcc-std()-Z
16. tBodyAccJerk-mean()-X
17. tBodyAccJerk-mean()-Y
18. tBodyAccJerk-mean()-Z
19. tBodyAccJerk-std()-X
20. tBodyAccJerk-std()-Y
21. tBodyAccJerk-std()-Z
22. tBodyGyro-mean()-X
23. tBodyGyro-mean()-Y
24. tBodyGyro-mean()-Z
25. tBodyGyro-std()-X
26. tBodyGyro-std()-Y
27. tBodyGyro-std()-Z
28. tBodyGyroJerk-mean()-X
29. tBodyGyroJerk-mean()-Y
30. tBodyGyroJerk-mean()-Z
31. tBodyGyroJerk-std()-X
32. tBodyGyroJerk-std()-Y
33. tBodyGyroJerk-std()-Z
34. tBodyAccMag-mean()
35. tBodyAccMag-std()
36. tGravityAccMag-mean()
37. tGravityAccMag-std()
38. tBodyAccJerkMag-mean()
39. tBodyAccJerkMag-std()
```

```
40. tBodyGyroMag-mean()
41. tBodyGyroMag-std()
42. tBodyGyroJerkMag-mean()
43. tBodyGyroJerkMag-std()
44. fBodyAcc-mean()-X
45. fBodyAcc-mean()-Y
46. fBodyAcc-mean()-Z
47. fBodyAcc-std()-X
48. fBodyAcc-std()-Y
49. fBodyAcc-std()-Z
50. fBodyAcc-meanFreq()-X
51. fBodyAcc-meanFreq()-Y
52. fBodyAcc-meanFreq()-Z
53. fBodyAccJerk-mean()-X
54. fBodyAccJerk-mean()-Y
55. fBodyAccJerk-mean()-Z
56. fBodyAccJerk-std()-X
57. fBodyAccJerk-std()-Y
58. fBodyAccJerk-std()-Z
59. fBodyAccJerk-meanFreq()-X
60. fBodyAccJerk-meanFreq()-Y
61. fBodyAccJerk-meanFreq()-Z
62. fBodyGyro-mean()-X
63. fBodyGyro-mean()-Y
64. fBodyGyro-mean()-Z
65. fBodyGyro-std()-X
66. fBodyGyro-std()-Y
67. fBodyGyro-std()-Z
68. fBodyGyro-meanFreq()-X
69. fBodyGyro-meanFreq()-Y
70. fBodyGyro-meanFreq()-Z
71. fBodyAccMag-mean()
72. fBodyAccMag-std()
73. fBodyAccMag-meanFreq()
74. fBodyBodyAccJerkMag-mean()
75. fBodyBodyAccJerkMag-std()
76. fBodyBodyAccJerkMag-meanFreq()
77. fBodyBodyGyroMag-mean()
78. fBodyBodyGyroMag-std()
79. fBodyBodyGyroMag-meanFreq()
80. fBodyBodyGyroJerkMag-mean()
81. fBodyBodyGyroJerkMag-std()
82. fBodyBodyGyroJerkMag-meanFreq()
```

Conclusion

R is a very powerful language for large data preparation and analysis. Its commands are very brief but very powerful. Lots of great packages are freely available in the cran.r-project.org websites that make the R language a very promising skill to master.