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.

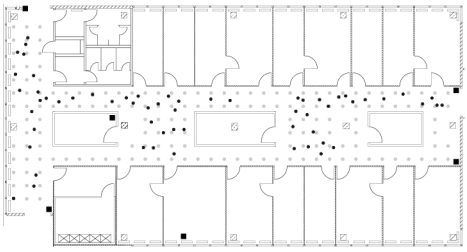


Figure 1: Figure 1: Floor Plan of the Test Environment. *There are 6 fixed access points which are denoted by black square markers. The "offline" training data were collected at the locations marked with the grey dots. We can see that the grey dots are spaced a meter apart.*

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.

```
library(codetools)
library(lattice)
library(fields)
```

```
## Loading required package: spam
```

```
## Loading required package: dotCall64
```

```
## Loading required package: grid
```

```
## Spam version 2.1-2 (2017-12-21) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.
```

```
##
## Attaching package: 'spam'
```

```
## The following objects are masked from 'package:base':
##
##     backsolve, forwardsolve
```

```
## Loading required package: maps
```

Read in the data

```
# 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
```

```
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.

```
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:

```
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.

```
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.

```
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:

```
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.

```
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

```
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

```
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:

```
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

```
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 readable 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 documentation, 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 simply 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.

```
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)

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

Exploratory Data Analysis

Orientation

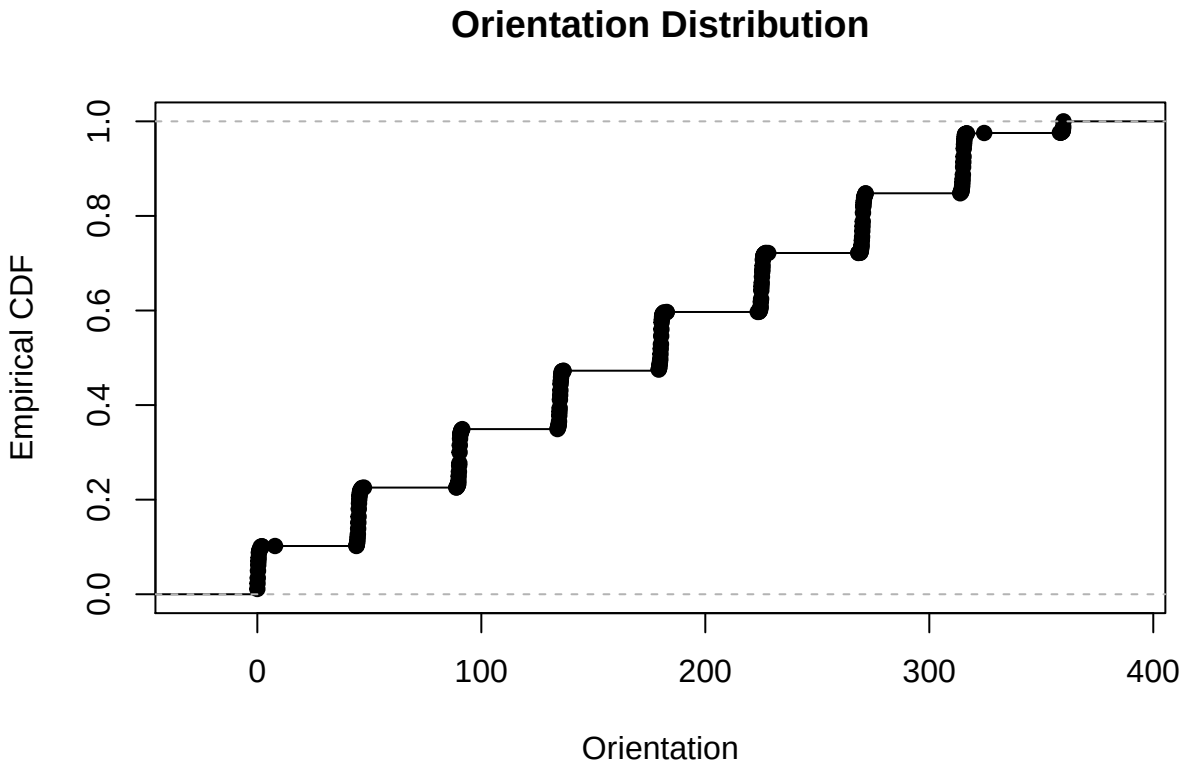
We have eight values for orientation, or we should. If the reader recalls the observations for orientation were set up to be at levels of 0 degrees, 45 degrees, 90 degrees and so on.

```
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.

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

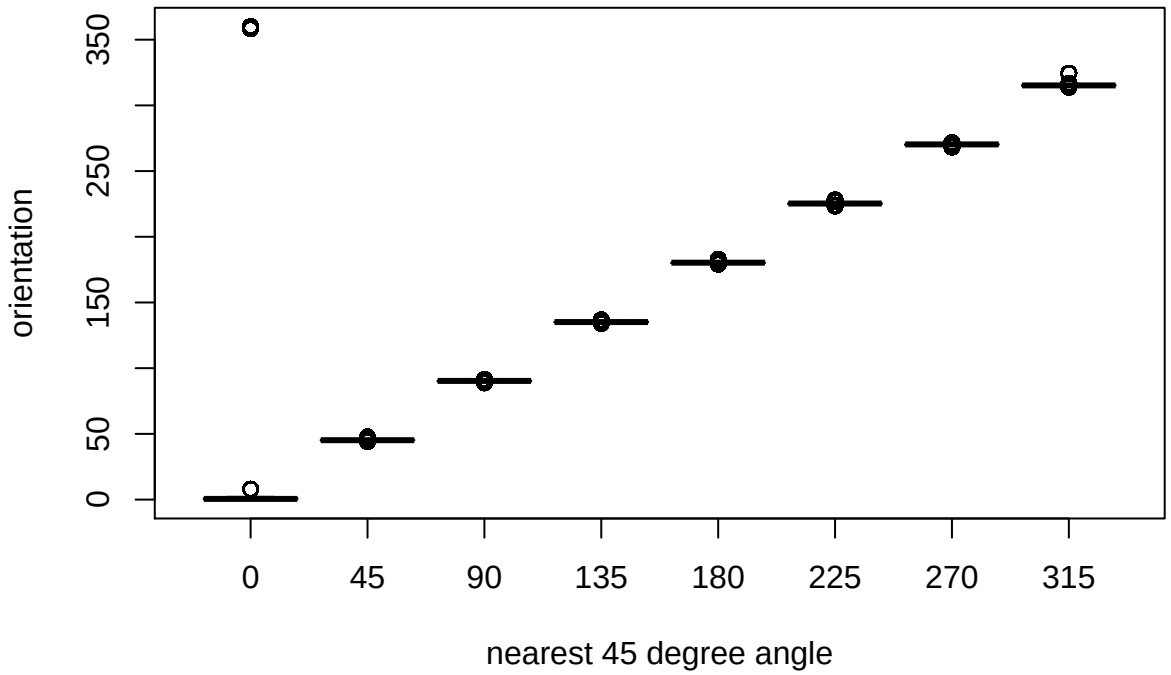


Form the plot we do see observations clustered around 0, 45, 90 degrees and so on, but we clearly have spread between. So, for example we have some 47.5 degrees, 358.2 degrees and so on. This is not a loss, this information could be valuable as is. We could also gain value from creating a bin for these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

MAC Address Exploration

From the summary function above we observed that 126,529 occurrences of the address 00:14:br:3b:c7:c6, and the exact same number of occurrences of channel 2432000000. This would lead us to believe that there is one-to-one mapping. To prove this let's run a couple of test.

```
c(length(unique(offline$mac)), length(unique(offline$channel)))
```

```
## [1] 12  8
```

We find that there are 12 MAC addresses and eight channels. If we remember, there were only six access points. So why do see that there are eight channels and 12 MAC addresses? Re-reading the documentation we can see that there are additional access points that are not part of the testing area. These were not on the floor plan in figure 1. We need to check the counts of observations for the various MAC addresses.

```
table(offline$mac)
```

```
##
## 00:04:0e:5c:23:fc 00:0f:a3:39:dd:cd 00:0f:a3:39:e0:4b 00:0f:a3:39:e1:c0
##               418            145619             43508            145862
## 00:0f:a3:39:e2:10 00:14:bf:3b:c7:c6 00:14:bf:b1:97:81 00:14:bf:b1:97:8a
##             19162            126529            120339            132962
## 00:14:bf:b1:97:8d 00:14:bf:b1:97:90 00:30:bd:f8:7f:c5 00:e0:63:82:8b:a9
##            121325            122315               301               103
```

So, we can see that the first, and last two MAC addresses are not close to the testing area. They could have not been working for the short amount of time during the measurement process. These addresses also don't have as high of counts.

The documentation tells us that the access points consist of five Linksys/Cisco and one Lancom L-54g routers. Using MAC Address Lookup we find that the MAC addresses that start with 00:14:bf belong to Linksys, the devices that start with 00:0f:a3 belong to Alpha Networks, and Lancom devices start with 00:a0:57. This

means that there is a discrepancy with the documentation. With this known we will keep the top seven devices.

```
subMacs <- names(sort(table(offline$mac), decreasing = TRUE))[1:7]
offline <- offline[offline$mac %in% subMacs, ]
```

Let's validate by creating a table of counts for the remaining MACxchannel combinations and confirm there is one non-zero entry in each row.

```
macChannel <- with(offline, table(mac, channel))
apply(macChannel, 1, function(x) sum(x > 0))
```

```
## 00:0f:a3:39:dd:cd 00:0f:a3:39:e1:c0 00:14:bf:3b:c7:c6 00:14:bf:b1:97:81
##                 1                 1                 1                 1
## 00:14:bf:b1:97:8a 00:14:bf:b1:97:8d 00:14:bf:b1:97:90
##                 1                 1                 1
```

Now we do see that there is a one-to-one relationship between MAC address and channel for the seven devices. From here we can remove channel from offline.

```
offline <- offline[, "channel" != names(offline)]
```

Explore the Position of the Hand-Held Device

Finally, let's look at the position variables, posX and posY. How many different locations do we have data? Using the by() function we can add up the numbers of rows in our data for each unique (x,y) combination. This is initiated with by creating a data frame for each location.

```
locDF <- with(offline, by(offline, list(posX, posY), function(x) x))
# Check
length(locDF)
```

```
## [1] 476
```

So this list is longer than the number of combinations of actual (x,y) locations that were recorded. Let's check to see if we have missing values.

```
sum(sapply(locDF, is.null))
```

```
## [1] 310
```

Let's drop those combinations where nothing exists. Also we need to confirm how many values we are stuck with.

```
locDF <- locDF[!sapply(locDF, is.null)]
# Check
length(locDF)
```

```
## [1] 166
```

We can operate on each of these data frame to determine the number of observations recorded at each location with:

```
locCounts <- sapply(locDF, nrow)

# If we want to keep the position information with the lcoation, we do this
# with
locCounts <- sapply(locDF, function(df) c(df[1, c("posX", "posY")], count = nrow(df)))
# Confirm we have a matrix
class(locCounts)
```

```
## [1] "matrix"
```

```
# Confirm 3 rows
dim(locCounts)
```

```
## [1]   3 166
```

```
# Examine a few of the counts
locCounts[, 1:8]
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## posX  0    1    2    0    1    2    0    1
## posY  0    0    0    1    1    1    2    2
## count 5505 5505 5506 5524 5543 5558 5503 5564
```

At each position we roughly have 5,500 recordings at each position. If we think about it, with 8 orientations X 110 replications X 7 access points, which is 6,100 signal strength measurements.

We can visualize all 166 counts by adding the counts as text at their respective locations, changing the size and angle of the characters to avoid overlapping text. We first transpose the matrix to keep the locations as columns we then make the plot with:

```
pdf(file = "Geo_XYByCount.pdf", width = 10)
oldPar = par(mar = c(3.1, 3.1, 1, 1))

locCounts = t(locCounts)
plot(locCounts, type = "n", xlab = "", ylab = "")
text(locCounts, labels = locCounts[, 3], cex = 0.8, srt = 45)

par(oldPar)
dev.off()
```

```
## pdf
##   2
```

This picture in figure 2 should look familiar. It looks just like our floorplan from figure 1. We can see that there are roughly the same number signals detected at each location.

We have examined all of the variables with exception of time and signal. The process has helped us clean our data and reduce it to those records that are relevant to our analysis. We now leave the examination of the signals to the next section where we study the it's distributional properties. Time is not directly related to our model, but it does give us the order in which the observations were taken. In an experiment time could help uncover potential bias. We could encounter bias if the person carrying the hand-held device may have changed how the device was carried as the experiment progressed and this could have changed the strength of the signal.

We will need to read the online data in R. We will use a readData() function to accomplish this.

```
readData = function(filename = "http://rdatasciencecases.org/Data/offline.final.trace.txt",
    subMacs = c("00:0f:a3:39:e1:c0", "00:0f:a3:39:dd:cd", "00:14:bf:b1:97:8a",
        "00:14:bf:3b:c7:c6", "00:14:bf:b1:97:90", "00:14:bf:b1:97:8d", "00:14:bf:b1:97:81")) {
    txt = readLines(filename)
    lines = txt[substr(txt, 1, 1) != "#"]
    tmp = lapply(lines, processLine)
    offline = as.data.frame(do.call("rbind", tmp), stringsAsFactors = FALSE)

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

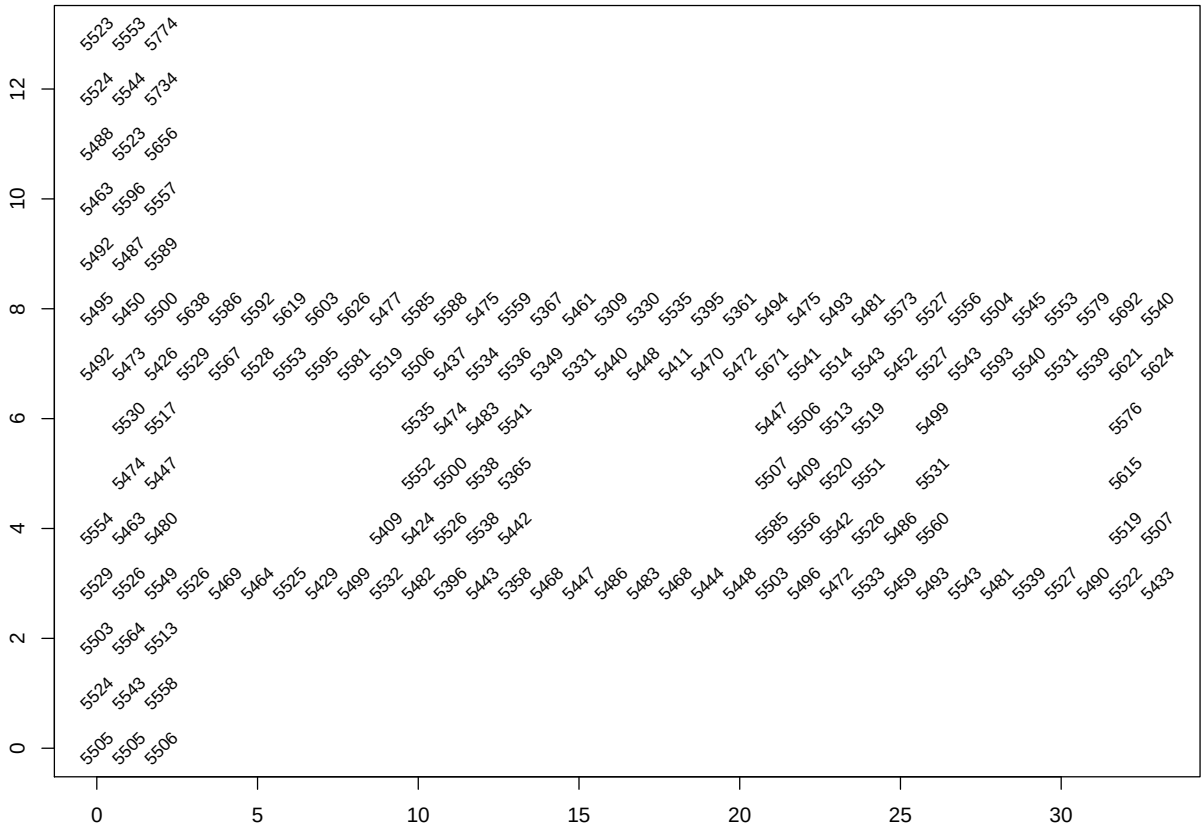


Figure 2: Figure 2: Counts of signals at each position. *Plotted at each location in the building is the total number of signals detected from all access points for the offline data. In a perfect world there is 110 signals measured with 8 angles for each 6 access points for a total of 5,280 recordings. These data include a 7th MAC address and not all signals were detected, so there are about 5,500 recordings at each location.*

```
    # keep only signals from access points
    offline = offline[offline$type == "3", ]

    # drop scanMac, posZ, channel, and type - no info in them
    dropVars = c("scanMac", "posZ", "channel", "type")
    offline = offline[, !(names(offline) %in% dropVars)]

    # drop more unwanted access points
    offline = offline[offline$mac %in% subMacs, ]

    # convert numeric values
    numVars = c("time", "posX", "posY", "orientation", "signal")
    offline[numVars] = lapply(offline[numVars], as.numeric)

    # convert time to POSIX
    offline$rawTime = offline$time
    offline$time = offline$time/1000
    class(offline$time) = c("POSIXt", "POSIXct")

    # round orientations to nearest 45
    offline$angle = roundOrientation(offline$orientation)

    return(offline)
}

offlineRedo = readData()
# Check to see if this fucntion matches what we did.
identical(offline, offlineRedo)
```

```
## [1] TRUE
```

Confirm what variables are global

```
findGlobals(readData, merge = FALSE)$variables
```

```
## [1] "as.numeric"  "processLine"
```

Signal Strength Analysis

Using visualization and statistical summaries we now turn to investigating the response variable, signal strength. We are going to have to learn more about how the signals behave before designing a model for IPS. To do this we need to ask the following questions.

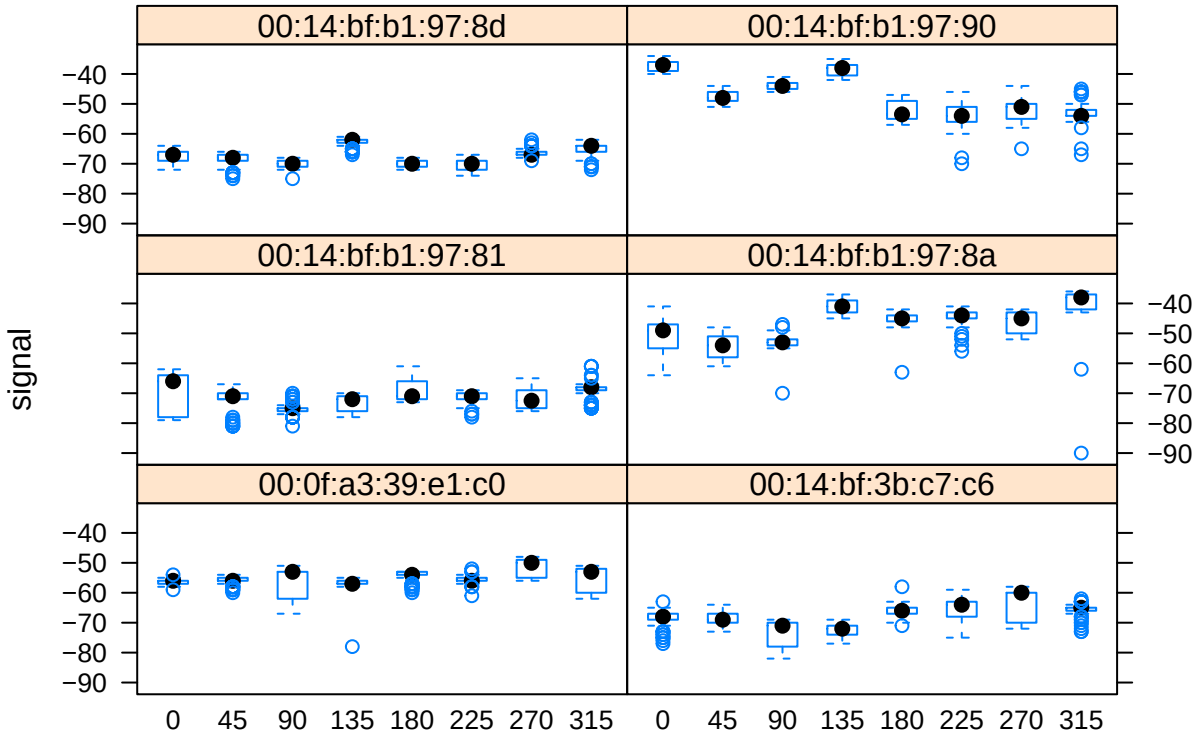
- Signal strength has been measured to an access point multiple times at each location and orientation. So, how do these signal strengths behave? What is the distribution of the repeated measurements at each location and orientation? Does signal strength behave the same at all of the locations? Or, is this a case of location, orientation, and access points having an effect on the distribution?
- In perfect conditions (lab setting), signal strength decays linearly with log distance and a simple triangulation using the signal strength from three access points can give an accurate pinpoint to the location of a device. The physical characteristics of a building and the activity in it can add a large amount of "noise" to signal strength measurements. Our questions become how do we describe the relationship between the signal strength and the distance from the device to the access point? How does orientation affect this relationship. Finally, is this relationship the same for all access points?

Distribution of Signal Strength

We need to compare the distribution of signal strength at different orientations and for different access points. This is going to require us to subdivide our data. We are interested in seeing if these distributions are normal or skewed. We also want to look at the variances.

We will fix a location on the map to see the impact of orientation and signal strength, while the experimenter rotates through the eight angles. We will also measure the MAC addresses separately because at an orientation of 90 degrees the experimenter may be facing toward one access point and away from another. We will check this with a boxplot.

```
bwplot(signal ~ factor(angle) | mac, data = offline, subset = posX == 2 & posY ==
    12 & mac != "00:0f:a3:39:dd:cd", layout = c(2, 3))
```



The signal strength does vary with the orientation for both close and distant access points. Recall from the summary section that signal strengths are measured in negative values.

```
summary(offline$signal)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -98.00  -67.00  -59.00  -59.92  -53.00  -25.00
```

The small values (-98) are weaker signals, while the larger values are the stronger signals. When other locations are examined we see a similar dependence of signal strength on angle.

```
pdf(file = "Geo_DensitySignalByMacAngle.pdf", width = 8, height = 12)
oldPar = par(mar = c(3.1, 3, 1, 1))

densityplot(~signal | mac + factor(angle), data = offline, subset = posX ==
    24 & posY == 4 & mac != "00:0f:a3:39:dd:cd", bw = 0.5, plot.points = FALSE)

par(oldPar)
dev.off()
```

```
## pdf
##   2
```

Many of the distributions in figure 3 look normal, but there are some serious deviations with secondary modes and skewness. The center of the distribution does vary with the angle and MAC address. This would be indicative of a relationship.

Basically we would need thousands of boxplots to see all of the relationships, we don't have time for that. We will examine measures of center and measures of spread of signal strength for all location-orientation-access point combinations. For each combination we basically have around 100 observations. To compute summary statistics for these various combinations we need to make a factor that contains all of the unique combinations of the observed (x,y) pairs for the 166 locations.

```
offline$posXY <- paste(offline$posX, offline$posY, sep = "-")
```

Now, we create a list of data frames for every combination of (x, y), angle, and access point as follows

```
byLocAngleAP <- with(offline, by(offline, list(posXY, angle, mac), function(x) x))
```

We should be able to calculate summary statistics on each of these data frames with

```
signalSummary = lapply(byLocAngleAP, function(oneLoc) {
    ans = oneLoc[1, ]
    ans$medSignal = median(oneLoc$signal)
    ans$avgSignal = mean(oneLoc$signal)
    ans$num = length(oneLoc$signal)
    ans$sdSignal = sd(oneLoc$signal)
    ans$iqrSignal = IQR(oneLoc$signal)
    ans
})

offlineSummary = do.call("rbind", signalSummary)
```

Now we take our results from above and take a visual look at the measures of spread with boxplots. Average signal will be converted to a categorical variable for this exercise.

```
breaks = seq(-90, -30, by = 5)
bwplot(sdSignal ~ cut(avgSignal, breaks = breaks), data = offlineSummary, subset = mac !=
    "00:0f:a3:39:dd:cd", xlab = "Mean Signal", ylab = "SD Signal")
```

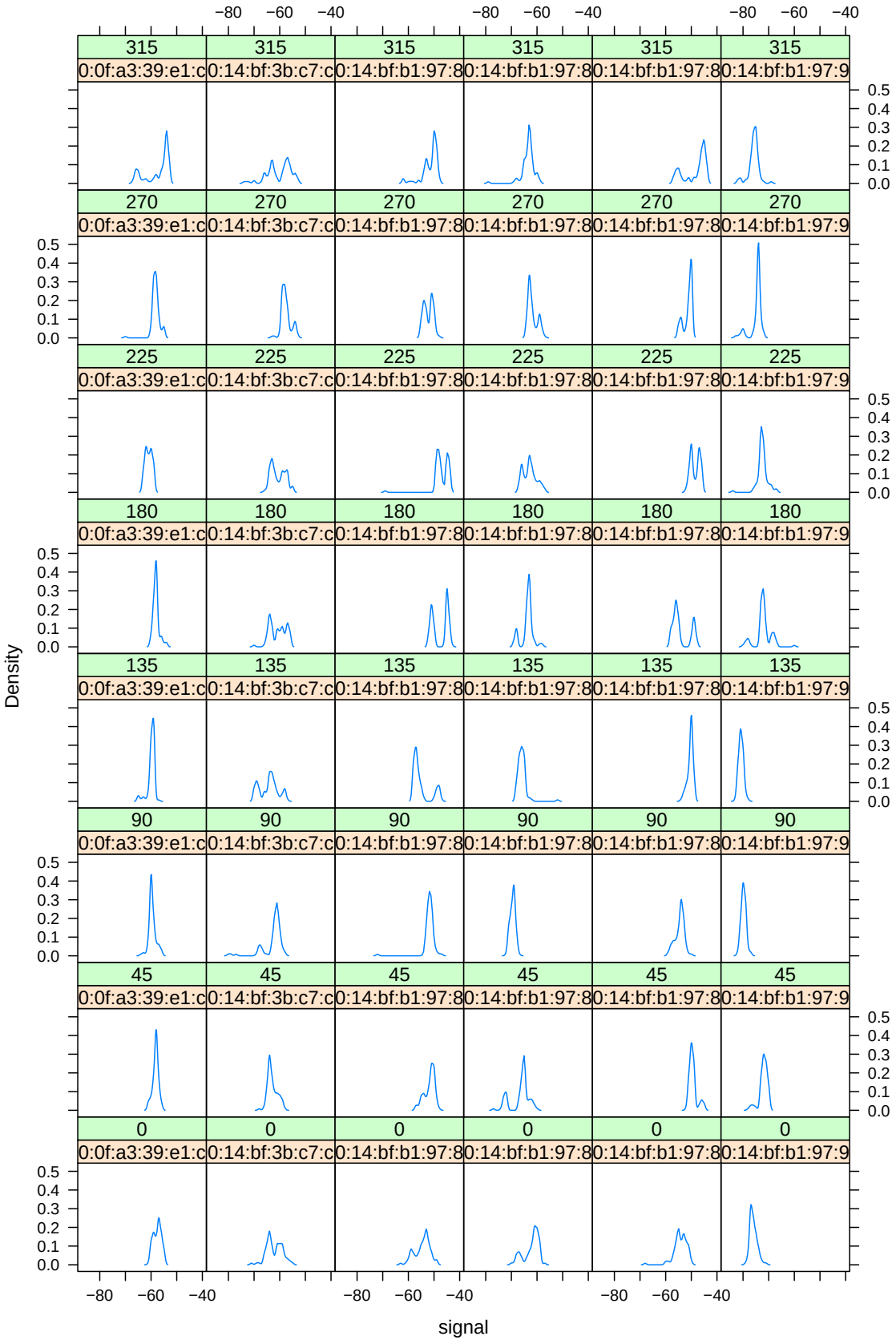
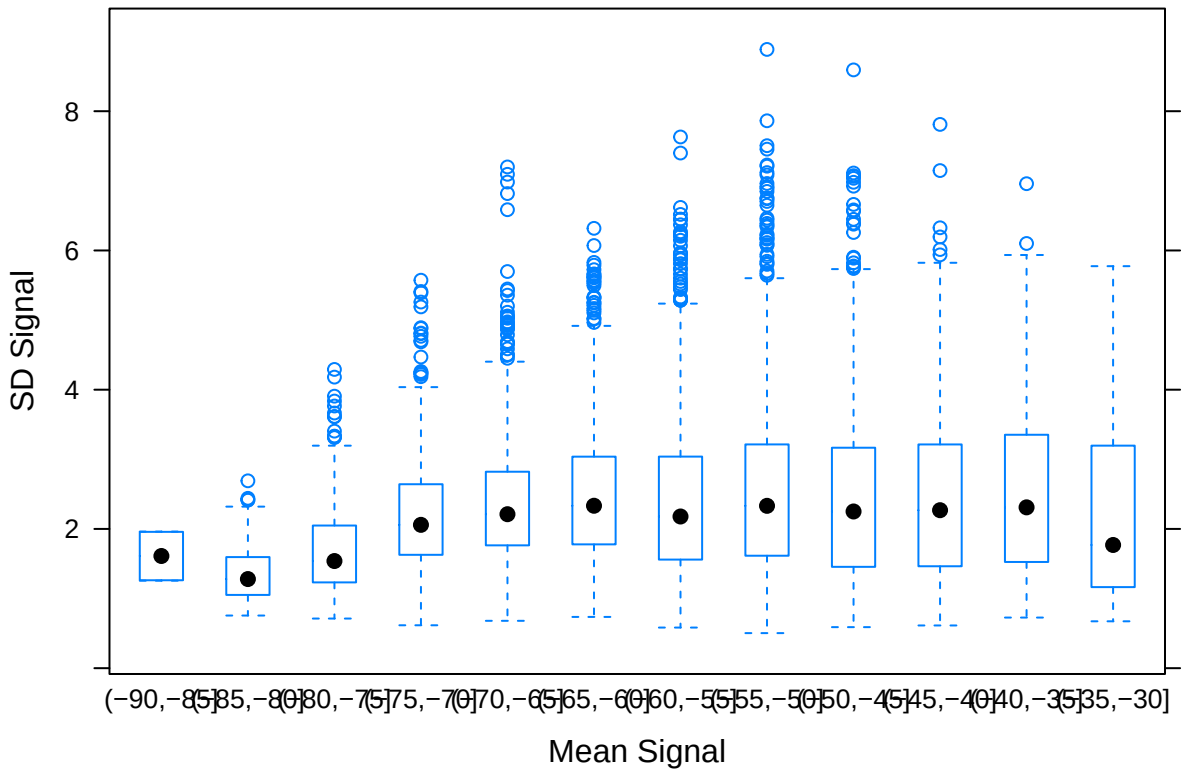


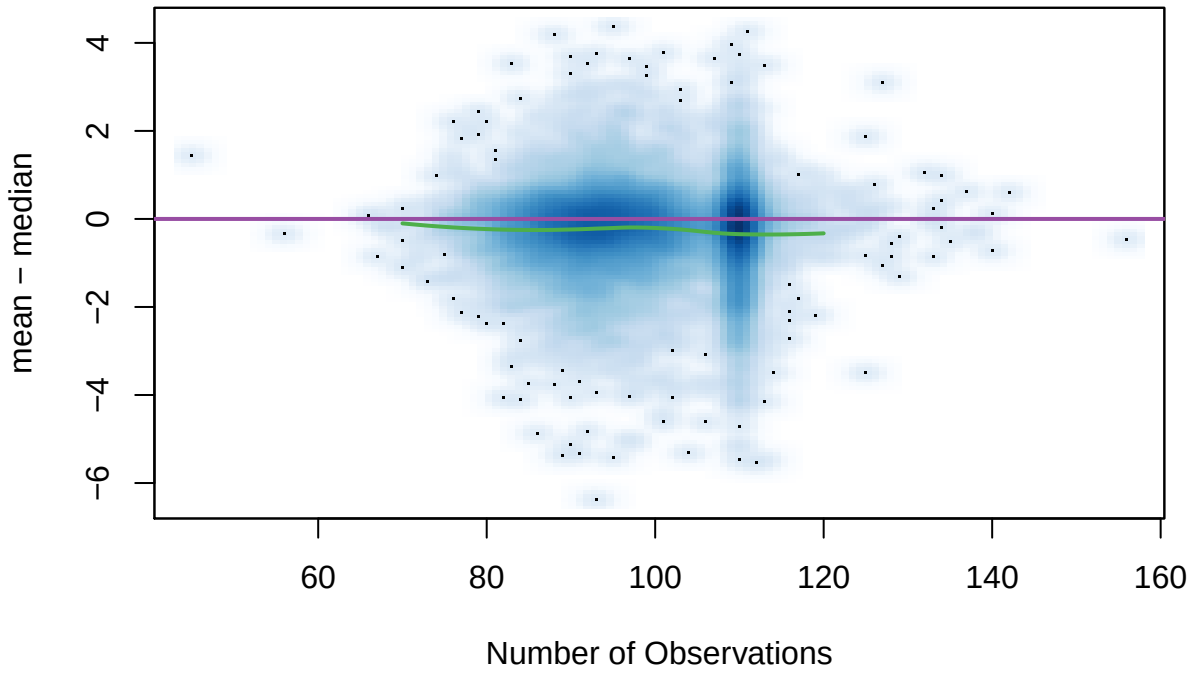
Figure 3: Figure 3: Distribution of Signal by Angle for Each Access Point. *The density curves shown for signal strengths are measured at the position $x = 24$ and $y = 4$. Many look normal, with others look like there is skew present*



We see that weakest signals have the smaller standard deviation and it would appear that as signal strength increases so does the standard deviation. This is going to be an important concept going into modeling.

We can examine the skew of signal strength by plotting the difference avgSignal - medSignal, against the number of observations. We are going to use a smooth scatter to help with over plotting. We will also add a local average of the difference between the mean and the median to better help address the size.

```
with(offlineSummary, smoothScatter((avgSignal - medSignal) ~ num, xlab = "Number of Observations",
    ylab = "mean - median"))
abline(h = 0, col = "#984ea3", lwd = 2)

lo.obj = with(offlineSummary, loess(diff ~ num, data = data.frame(diff = (avgSignal -
    medSignal), num = num)))

lo.obj.pr = predict(lo.obj, newdata = data.frame(num = (70:120)))
lines(x = 70:120, y = lo.obj.pr, col = "#4daf4a", lwd = 2)
```



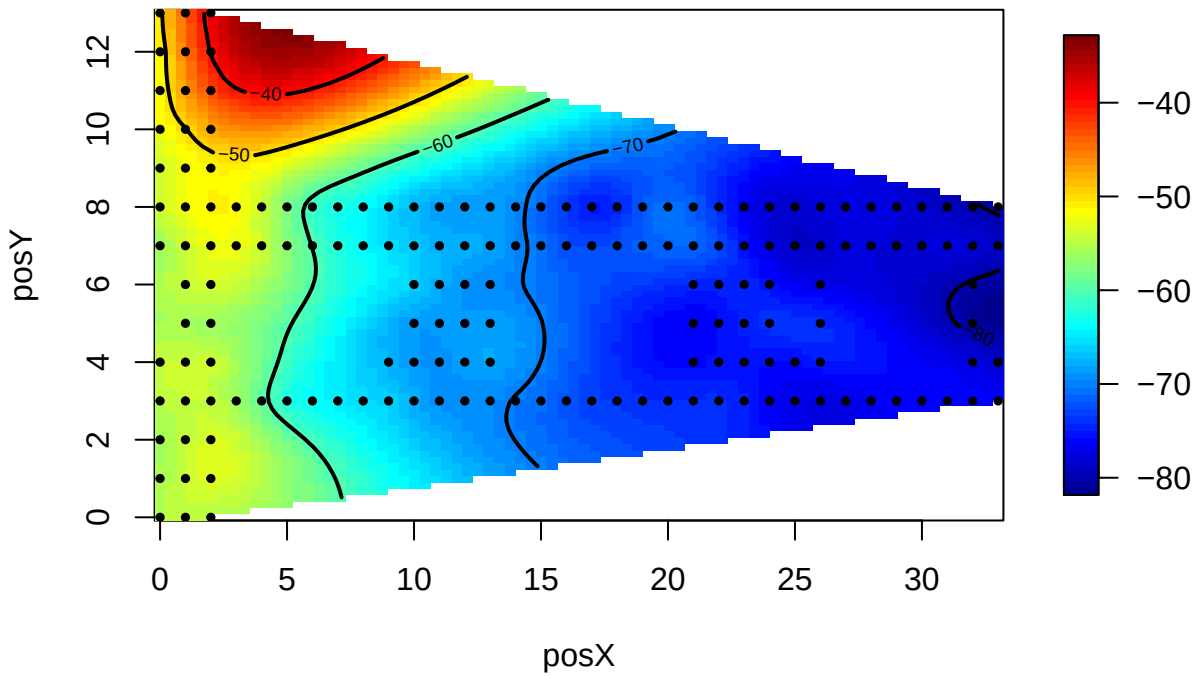
The above comparison of mean and median signal strength displays a smoothed scatter plot. The difference between the mean and median signal strength for each combination of location, access point and angle against the number of observations. The differences are close to zero with a typical deviation of 1 to 2 bBm.

Another plot we are going to try is with the use of predictSurface() to predict the value for the fitted surface at a grid of the observed posX and posY values with a contour plot. This is similar to what you see in a topographical map. The areas where we have a strong signal corresponds to the higher elevated areas of a contour map. Just like in the last visualization we want to control for the access point and the orientation. We will begin by selecting one MAC address and one orientation to examine.

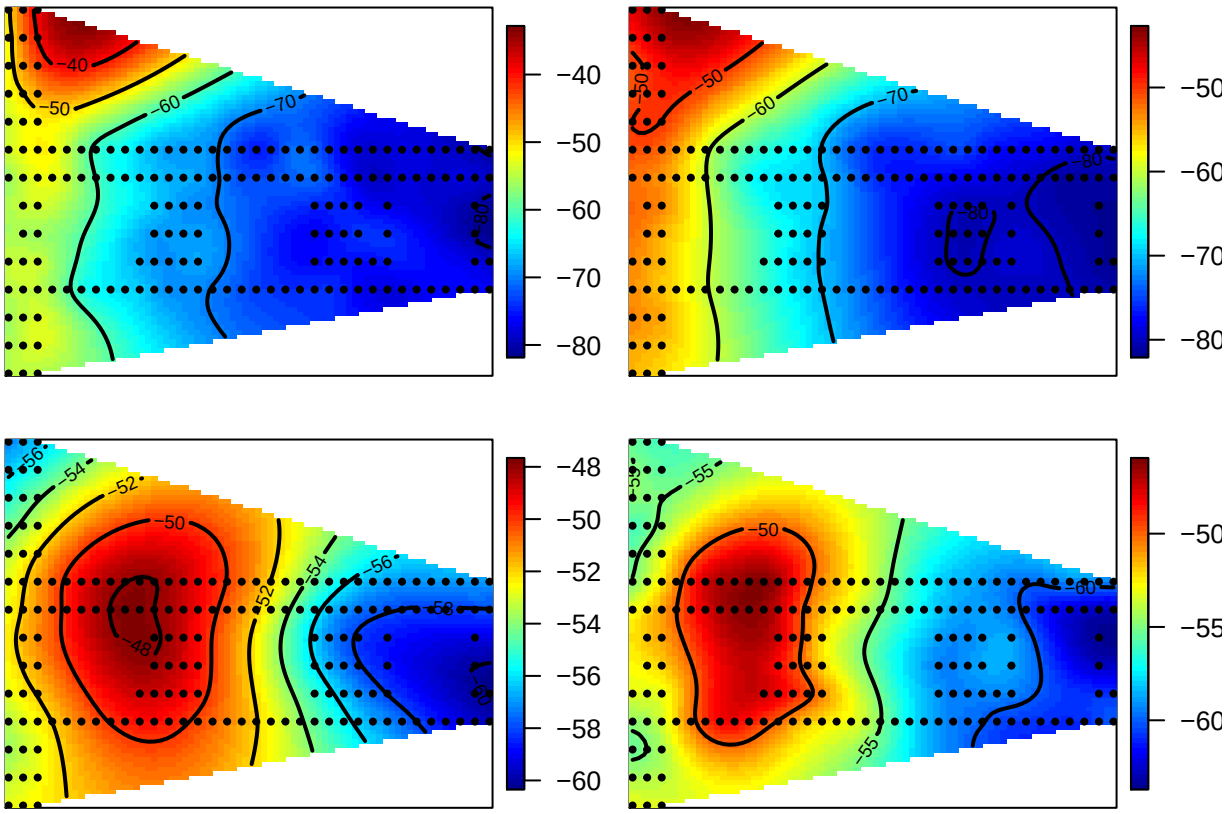
```
oneAPAngle <- subset(offline, mac == subMacs[5] & angle == 0)
```

We can make a topographical map using color, just like a heatmap. The fields package uses the method of thin plate splines to fit a surface to the signal strength value at the observed locations.

```
oneAPAngle = subset(offlineSummary, mac == subMacs[5] & angle == 0)

smoothSS = Tps(oneAPAngle[, c("posX", "posY")], oneAPAngle$avgSignal)

vizSmooth = predictSurface(smoothSS)

plot.surface(vizSmooth, type = "C")

points(oneAPAngle$posX, oneAPAngle$posY, pch = 19, cex = 0.5)
```




```
surfaceSS = function(data, mac, angle = 45) {
    require(fields)
    oneAPAngle = data[data$mac == mac & data$angle == angle, ]
    smoothSS = Tps(oneAPAngle[, c("posX", "posY")], oneAPAngle$avgSignal)
    vizSmooth = predictSurface(smoothSS)
    plot.surface(vizSmooth, type = "C", xlab = "", ylab = "", xaxt = "n", yaxt = "n")
    points(oneAPAngle$posX, oneAPAngle$posY, pch = 19, cex = 0.5)
}

parCur = par(mfrow = c(2, 2), mar = rep(1, 4))

mapply(surfaceSS, mac = subMacs[rep(c(5, 1), each = 2)], angle = rep(c(0, 135),
    2), data = list(data = offlineSummary))
```




```
## $`00:14:bf:b1:97:90`
## NULL
##
## $`00:14:bf:b1:97:90`
## NULL
##
## $`00:0f:a3:39:e1:c0`
## NULL
##
## $`00:0f:a3:39:e1:c0`
## NULL
```

```
par(parCur)
```

This figure is telling us that the location of the access point is the dark red region at the top of the "hill". We can also see the effect of the orientation on signal strength. In addition we can also detect a corridor effect. The signal is greater relative to the distance along the corridors where the signals are not blocked by walls.

We know that the locations of the access points are based on the floor plan of the building. However, we have not been given the exact location and we don not know the mapping between MAC address and access point. Because of the contour maps that we created we can connect the MAC address to the access point marked on the floor plan. In the upper left plot we see that the dark red area corresponds to the access point in the original floor plan. Since the documentation states that the training data was measured at 1 meter intervals in the building we can estimate the location using those points as references.

To test this we are going to subset our data

```
offlineSummary <- subset(offlineSummary, mac != subMacs[2])
```

Now we create a matrix with the positions for the 6 access points on the floor plan.

```
AP <- matrix(c(7.5, 6.3, 2.5, -0.8, 12.8, -2.8, 1, 14, 33.5, 9.3, 33.5, 2.8),
    ncol = 2, byrow = TRUE, dimnames = list(subMacs[-2], c("x", "y")))

AP
```

```
##                       x    y
## 00:0f:a3:39:e1:c0  7.5  6.3
## 00:14:bf:b1:97:8a  2.5 -0.8
## 00:14:bf:3b:c7:c6 12.8 -2.8
## 00:14:bf:b1:97:90  1.0 14.0
## 00:14:bf:b1:97:8d 33.5  9.3
## 00:14:bf:b1:97:81 33.5  2.8
```

These row name are useful when indexing the data.

To examine the relationship between signal strength and distance from the access point, we need to compute the distance from the locations of the device emitting the signal to the access point receiving the signal. First, we compute the difference for the y coordinates.

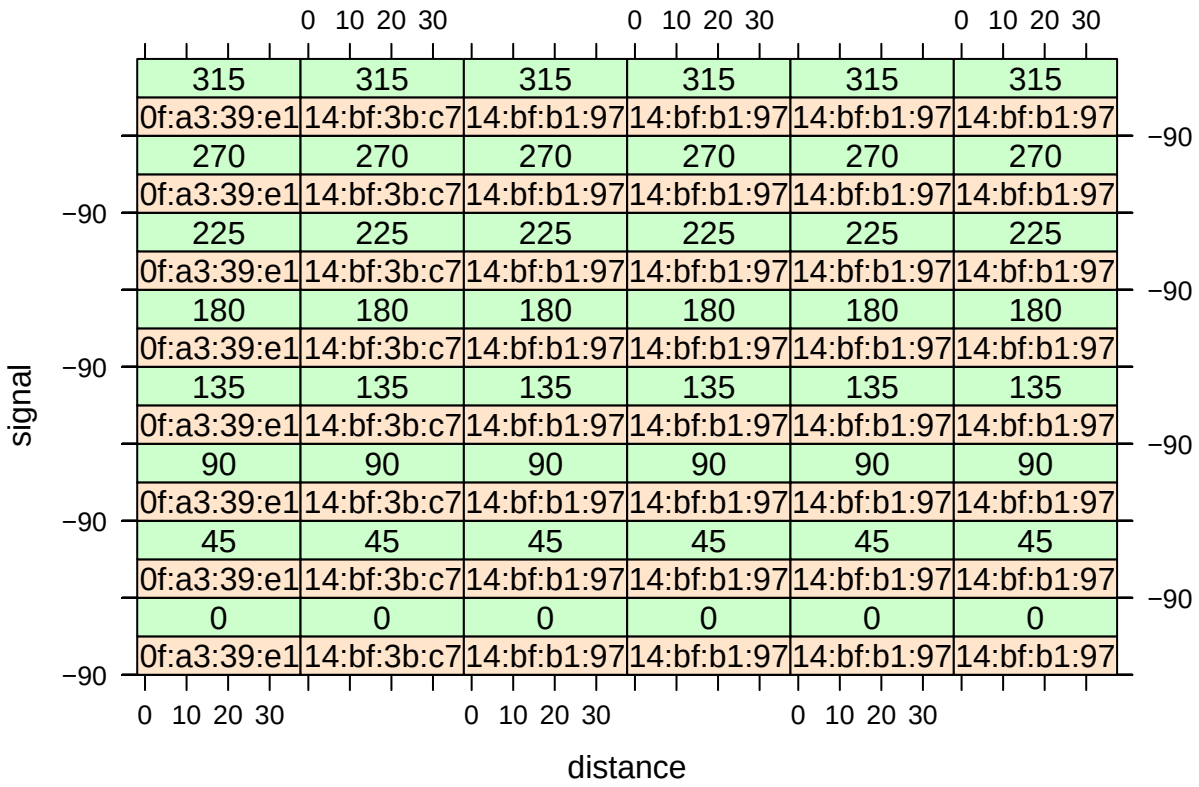
```
diffs <- offlineSummary[, c("posX", "posY")] - AP[offlineSummary$mac, ]
```

Now we use Euclidean distance between the position of the hand-held device and the access point with

```
offlineSummary$dist <- sqrt(diffs[, 1]^2 + diffs[, 2]^2)
```

Finally, we make a series of scatter plots for each access point and device orientation.

```
xyplot(signal ~ dist | factor(mac) + factor(angle), data = offlineSummary, pch = 19,
    cex = 0.3, xlab = "distance")
```




```
pdf(file = "Geo_ScatterSignalDist.pdf", width = 7, height = 10)
oldPar = par(mar = c(3.1, 3.1, 1, 1))
```

```
xyplot(signal ~ dist | factor(mac) + factor(angle), data = offlineSummary, pch = 19,
    cex = 0.3, xlab = "distance")
par(oldPar)
dev.off()
```

```
## pdf
##   2
```

The scatter plots appear to have curvature in the plots. A log transformation might improve the relationship. The values are negative so that would take some care.

Methods

Nearest Neighbor Methods to Predict Location

There are many statistical techniques that can be used to estimate the location of device from the strength of the signal. For this analysis we are going to use a common approach called k-nearest neighbors, or k-NN. This method takes training data where the signal is measured to signal is measured to several access points from known positions in a building. When we get a new observation, a new set of signal strengths for a location that is not known, we find the observation in our training data that is closest to this new observation. For k-nearest neighbors where k is larger than 1, then we find the k closest training points and estimate the new observations position by an aggregation of the positions of the k training points.

Preperation of the Test Data

The online data are in the online.final.trace.txt and these observations are our test data. We will use the readData() function from earlier to process the raw data.

```
macs <- unique(offlineSummary$mac)
online <- readData("http://rdatasciencecases.org/Data/online.final.trace.txt",
    subMacs = macs)
```

Create a unique location identifier with

```
online$posXY <- paste(online$posX, online$posY, online$posY, sep = "-")
# Check
length(unique(online$posXY))
```

```
## [1] 60
```

Check the number of signal strengths recorded at each location with

```
tabonlineXYA <- table(online$posXY, online$angle)
tabonlineXYA[1:6, ]
```

```
##
##                      0  45  90 135 180 225 270 315
##   0-0.05-0.05        0   0   0 593   0   0   0   0
##   0.15-9.42-9.42     0   0 606   0   0   0   0   0
##   0.31-11.09-11.09   0   0   0   0   0 573   0   0
##   0.47-8.2-8.2     590   0   0   0   0   0   0   0
##   0.78-10.94-10.94 586   0   0   0   0   0   0   0
##   0.93-11.69-11.69   0   0   0   0 583   0   0   0
```

The output would indicate that the signal strengths at one orientation for each location.

We will be computing distances between vectors of 6 signal strengths. We will organize the data in a different structure than we have used thus far. To organize the data we are going have 6 columns of signal strengths. We summarize the online data into the new format providing average signal strength at each location.

```
keepVars = c("posXY", "posX", "posY", "orientation", "angle")
byLoc = with(online, by(online, list(posXY), function(x) {
    ans = x[1, keepVars]
    avgSS = tapply(x$signal, x$mac, mean)
    y = matrix(avgSS, nrow = 1, ncol = 6, dimnames = list(ans$posXY, names(avgSS)))
    cbind(ans, y)
}))

onlineSummary = do.call("rbind", byLoc)

dim(onlineSummary)
```

```
## [1] 60 11
```

```
names(onlineSummary)
```

```
##  [1] "posXY"             "posX"              "posY"
##  [4] "orientation"       "angle"             "00:0f:a3:39:e1:c0"
##  [7] "00:14:bf:3b:c7:c6" "00:14:bf:b1:97:81" "00:14:bf:b1:97:8a"
## [10] "00:14:bf:b1:97:8d" "00:14:bf:b1:97:90"
```

Orientation Choice

In the k-NN model, our goals is to find records in our offline data (training set) that are simular in nature to our new observations. Earlier we saw that orientation can impact the signal strength. To accomplish this we might use all records with an orienation that is within a specified range of the new point's orientation. Our observations were spaced at 45 degree increments. We can use those number of neighboring angles to include from the training data. As an example, we can use one orientation then we include training data with angles that match the rounded orientation value of the new observation. In the case we want two orientation we would pick two multiples of 45 that flank the new observations orientation. So, for three we the closest 45 degree increment and one on either side of it and so on. You may get where I am going here, for m the number of angles and angleNewObs the angle of the new observations, our angles include from our training set. Here is how this looks in code:

```
m <- 3
angleNewObs <- 230
refs <- seq(0, by = 45, length = 8)
nearestAngle <- roundOrientation(angleNewObs)

if (m%%2 == 1) {
    angles = seq(-45 * (m - 1)/2, 45 * (m - 1)/2, length = m)
} else {
    m = m + 1
    angles = seq(-45 * (m - 1)/2, 45 * (m - 1)/2, length = m)
    if (sign(angleNewObs - nearestAngle) > -1)
        angles = angles[-1] else angles = angles[-m]
}

## m odd and even are handled seperatly. Map angles to refs Adjustments
angles <- angles + nearestAngle
angles[angles < 0] = angles[angles < 0] + 360
angles[angles > 360] = angles[angles > 360] - 360
```

Now that we have the desired angles we need to subset the observations.

```
offlineSubset <- offlineSummary[offlineSummary$angle %in% angles, ]
```

Now we aggregate the signal strengths from the angles and create a data strcture that is simular to onlineSummary.

```
reshapeSS <- function(data, varSignal = "signal", keepVars = c("posXY", "posX",
    "posY")) {
    byLocation <- with(data, by(data, list(posXY), function(x) {
        ans = x[1, keepVars]
        avgSS = tapply(x[, varSignal], x$mac, mean)
        y = matrix(avgSS, nrow = 1, ncol = 6, dimnames = list(ans$posXY, names(avgSS)))
        cbind(ans, y)
    }))

    newDataSS <- do.call("rbind", byLocation)
    return(newDataSS)
}
```

Summarize and reshape offlineSubset

```
trainSS <- reshapeSS(offlineSubset, varSignal = "avgSignal")
```

Reshape function with the same logic as above

```
trainSS = reshapeSS(offlineSubset, varSignal = "avgSignal")

selectTrain = function(angleNewObs, signals = NULL, m = 1) {
    # m is the number of angles to keep between 1 and 5
    refs = seq(0, by = 45, length = 8)
    nearestAngle = roundOrientation(angleNewObs)

    if (m%%2 == 1)
        angles = seq(-45 * (m - 1)/2, 45 * (m - 1)/2, length = m) else {
        m = m + 1
        angles = seq(-45 * (m - 1)/2, 45 * (m - 1)/2, length = m)
        if (sign(angleNewObs - nearestAngle) > -1)
            angles = angles[-1] else angles = angles[-m]
    }
    angles = angles + nearestAngle
    angles[angles < 0] = angles[angles < 0] + 360
    angles[angles > 360] = angles[angles > 360] - 360
    angles = sort(angles)

    offlineSubset = signals[signals$angle %in% angles, ]
    reshapeSS(offlineSubset, varSignal = "avgSignal")
}
```

```
train130 <- selectTrain(130, offlineSummary, m = 3)
head(train130)
```

```
##      posXY posX posY 00:0f:a3:39:e1:c0 00:14:bf:3b:c7:c6 00:14:bf:b1:97:81
## 0-0    0-0    0    0         -52.37243         -66.13039         -63.19262
## 0-1    0-1    0    1         -52.98182         -65.37177         -63.72941
## 0-10  0-10    0   10         -56.34184         -65.67238         -69.16041
## 0-11  0-11    0   11         -54.73420         -67.17593         -70.34538
```

```
## 0-12  0-12    0   12         -56.03030         -70.46493         -72.28758
## 0-13  0-13    0   13         -54.55152         -71.19211         -72.58496
##      00:14:bf:b1:97:8a 00:14:bf:b1:97:8d 00:14:bf:b1:97:90
## 0-0          -35.58063         -64.25411         -55.33780
## 0-1          -39.37649         -65.44867         -59.15328
## 0-10         -44.71545         -66.85781         -50.45502
## 0-11         -48.34689         -66.78383         -54.93054
## 0-12         -45.17264         -66.72696         -50.49886
## 0-13         -43.32784         -68.72616         -54.48160
```

```
# Should be 166
length(train130[[1]])
```

```
## [1] 166
```

Let's go find some Nearest Neighbors

Like the headline says, lets do some modeling. We now have our training data set so we can predict the location of our new point. We want to look at the distance in terms of signal strength from our training data to the new point. We will need to calculate the distance from the new point to all obseravations in the training set.

We are going to use a function to handle this. The parameters of this function are a numeric vector of six new signal strengths and the return value from selectTrain(). The function will return the locations of the training observations in order of closeness to the new observations.

We subset the ordered locations to estimate the location of the new observation. For some value of k of nearest neighbors, we can simply average the first k observations. This is where our est XY comes from.

We don't take simple averages, for example, we can use weights in the average that are inversely proportional to the distance in signal strength from the test observation. For this case we have to return the distance values from the findNN() function. This alternative approach allows to include the k points that are close, but to differentiate between them by how close they actually are from the new observation's signals. The weights might be:

$$\frac{1/d_i}{\sum_{i=1}^{k} 1/d_i}$$

For the i-th closest neighboring observations where d, is the distance from our new point to that nearest reference point. We may want to consider different metrics. We have used Euclidean distance, but we may want to try Manhattan distance. One could also use the medians and not average when combining neighbors predict (x,y). This would be useful if the values we are averaging are skewed.

Prediction is accomplished with our predXY function. To recap we have 3 functions, we find our neighbors, we estimate our neighbors and we predict where our x and y are going to be.

```
findNN <- function(newSignal, trainSubset) {
    diffs = apply(trainSubset[, 4:9], 1, function(x) x - newSignal)
    dists = apply(diffs, 2, function(x) sqrt(sum(x^2)))
    closest = order(dists)
    return(trainSubset[closest, 1:3])
}

predXY <- function(newSignals, newAngles, trainData, numAngles = 1, k = 3) {

    closeXY = list(length = nrow(newSignals))
```

```
    for (i in 1:nrow(newSignals)) {
        trainSS = selectTrain(newAngles[i], trainData, m = numAngles)
        closeXY[[i]] = findNN(newSignal = as.numeric(newSignals[i, ]), trainSS)
    }

    estXY <- lapply(closeXY, function(x) sapply(x[, 2:3], function(x) mean(x[1:k])))
    estXY = do.call("rbind", estXY)
    return(estXY)
}
```

We will begin the modeling process with three nearest neighbors and three orientations.

```
estXYk3 <- predXY(newSignals = onlineSummary[, 6:11], newAngles = onlineSummary[,
    4], offlineSummary, numAngles = 3, k = 3)

estXYk1 <- predXY(newSignals = onlineSummary[, 6:11], newAngles = onlineSummary[,
    4], offlineSummary, numAngles = 3, k = 1)

calcError <- function(estXY, actualXY) sum(rowSums((estXY - actualXY)^2))

## Apply the functions to our two sets
actualXY <- onlineSummary[, c("posX", "posY")]
sapply(list(estXYk1, estXYk3), calcError, actualXY)
```

```
## [1] 659.4003 306.7025
```

What is this telling us? Well, not much from our numbers, we need to visualize the information to gain any intellegence out of our models.

```
floorErrorMap <- function(estXY, actualXY, trainPoints = NULL, AP = NULL) {

    plot(0, 0, xlim = c(0, 35), ylim = c(-3, 15), type = "n", xlab = "", ylab = "",
        axes = FALSE)
    box()
    if (!is.null(AP))
        points(AP, pch = 15)
    if (!is.null(trainPoints))
        points(trainPoints, pch = 19, col = "grey", cex = 0.6)

    points(x = actualXY[, 1], y = actualXY[, 2], pch = 19, cex = 0.8)
    points(x = estXY[, 1], y = estXY[, 2], pch = 8, cex = 0.8)
    segments(x0 = estXY[, 1], y0 = estXY[, 2], x1 = actualXY[, 1], y1 = actualXY[,
        2], lwd = 2, col = "red")
}

trainPoints <- offlineSummary[offlineSummary$angle == 0 & offlineSummary$mac ==
    "00:0f:a3:39:e1:c0", c("posX", "posY")]

pdf(file = "GEO_FloorPlanK3Errors.pdf", width = 10, height = 7)
oldPar = par(mar = c(1, 1, 1, 1))
floorErrorMap(estXYk3, onlineSummary[, c("posX", "posY")], trainPoints = trainPoints,
    AP = AP)
par(oldPar)
dev.off()
```

```
## pdf
##   2
```

```
pdf(file = "GEO_FloorPlanK1Errors.pdf", width = 10, height = 7)
oldPar = par(mar = c(1, 1, 1, 1))
floorErrorMap(estXYk1, onlineSummary[, c("posX", "posY")], trainPoints = trainPoints,
    AP = AP)
par(oldPar)
dev.off()
```

```
## pdf
##   2
```

If we look at the figures in Figure 4 and 5. We can see that our three nearest neighbors (Figure 5) gave us some tighter looking clusters than our single nearest neighbor did(Figure 4). There is a valid question from this and that is that another number nearest neighbors may yield better results.

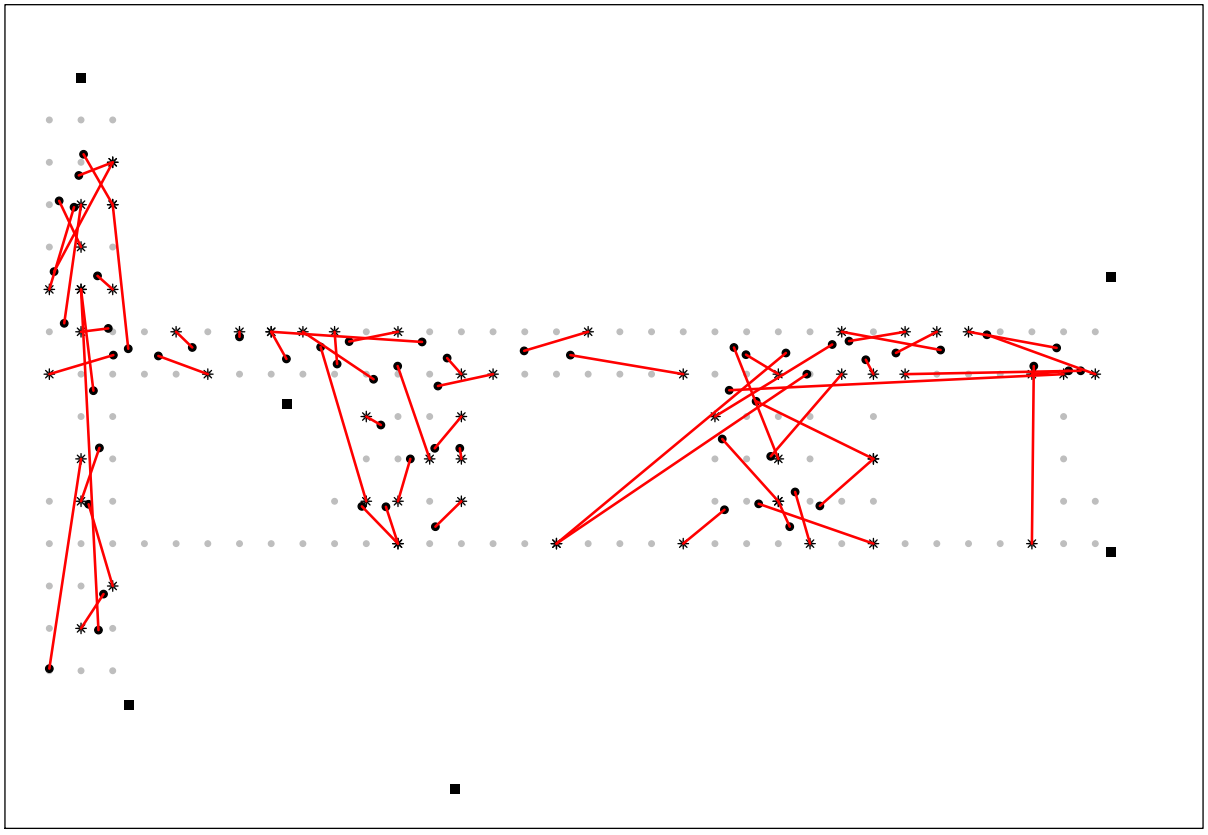


Figure 4: Figure 4: Floor plan with single NN showing Predicted and Actual Locations. *The red line segments connect the test locations in BLACK dots to their predicted locations ASTERISKS.*

Cross-Validation

The choice of k, the number of neighbors to include in the estimate of a new observation's position is at it's core a model selection problem. In a perfect world we want to choose the value of k independent of our test data so that we do not overfit the model to the training data. The method of v-fold cross-validation can help us to not overfit. The idea behind this tells us to divide our training data into v non-overlapping subsets of

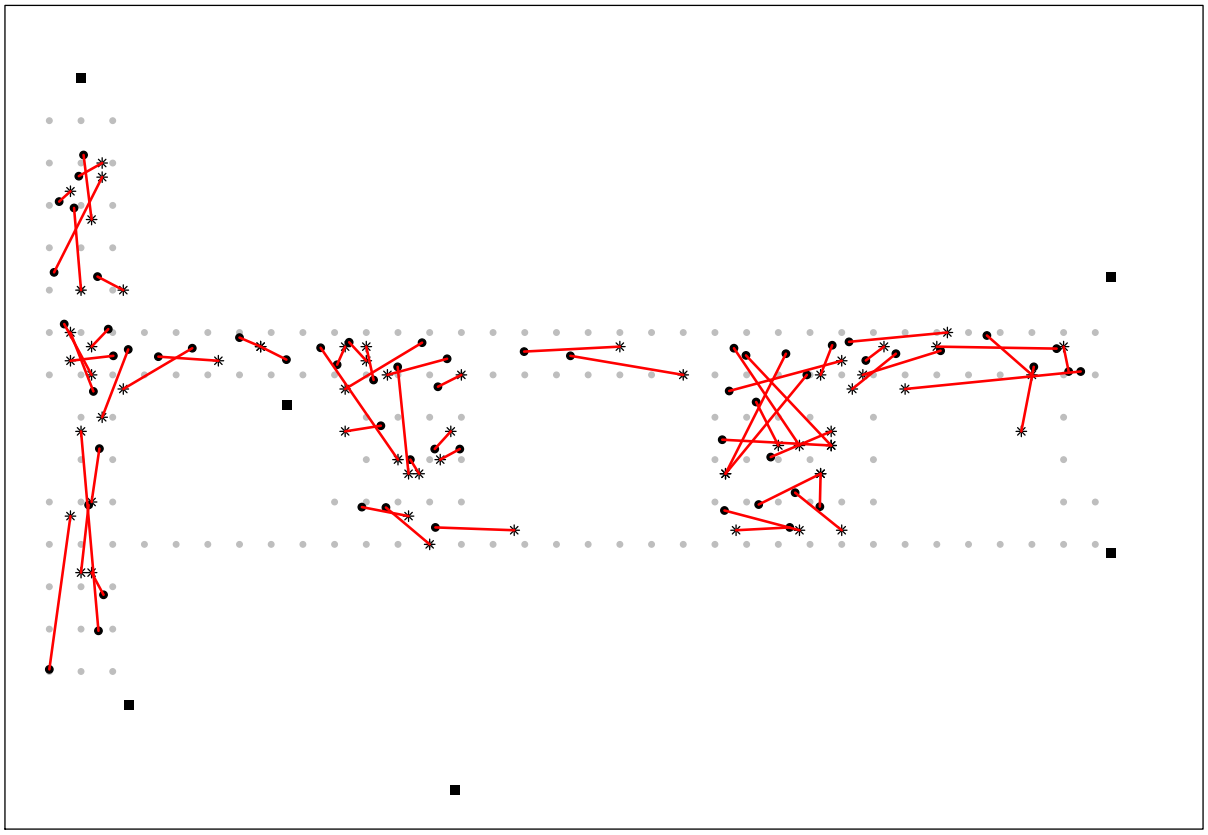


Figure 5: Figure 5: Floor plan with 3 NN's showing Predicted and Actual Locations. *The red line segments connect the test locations in BLACK dots to their predicted locations ASTERISKS.*

equal size. Then, for each subset, we build models with the data that are not in that subset. From here we assess the predictive ability of the model using the subset that was left out. This is then repeated on the model fitting and assessment for each of the v folds and aggregate the prediction errors across the folds.

There exist software that automatically does this and offers several options for cross-validation in the model building process. Some programs offer sugguestions based on your data. However, at the end of the day the analyst or data scientist must be able to use their judgement to find the best method based on the factors, and question at hand.

In our nearest neighbor scenario, we will use all eight orientations and six MAC addresses with each location. We will cross-validate on the 166 locations. Suppose that we take v = 11; so each fold has floor(166/v), or 15 locations. We can randomly select these locations with:

```
v <- 11
permuteLocs <- sample(unique(offlineSummary$posXY))
permuteLocs <- matrix(permuteLocs, ncol = v, nrow = floor(length(permuteLocs)/v))
```

```
## Warning in matrix(permuteLocs, ncol = v, nrow = floor(length(permuteLocs)/
## v)): data length [166] is not a sub-multiple or multiple of the number of
## rows [15]
```

```
## This will generate a warning message with the call to matrix() because v
## does not divide evenly into 166, so permuteLocs does not contain all 166
## locations. Each subset of 1s5 locations is used as the 'online' data.
onlineFold <- subset(offlineSubset, posXY %in% permuteLocs[, 1])
```

We need to summarize these data so that the data structure matches that of onlineSummary. We will need to select an orientation at random. This needs to be done so each observation has only one orientation.

```
reshapeSS <- function(data, varSignal = "signal", keepVars = c("posXY", "posX",
    "posY"), sampleAngle = FALSE, refs = seq(0, 315, by = 45)) {
    byLocation = with(data, by(data, list(posXY), function(x) {
        if (sampleAngle) {
            x = x[x$angle == sample(refs, size = 1), ]
        }
        ans = x[1, keepVars]
        avgSS = tapply(x[, varSignal], x$mac, mean)
        y = matrix(avgSS, nrow = 1, ncol = 6, dimnames = list(ans$posXY, names(avgSS)))
        cbind(ans, y)
    }))

    newDataSS = do.call("rbind", byLocation)
    return(newDataSS)
}
```

Summarize and format offline with

```
offline <- offline[offline$mac != "00:0f:a3:39:dd:cd", ]

keepVars <- c("posXY", "posX", "posY", "orientation", "angle")

onlineCVSummary <- reshapeSS(offline, keepVars = keepVars, sampleAngle = TRUE)
```

Fold Design

In each fold, we want to find the k-NN estimates for k = 1,2...,K, for some large K. Futhermore, we want to aggregate the errors over the folds. This is done by wrapping our code in loops over the folds and the

number of the neighbors. Our K in this chunk will be K = 20.

```
# This chunk of code takes a few minutes
onlineFold <- subset(onlineCVSummary, posXY %in% permuteLocs[, 1])

offlineFold <- subset(offlineSummary, posXY %in% permuteLocs[, -1])

estFold <- predXY(newSignals = onlineFold[, 6:11], newAngles = onlineFold[,
    4], offlineFold, numAngles = 3, k = 3)

actualFold <- onlineFold[, c("posX", "posY")]
calcError(estFold, actualFold)
```

```
## [1] 186.7778
```

```
K <- 20
err <- rep(0, K)

for (j in 1:v) {
    onlineFold = subset(onlineCVSummary, posXY %in% permuteLocs[, j])
    offlineFold = subset(offlineSummary, posXY %in% permuteLocs[, -j])
    actualFold = onlineFold[, c("posX", "posY")]

    for (k in 1:K) {
        estFold = predXY(newSignals = onlineFold[, 6:11], newAngles = onlineFold[,
            4], offlineFold, numAngles = 3, k = k)
        err[k] = err[k] + calcError(estFold, actualFold)
    }
}
```

Visualization of the journey:

```
pdf(file = "Geo_CVChoiceOfK.pdf", width = 10, height = 6)
oldPar = par(mar = c(4, 3, 1, 1))
plot(y = err, x = (1:K), type = "l", lwd = 2, ylim = c(1200, 2100), xlab = "Number of Neighbors",
    ylab = "Sum of Square Errors")

rmseMin <- min(err)
kMin <- which(err == rmseMin)[1]
segments(x0 = 0, x1 = kMin, y0 = rmseMin, col = gray(0.4), lty = 2, lwd = 2)
segments(x0 = kMin, x1 = kMin, y0 = 1100, y1 = rmseMin, col = grey(0.4), lty = 2,
    lwd = 2)

# mtext(kMin, side = 1, line = 1, at = kMin, col = grey(0.4))
text(x = kMin - 2, y = rmseMin + 40, label = as.character(round(rmseMin)), col = grey(0.4))
par(oldPar)
dev.off()
```

```
## pdf
##   2
```

What are looking at in Figure 6 is the sum of squared errors as a function of k. We can see that the errors decrease a lot at first and level out around five. After which, the errors start to increase slowly because the neighbors become too spread out geographically.

We will use the value of five for the nearest neighbors that we obtained from cross-validation, and we apply it to our training data to test.

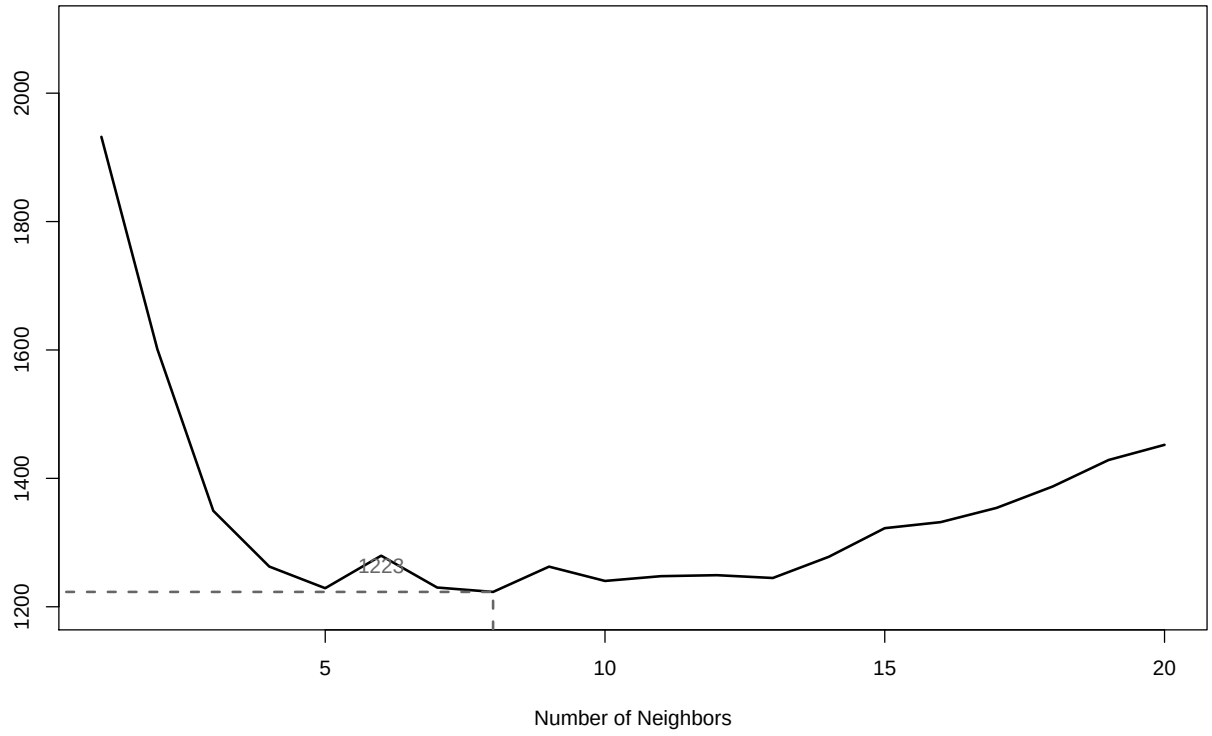


Figure 6: Figure 6: Cross Validation Selction of k. *The line plot shows the sum of squared errors as a function of number of neighbors used in predicting the location of a new observation. The sum of squared errors are obtained with cross-validation of the offline data.*

```
estXYk5 <- predXY(newSignals = onlineSummary[, 6:11], newAngles = onlineSummary[,
    4], offlineSummary, numAngles = 3, k = 5)
# Add up errors in prediction
calcError(estXYk5, actualXY)
```

```
## [1] 275.5083
```

The results with k = 1 and k = 3 were 659 and 307, respectiviely. The choice of k = 5 may not be the optimized value for our online data because the value was chosen without reference to the online data.

Possible faster code

```
predXY <- function(newSignals, newAngles, trainData, numAngles = 1, k = 3) {

    closeXY = list(length = nrow(newSignals))

    for (i in 1:nrow(newSignals)) {
        trainSS = selectTrain(newAngles[i], trainData, m = numAngles)
        closeXY[[i]] = findNN(newSignal = as.numeric(newSignals[i, ]), trainSS)
    }

    estXY = lapply(closeXY, function(x) sapply(x[, 2:3], function(x) mean(x[1:k])))
    estXY = do.call("rbind", estXY)
    return(estXY)
}
```