Predicting Location via Indoor Positioning Systems

Mike Crowder, Brian Kolovich, Brandon Lawerance, Geardo Garza

6/2/2018

Abstract

Fill in after writing the paper

Introduction

This case was explored in detail in the book by Deborah Nolan and Duncan Temple Lang called "Data Science in R: A Case Studies Approach to Computational Reasoning and Problem Solving". Indoor Positioning Systems (IPS) are often used because what we know as Global Positioning Systems (GPS) have a hard time working within buildings. Given the growth of wireless local area networks (LANs),IPS have the ability to use WiFi signals detected from network access points. Often questions about where is an object whether it be another person, yourself an object have the ability to be answered in real time.

Background

Our dataset is from a the Community Resource for Archiving Wireless Data At Dartmouth (CRAWDAD). The "offline" is a referenced data set that houses signal strengths measured with a hand-held device on a grid of 166 points spaced 1 meter apart located in the hallways of a building at the University of Mannheim in Germany.

The floor plan measures 15 meters by 36 meters (49 feet by 118 feet). A floor plan is given in figure 1.

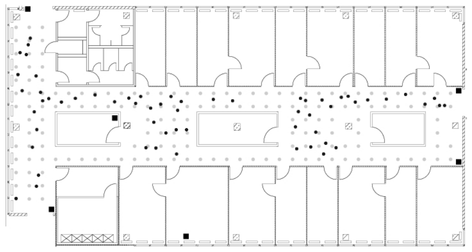


Figure 1: Figure 1: Floor Plan of the Test Environment. *There are 6 fixed access points which are denoted by black square markers. The "offline" training data were collected at the locations marked with the grey dots. We can see that the grey dots are spaced a meter apart.*

Grey circles give us a reference to mark the locations in which the "offline" measurements were taken and the black squares mark six access points. The reference locations give us a calibration of the signal strengths for in the building. These locations will be used to build our model to predict the locations of our hand-held device when it's location is unknown.

The hand-held device provided us with x and y coordinates, much like that of latitude and longitude on a map. There is also an orientation of the device. Signal strengths are given at eight orientations in 45 degree increments (0, 45, 90 and so on). 110 signal strength measurements were recorded for each of the six access points for every location and orientation combination.

We had a couple of ways of setting up the data in this analysis, without getting into too much detail below is a table of the variables in the data set that we will use in the analysis.

Variable	Description
t	timestamp in milliseconds since midnight, January 1, 1970 UTC
id	MAC address of the scanning device
pos	degree orientation of the user carrying the scanning device
MAC	MAC address of a responding peer (i.e. access point or a device in adhoc mode) with values for signal strength in dBm (Decibel-milliwatts)

Libraries Required

If you don't have these libraries installed in your R environment please do so.

```r
# Place R code here #
```

Read in the data

```r
# Use URL to bring in text data
url <- "http://rdatasciencecases.org/Data/offline.final.trace.txt"
# Read in entire document
txt <- readLines(url)
# Each line in the offline file has been read into R as a string in the
# character vector txt Use the function substr() to locate lines/strings
# that start with '#' and count how many we have
sum(substr(txt, 1, 1) == "#")
```

```
## [1] 5312
```

```r
length(txt)
```

```
## [1] 151392
```

With our "offline" data set there are 151,392 lines. The data documentation would tell us that we should expect there to be 146,080 lines (166 locations x 8 angles x 100 recordings). The difference between these two (151,392 and 146,080) is 5,312.

As a general rule it is better to check the entire data set instead of the first few lines.

Processing the Raw Data

Now that we have an idea of how to represent our data in R, we are now able to start the fun stuff. The data as is not in a format where we can simply use a function like read.table(). Our data are separated by semicolons. This gives us a basic structure in which we can process the data.

```r
strsplit(txt[4], ";")[[1]]
```

```
##  [1] "t=1139643118358"
##  [2] "id=00:02:2D:21:0F:33"
##  [3] "pos=0.0,0.0,0.0"
##  [4] "degree=0.0"
##  [5] "00:14:bf:b1:97:8a=-38,2437000000,3"
##  [6] "00:14:bf:b1:97:90=-56,2427000000,3"
##  [7] "00:0f:a3:39:e1:c0=-53,2462000000,3"
##  [8] "00:14:bf:b1:97:8d=-65,2442000000,3"
##  [9] "00:14:bf:b1:97:81=-65,2422000000,3"
## [10] "00:14:bf:3b:c7:c6=-66,2432000000,3"
## [11] "00:0f:a3:39:dd:cd=-75,2412000000,3"
## [12] "00:0f:a3:39:e0:4b=-78,2462000000,3"
## [13] "00:0f:a3:39:e2:10=-87,2437000000,3"
## [14] "02:64:fb:68:52:e6=-88,2447000000,1"
## [15] "02:00:42:55:31:00=-84,2457000000,1"
```

So, within these shorter strings, the "name" of the variable is separated by an '=' sign from the associated value. There are observations that contain multiple values where the separator is a ','. For example, "pos=0.0,0.0,0.0" consists of 3 position variables that are not named.

The vector, which is created by splitting on the semi-colon, and further split each element at the '=' sign is processed by splitting ','. We can create a function like the below:

```r
unlist(lapply(strsplit(txt[4], ";")[[1]], function(x) sapply(strsplit(x, "=")[[1]],
    strsplit, ",")))
```

```
##                    t       1139643118358                  id
##                  "t"     "1139643118358"                "id"
##   00:02:2D:21:0F:33                 pos        0.0,0.0,0.01
## "00:02:2D:21:0F:33"               "pos"               "0.0"
##        0.0,0.0,0.02        0.0,0.0,0.03              degree
##               "0.0"               "0.0"            "degree"
##                 0.0   00:14:bf:b1:97:8a   -38,2437000000,31
##               "0.0" "00:14:bf:b1:97:8a"               "-38"
##   -38,2437000000,32   -38,2437000000,33   00:14:bf:b1:97:90
##        "2437000000"                 "3" "00:14:bf:b1:97:90"
##   -56,2427000000,31   -56,2427000000,32   -56,2427000000,33
##               "-56"        "2427000000"                 "3"
##   00:0f:a3:39:e1:c0   -53,2462000000,31   -53,2462000000,32
## "00:0f:a3:39:e1:c0"               "-53"        "2462000000"
##   -53,2462000000,33   00:14:bf:b1:97:8d   -65,2442000000,31
##                 "3" "00:14:bf:b1:97:8d"               "-65"
##   -65,2442000000,32   -65,2442000000,33   00:14:bf:b1:97:81
##        "2442000000"                 "3" "00:14:bf:b1:97:81"
##   -65,2422000000,31   -65,2422000000,32   -65,2422000000,33
##               "-65"        "2422000000"                 "3"
##   00:14:bf:3b:c7:c6   -66,2432000000,31   -66,2432000000,32
## "00:14:bf:3b:c7:c6"               "-66"        "2432000000"
##   -66,2432000000,33   00:0f:a3:39:dd:cd   -75,2412000000,31
##                 "3" "00:0f:a3:39:dd:cd"               "-75"
##   -75,2412000000,32   -75,2412000000,33   00:0f:a3:39:e0:4b
##        "2412000000"                 "3" "00:0f:a3:39:e0:4b"
##   -78,2462000000,31   -78,2462000000,32   -78,2462000000,33
##                "-78"         "2462000000"                  "3"
##   00:0f:a3:39:e2:10   -87,2437000000,31   -87,2437000000,32
## "00:0f:a3:39:e2:10"                "-87"         "2437000000"
##   -87,2437000000,33   02:64:fb:68:52:e6   -88,2447000000,11
##                  "3" "02:64:fb:68:52:e6"                "-88"
##   -88,2447000000,12   -88,2447000000,13   02:00:42:55:31:00
##         "2447000000"                  "1" "02:00:42:55:31:00"
##   -84,2457000000,11   -84,2457000000,12   -84,2457000000,13
##                "-84"         "2457000000"                  "1"
```

We can make this more efficient by taking the “tokens” we created from above and form them into the appropriate form by using

```
# create a spilt using ;,=,','
tokens <- strsplit(txt[4], "[;=,]")[[1]]
```

Let’s look at the first 10 elements of our “tokens” variable to give us the information about the hand-held device. We can also extract the values of these variables.

```
tokens[1:10]
```

```
##  [1] "t"                 "1139643118358"     "id"
##  [4] "00:02:2D:21:0F:33" "pos"               "0.0"
##  [7] "0.0"               "0.0"               "degree"
## [10] "0.0"
```

```
# Extract values of variables
tokens[c(2, 4, 6:8, 10)]
```

```
## [1] "1139643118358"     "00:02:2D:21:0F:33" "0.0"
## [4] "0.0"               "0.0"               "0.0"
```

From our data information we know that these variables correspond to the variables time, MAC address, *x*,*y*,*z* and orientation.

Take a look at the recorded signals within this observation. These are the remaining values in the split vector

```
tokens[-(1:10)]
```

```
##  [1] "00:14:bf:b1:97:8a" "-38"               "2437000000"
##  [4] "3"                 "00:14:bf:b1:97:90" "-56"
##  [7] "2427000000"        "3"                 "00:0f:a3:39:e1:c0"
## [10] "-53"               "2462000000"        "3"
## [13] "00:14:bf:b1:97:8d" "-65"               "2442000000"
## [16] "3"                 "00:14:bf:b1:97:81" "-65"
## [19] "2422000000"        "3"                 "00:14:bf:3b:c7:c6"
## [22] "-66"               "2432000000"        "3"
## [25] "00:0f:a3:39:dd:cd" "-75"               "2412000000"
## [28] "3"                 "00:0f:a3:39:e0:4b" "-78"
## [31] "2462000000"        "3"                 "00:0f:a3:39:e2:10"
## [34] "-87"               "2437000000"        "3"
## [37] "02:64:fb:68:52:e6" "-88"               "2447000000"
## [40] "1"                 "02:00:42:55:31:00" "-84"
## [43] "2457000000"        "1"
```

These rows are a 4-column matrix or data frame giving the MAC address, signal, channel and device type. We need to detangle these and build a matrix. After the detanglement we can bind these columns with the values from the first 10 entries.

```r
tmp <- matrix(tokens[-(1:10)], ncol = 4, byrow = TRUE)
mat <- cbind(matrix(tokens[c(2, 4, 6:8, 10)], nrow = nrow(tmp), ncol = 6, byrow = TRUE),
    tmp)
# Check
dim(mat)
```

```
## [1] 11 10
```

Now that we know that the above chunk of code works we can build a function to iterate through each row in the input file.

```r
processLine = function(x) {
    tokens = strsplit(x, "[;=,]")[[1]]
    tmp = matrix(tokens[-(1:10)], ncol = 4, byrow = TRUE)
    cbind(matrix(tokens[c(2, 4, 6:8, 10)], nrow = nrow(tmp), ncol = 6, byrow = TRUE),
        tmp)
}
```

Try to apply the function to several lines of the input:

```r
tmp = lapply(txt[4:20], processLine)
sapply(tmp, nrow)
```

```
##  [1] 11 10 10 11  9 10  9  9 10 11 11  9  9  9  8 10 14
```

Now that we have done the work of looking at the data, deciding the best way to break it down and the best way of separating it, we can now look at making this into a data frame. Do execute this we are going to use the do.call() function.

```r
offline = as.data.frame(do.call("rbind", tmp))
# Check it
dim(offline)
```

```
## [1] 170  10
```

Validate Our Dataset

Work code through the entire data set

```r
lines <- txt[substr(txt, 1, 1) != "#"]
tmp = lapply(lines, processLine)
```

```
## Warning in matrix(tokens[c(2, 4, 6:8, 10)], nrow = nrow(tmp), ncol = 6, :
## data length exceeds size of matrix

## Warning in matrix(tokens[c(2, 4, 6:8, 10)], nrow = nrow(tmp), ncol = 6, :
## data length exceeds size of matrix

## Warning in matrix(tokens[c(2, 4, 6:8, 10)], nrow = nrow(tmp), ncol = 6, :
## data length exceeds size of matrix

## Warning in matrix(tokens[c(2, 4, 6:8, 10)], nrow = nrow(tmp), ncol = 6, :
## data length exceeds size of matrix

## Warning in matrix(tokens[c(2, 4, 6:8, 10)], nrow = nrow(tmp), ncol = 6, :
## data length exceeds size of matrix

## Warning in matrix(tokens[c(2, 4, 6:8, 10)], nrow = nrow(tmp), ncol = 6, :
```

```
## data length exceeds size of matrix
```

Well, lets dig to see what is going on here

```r
options(error = recover, warn = 2)
tmp = lapply(lines, processLine)
```

```
## Warning in matrix(tokens[c(2, 4, 6:8, 10)], nrow = nrow(tmp), ncol = 6, :
## data length exceeds size of matrix

## Warning in matrix(tokens[c(2, 4, 6:8, 10)], nrow = nrow(tmp), ncol = 6, :
## data length exceeds size of matrix

## Warning in matrix(tokens[c(2, 4, 6:8, 10)], nrow = nrow(tmp), ncol = 6, :
## data length exceeds size of matrix

## Warning in matrix(tokens[c(2, 4, 6:8, 10)], nrow = nrow(tmp), ncol = 6, :
## data length exceeds size of matrix

## Warning in matrix(tokens[c(2, 4, 6:8, 10)], nrow = nrow(tmp), ncol = 6, :
## data length exceeds size of matrix

## Warning in matrix(tokens[c(2, 4, 6:8, 10)], nrow = nrow(tmp), ncol = 6, :
## data length exceeds size of matrix
```

We will need to modify the function we made call ProcessLine(). We need to discard observations or add a single channel, and type. We will choose to remove these observations as they do not help us in developing our position system. Change the function to return NULL if the tokens vector only has 10 elements. The revised function becomes:

```r
processLine = function(x) {
    tokens = strsplit(x, "[;=,]")[[1]]

    if (length(tokens) == 10)
        return(NULL)

    tmp = matrix(tokens[-(1:10)], , 4, byrow = TRUE)
    cbind(matrix(tokens[c(2, 4, 6:8, 10)], nrow(tmp), 6, byrow = TRUE), tmp)
}
```

Try again

```r
options(error = recover, warn = 1)
tmp <- lapply(lines, processLine)
offline <- as.data.frame(do.call("rbind", tmp), stringsAsFactors = FALSE)

dim(offline)
```

```
## [1] 1181628      10
```

From the dim() function we can see we returned 1.18M rows. Our next step is convert our data into the proper data types. So we can do get to analysis and start looking at what this data is telling us.

Cleaning the Data for Analysis

First and foremost we need to name our variables. When naming variables, we have to make them meaningful.

```
names(offline) = c("time", "scanMac", "posX", "posY", "posZ", "orientation",
    "mac", "signal", "channel", "type")
```

Now we covert the position, signal, and time variables to numeric.

```
numVars = c("time", "posX", "posY", "posZ", "orientation", "signal")
offline[numVars] = lapply(offline[numVars], as.numeric)
```

The device could use a change to something that is more compreshensible than numbers 1 and 3. To facilitate this we can turn the variable into a factor with the levels, of "adhoc" and "access point". However, we will use only the signal strengths measured to the fix access points to develop and test our model. With this information now in hand we will remove records for adhoc measurements and remove the type variable from our data frame.

```
offline = offline[offline$type == "3", ]
offline = offline[, "type" != names(offline)]
dim(offline)
```

```
## [1] 978443      9
```

This removed over 100K observations from our data frame.

From here we now consider the variable time. From the docuementation, time is measured in the number of milliseconds from midnight on January 1st, 1970. We are able to scale the value of time to seconds and then simpy set the class of the time element in order to see the values as date-times in R. We will keep the more accurate time in rawTime in the case it is needed at a later time for analysis.

```
offline$rawTime = offline$time
offline$time = offline$time/1000
class(offline$time) = c("POSIXt", "POSIXct")
```

```
# Check what is going on
unlist(lapply(offline, class))
```

```
##       time1       time2     scanMac        posX        posY        posZ
##    "POSIXt"   "POSIXct" "character"   "numeric"   "numeric"   "numeric"
## orientation         mac      signal     channel     rawTime
##   "numeric" "character"   "numeric" "character"   "numeric"
```

From the looks of it, it would appear that we have the right data types and structure, now lets go through another check and look at a summary of our data to see if our data makes sense. We are going to be looking for sane values in the descriptive statistics.

```
summary(offline[, numVars])
```

```
##       time                          posX            posY       
##  Min.   :2006-02-11 01:31:58   Min.   : 0.00   Min.   : 0.000  
##  1st Qu.:2006-02-11 07:21:27   1st Qu.: 2.00   1st Qu.: 3.000  
##  Median :2006-02-11 13:57:58   Median :12.00   Median : 6.000  
##  Mean   :2006-02-16 08:57:37   Mean   :13.52   Mean   : 5.897  
##  3rd Qu.:2006-02-19 08:52:40   3rd Qu.:23.00   3rd Qu.: 8.000  
##  Max.   :2006-03-09 14:41:10   Max.   :33.00   Max.   :13.000  
##       posZ    orientation        signal     
##  Min.   :0   Min.   :  0.0   Min.   :-99.0  
##  1st Qu.:0   1st Qu.: 90.0   1st Qu.:-69.0  
##  Median :0   Median :180.0   Median :-60.0  
##  Mean   :0   Mean   :167.2   Mean   :-61.7  
##  3rd Qu.:0   3rd Qu.:270.0   3rd Qu.:-53.0  
##  Max.   :0   Max.   :359.9   Max.   :-25.0  
```

So far, so good. Let's check the character variables.

```r
summary(sapply(offline[, c("mac", "channel", "scanMac")], as.factor))
```

```
##                   mac                channel                    scanMac
##  00:0f:a3:39:e1:c0:145862   2462000000:189774   00:02:2D:21:0F:33:978443
##  00:0f:a3:39:dd:cd:145619   2437000000:152124
##  00:14:bf:b1:97:8a:132962   2412000000:145619
##  00:14:bf:3b:c7:c6:126529   2432000000:126529
##  00:14:bf:b1:97:90:122315   2427000000:122315
##  00:14:bf:b1:97:8d:121325   2442000000:121325
##  (Other)          :183831   (Other)   :120757
```

From what we can see from the above character summary we have a couple of items we need may need to modify. 1. We only have one value for scanMac. If you remember this is the MAC address for the hand-held device from which the measurement is taken. This variable will be discarded. 2. All of the values for posZ, the elevation of the hand-held device are 0. Why zero? Well we are just measuring the first floor. We can remove that variable as well.

We are now ready to start Exploratory Data Analysis (EDA)

```r
offline <- offline[, !(names(offline) %in% c("scanMac", "posZ"))]
```

Exploratory Data Analysis

Orientation

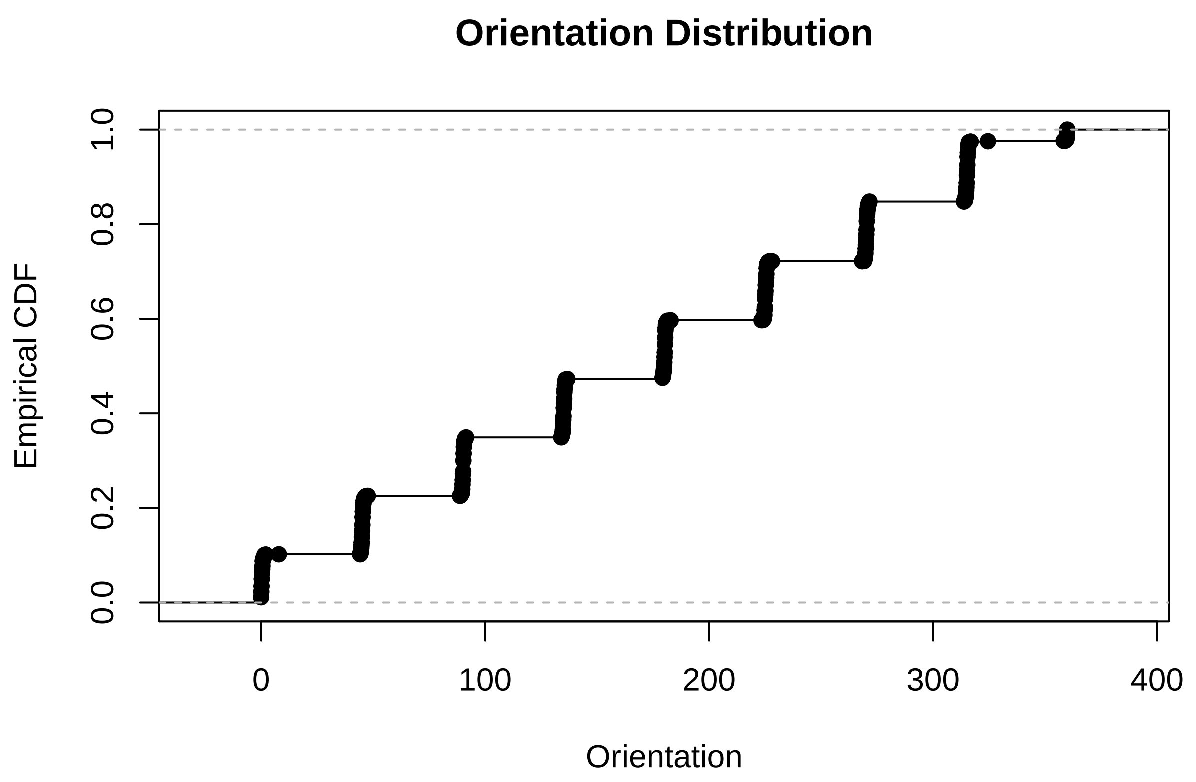
We have eight values for orientation, or we should. If the reader recalls the observations for orientation were set up to be at levels of 0 degrees, 45 degress, 90 degress and so on.

```r
length(unique(offline$orientation))
```

```
## [1] 203
```

So, 203 is greater than 8. So we have more than eight values. We will take a closer look at the distribution of the variable orientation. We are going to do this by looking at an empirical cumulative distribution function, or ECDF.

```r
plot(ecdf(offline$orientation), main = "Orientation Distribution", xlab = "Orientation",
    ylab = "Empirical CDF")
```

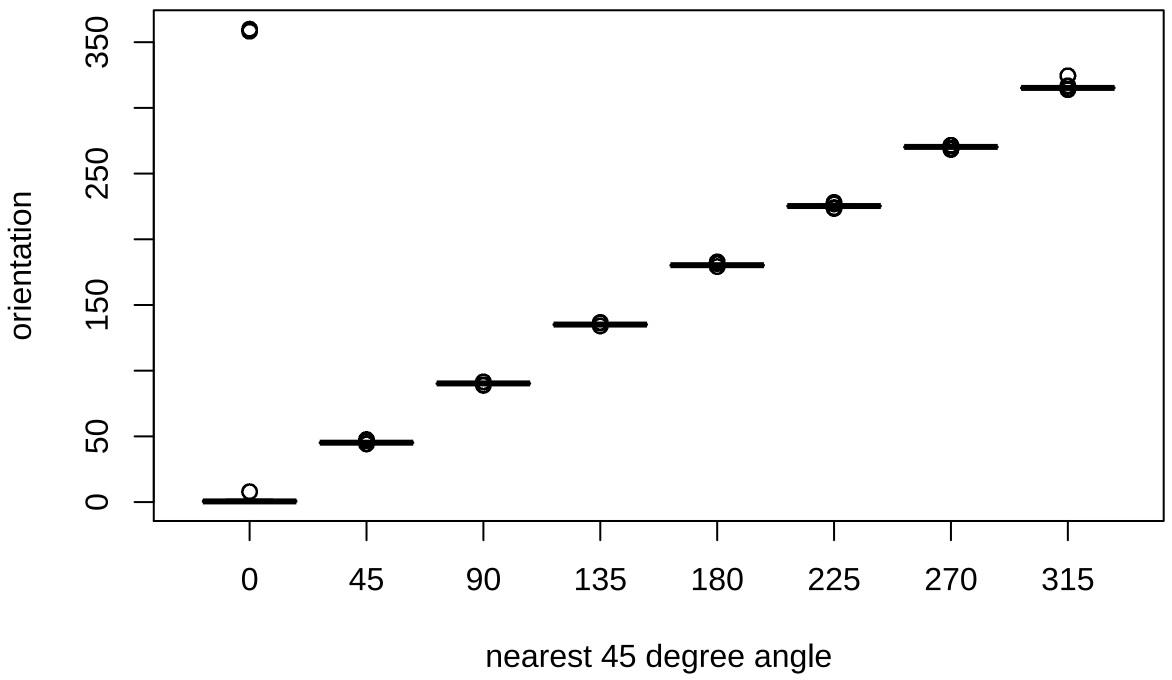


Form the plot we do see observations clustered around 0, 45, 90 degress and so on, but we clearly have spread between. So, for example we have some 47.5 degress, 358.2 degrees and so on. This is not a loss, this information could be valuable as is. We could also gain value from creating a bin for these values to match the orignial eight values. To accomplish this we are going to create a function.

```
roundOrientation = function(angles) {
    refs = seq(0, by = 45, length = 9)
    q = sapply(angles, function(o) which.min(abs(o - refs)))
    c(refs[1:8], 0)[q]
}
# We now use our function to created the rounded angles
offline$angle <- roundOrientation(offline$orientation)
```

Let's continue our analysis of orientation with a box plot with the new angles.

```
with(offline, boxplot(orientation ~ angle, xlab = "nearest 45 degree angle",
    ylab = "orientation"))
```



Our new angle variable worked, the outlier at the 0 degree at the top left are the values near 360 degrees that have been mapped to zero.