## **Introduction**

Indoor positioning systems utilize a system of sensors and tags to track and location items and people moving through an indoor space. This is similar in concept to global positioning systems, where satellite signals are used for communication. Because satellite signals are not reliable when inside of buildings, these systems must instead employ widely-available local area networks (LANs) to send and receive informational signals via WiFi between access points and the items of interest. [1] Tracking materials in warehouses and equipment in hospitals, as well as serving up the menu of the nearest café on a large corporate campus, are all applications that employ indoor positioning systems.

Indoor positioning systems require a reference map to indicate position of access points and the signals each receives from a typical tag, such as a cell phone or a laptop, at a given location. This reference map is built through repeated measurements from each access point for multiple known tag locations. Signals sent from tags at unknown locations can then be compared against the reference map to determine a predicted location. [1]

*k*-nearest neighbors (*k-*NN) is an algorithm that uses distance measurements to compare similarity of known data points to an unknown data point. The *k* indicates the number of known data points to find that are closest to the unknown point. Methods of measuring distance can vary, as can the method of combining the values from the nearest neighbors to predict the unknown values. [1]

In this study, we will predict the location of an unknown device based on the reference map and *k-*NN algorithm variations, and evaluate the efficacy of the map based on the use of a subset of the location information available from the data set.

## **Background**

In the earliest research of using WiFi signals for indoor positioning systems, researchers at University of Mannheim generated the dataset that was used in this case study. A grid with reference points spaced 1.0 meters apart was used as the basis for building the reference map. The reference map, termed the “offline” dataset in this study, was built by taking measurements at each of these reference points. [2]

For each measurement, the (*x,y*) coordinates of the grid are noted, as is the orientation of the device. The orientation of the device refers to how the device was oriented to the access point. Capturing this allows for the fact that there is some variability in signal strength when the human tester’s body was in between a given access point and the device. The measured values in each of these measurements consisted of the signal strength to the access points. Multiple replications were collected at each grid reference point; 110 measurements was the target value, and in some instances, the values were not recorded. [2]

With the reference “map” dataset created, the data for testing was collected in a similar fashion. Sixty locations and orientations were chosen at random. The locations and orientations did not generally correspond to an exact grid point from the reference map and are instead randomly spread between the reference points. Signal strength measurements to each access point were measured in the same fashion as the reference points: 110 replications in the given location and orientation, with some of the measurements not getting recorded. [2]

Each measurement consisted of a timestamp, an id that corresponds to the MAC (media access control) of the scanning device, the real position, the orientation, and then a set of values for each access point: its MAC, the signal strength, the frequency and the mode. There were some measurements that picked up MAC addresses for devices that were not of the access point mode, and those were removed from the analysis. This is largely because there wouldn’t be enough of them to significantly contribute to the findings of the experiment. The initial pass through the data also removed instances where the measurements were not recorded. [1]

The original analysis laid out in the Nolan and Lang book made some assumptions about which data to use, in particular, the subset of MAC addresses that corresponded to valid access points. There were twelve (12) access point MAC addresses in the dataset, and eight (8) different channels (frequencies). Since the mapping was not one-to-one based on the experimental design, which indicated six (6) access points, decisions about what to keep needed to be made. The documentation on the MAC addresses based on make and model of the access point hardware was not accurate, based on manufacturer details. After initial analysis of the signal strength, it was determined that two of the access points appeared to represent the same location on the map, and the authors chose to keep the first one (00:0f:a3:39:e1:c0) and drop the second one (00:0f:a3:39:dd:cd). [1] We evaluate the effects of that decision in our analysis.

A *k-*NN algorithm is used to predict where the test data points in the testing dataset are located. Our initial analysis focuses on the effects of the selection of MAC addresses to drop or keep using an unweighted *k-*NN method, meaning that the distance to the nearest neighbors is not taken into account. We also evaluate a weighted *k*-NN method, which relies on the weighting the impact of each neighbor on the predicted value based on the distance it is from the data point being predicted. In our weighting scheme, we take the weighting to be (1/*k*), where the weight for each corresponding neighbor is weighted by the inverse of how many neighbors it is away from the data point in question: the first nearest neighbor has a weight of 1, the second nearest neighbor has a weight of (1/2=) 0.5, etc. We looked at values of *k* ranging from 1 to 10.

## **Method**

The methods used in this case study are based on the analysis outlined in Case Studies in Data Science in R, Chapter 1. The questions of interest in this case study were whether the decision to exclude the originally excluded MAC impacted the *k-*NN predictions of the test locations, and whether or not a weighted *k*-NN was a more useful method of analysis.

The data cleaning approach in our analysis closely follows the approach given by the book, with several changes required to implement valid code. Updating the functions to accept all seven (7) MAC addresses was an important key to running this analysis. Beyond the initial code clean-up, the weighted *k*-NN needed to be defined. We found the k-NN algorithm code from the case to have errors and chose to use a k-NN package called RANN, which allowed us to perform basic k-NN by averaging the positions of the nearest neighbors. [3] In order to perform weighted k-NN we were able to use the distance measurement output from the RANN k-NN algorithm with our own code for the fullest control of the behavior. The final weights for each nearest neighbor are defined the same as they were defined in the book [1].

The authors address the requirement to align the orientation of the new device observation with the similar orientations present in the training data, as clearly signal strengths from the same location vary with the orientation of the device. The authors provided an approach to subset the training data to a variable number of angles from the new observations being analyzed. We chose to use 3 orientations throughout our analysis, which means we are subsetting the training data to only include 3 orientations: the exact orientation of the new observation and each orientation to either side in 45 degree increments. We believe this will provide additional accuracy versus using only the exact orientation of the new observation in the training data.

The test data includes information about the actual position of the test point, thus it was possible to calculate the error between the predicted value and the actual value for each variation of the trained *k*-NN. In order to address the first question of interest we performed k-NN on 3 different versions of the training data. The first version excluded the MAC address 00:0f:a3:39:dd:cd, which was the same approach taken by the authors of the case. For comparison, we performed analysis excluding 00:0f:a3:39:dd:cd, but including 00:0f:a3:39:e1:c0, which the authors had removed. Finally, we performed analysis using all of the MAC addresses (7). For each of these we performed the k-NN error analysis at values of k equal to 1, 3, and 5 (Table 1).

**Table 1. Error Comparisons for Varied MAC Address Training Subsets at k=1,3, 5**

|  |  |
| --- | --- |
| ***k*-NN scenario (k=1)** | **Error** |
| Exclude MAC = 00:0f:a3:39:dd:cd  (Original textbook solution) | 659.4 |
| Exclude MAC = 00:0f:a3:39:e1:c0 | 411.6 |
| Include all 7 MAC addresses | 478.6 |

|  |  |
| --- | --- |
| ***k*-NN scenario (k=3)** | **Error** |
| Exclude MAC = 00:0f:a3:39:dd:cd  (Original textbook solution) | 306.7 |
| Exclude MAC = 00:0f:a3:39:e1:c0 | 270.5 |
| Include all 7 MAC addresses | 249.4 |

|  |  |
| --- | --- |
| ***k*-NN scenario (k=5)** | **Error** |
| Exclude MAC = 00:0f:a3:39:dd:cd  (Original textbook solution) | 275.5 |
| Exclude MAC = 00:0f:a3:39:e1:c0 | 249.9 |
| Include all 7 MAC addresses | 209.6 |

When working with the weighted *k-*NN method, a range of values of *k* were used to test. Similar measures of error were calculated (Table 2).

**Table 2. Error Comparisons for Values of *k* in Weighted *k-*NN**

|  |  |  |
| --- | --- | --- |
| ***k*** | **Error**  *Raw Avg* | **Error**  *Weighted Avg* |
| 1 | 478.64 | 478.64 |
| 2 | 250.86 | 256.40 |
| 3 | 249.42 | 249.97 |
| 4 | 230.32 | 229.37 |
| 5 | 209.61 | 209.24 |
| 6 | 211.17 | 207.15 |
| 7 | 221.44 | 214.86 |
| 8 | 221.74 | 215.38 |
| 9 | 217.18 | 211.81 |
| 10 | 222.16 | 215.84 |

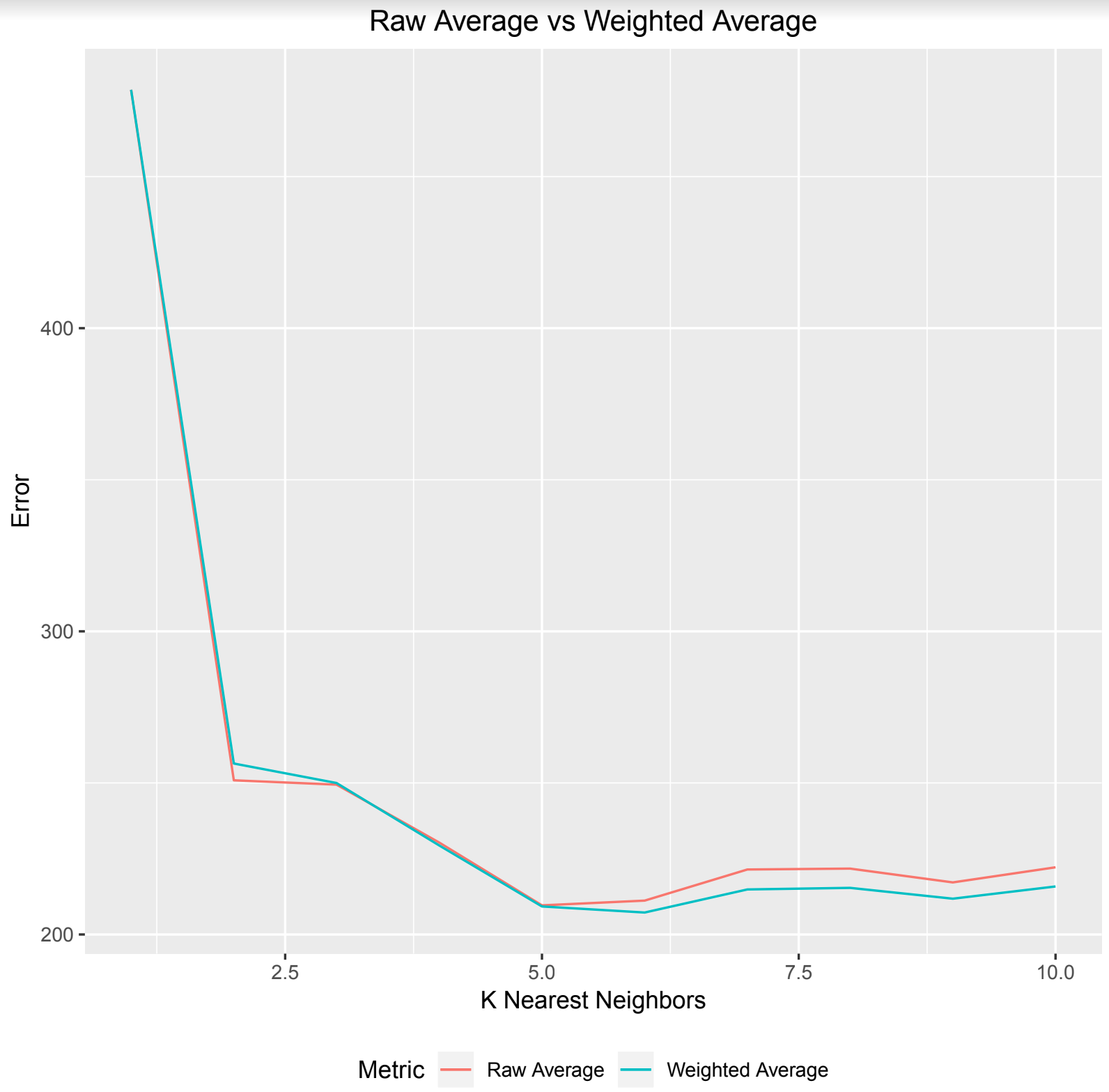
## **Results**

Based on our analysis in Table 1 it is clear that the authors of the case study removed the wrong MAC address from their training data as this decision made an impact on their prediction accuracy. For all cases of k that we analyzed we exhibit less error when MAC 00:0f:a3:39:dd:cd was included in the training data in place of MAC 00:0f:a3:39:e1:c0. It also becomes clear that using all 7 MAC addresses improves accuracy starting at k equal to 3, but improving even more with k equal to 5. For this reason we decide to proceed with the weighted k-NN analysis using all 7 MAC addresses in the training data.

The result shown in Table 2 above as well as in Figure 1 below indicate that the weighted *k*-NN measures begin to perform better than the unweighted average once we begin to use *k* greater than 5. However, the difference in minimal and the separation becomes clear once we use a *k* = 6 or higher.

Additionally, Figure 1 indicates that we see our best performance for out model using *k*=6 and the weighted average for estimating the location. However, there is noticeable plateau at *k*=5 and the performance gains of an additional neighbor may not warrant the extra compute costs. That decision would be reliant on the performance requirements of the system.

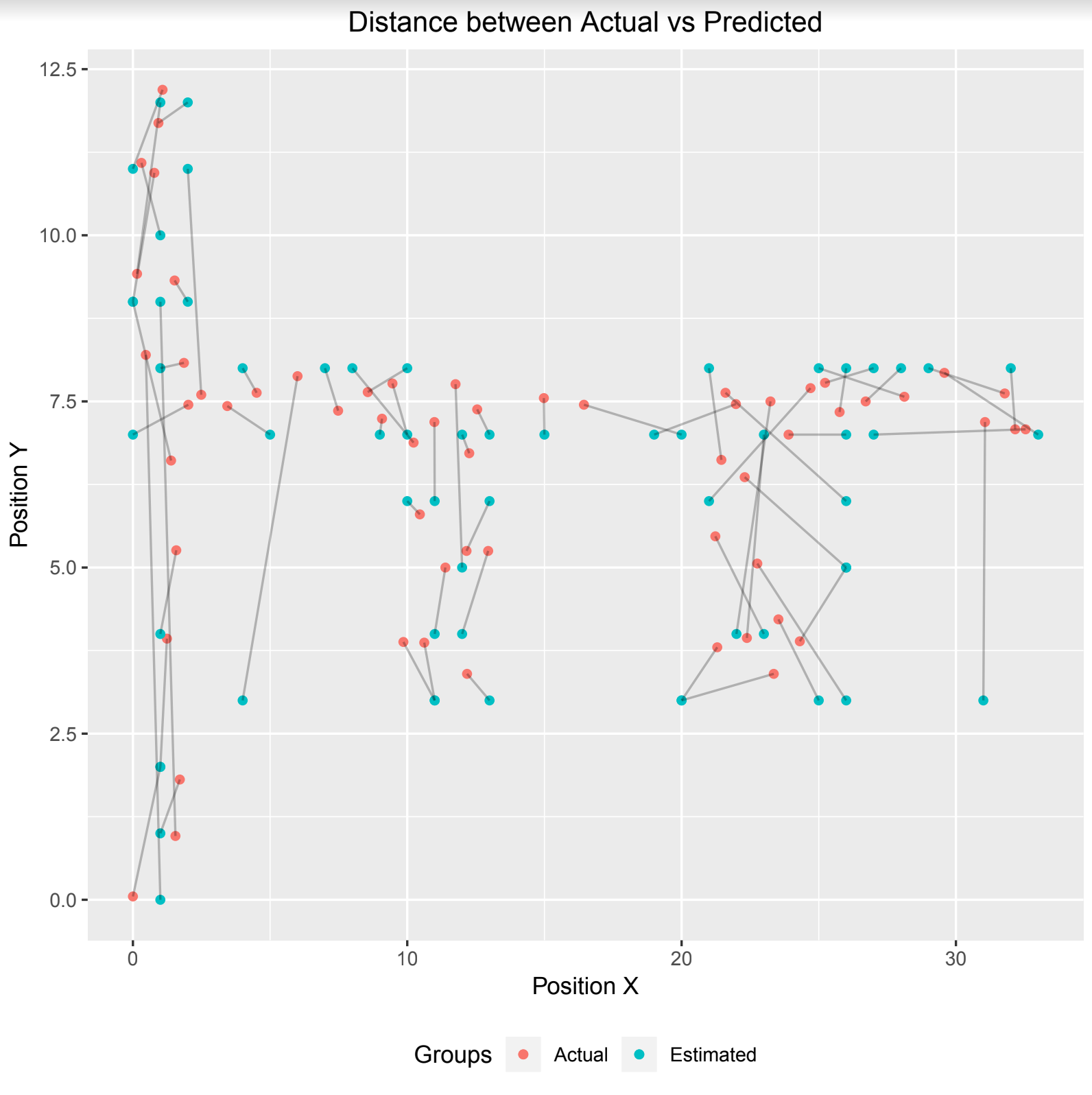
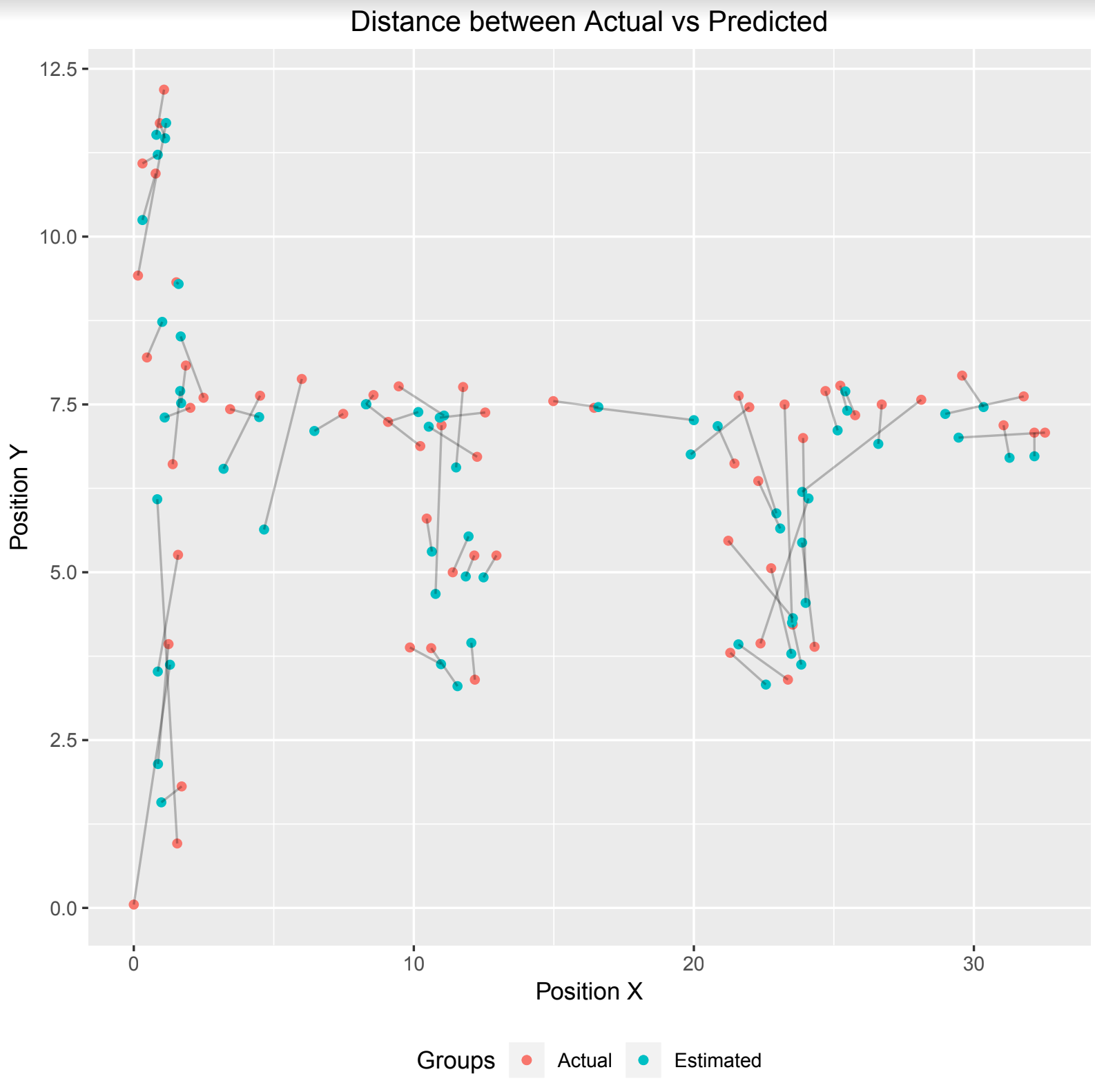
**Figure 1. Comparison of error between weighted and un-weighted averages**



To further validate some of the findings above we also wanted to visualize the distances between actual and predicted to see if changes are noticeable. Figure 2 shows the comparison of distances between the worst performing model (k=1) and the best performing (k=6) for our 60 online observations.

**Figure 2. Distances between predicted and actual locations**

***K* = 1 *K* = 6**

From the above you are able to see how much shorter the error in our RTLS predictions are between a single nearest neighbor and when the values of 6 closest are averaged.

## **Conclusions**

In conclusion, our analysis demonstrates that the authors of the case study should have taken more care when deciding which of the seemingly redundant MAC addresses reporting from the center of the building to eliminate from their training data. Our analysis not only proves that the alternative MAC address has better prediction accuracy, but that both MAC addresses included together provide the best prediction in most of our testing (except for k=1, which for most data is usually inaccurate).

We also demonstrate that using a weighted average to estimate the location of the device does not perform significantly better than the simple average for these data. We did see a slight improvement in accuracy for k=6 with the weighted average calculation, but as mentioned it might not be significant given the additional compute cost.

Upon observation of the differences between the estimated and actual positions of our best and worst performing models (Figure 2) it is clear that this data can be used for location prediction. The figure of our best performing model begins to resemble the floorplan of the space the observations were taken, which is reinforced by the appearance of the two open spaces that are central to the floorplan.

## **References**

1. Nolan, D. and Lang, D. T. “Data Science in R.” CRC Press, 2015 (Chapter 1).
2. King, Thomas & Kopf, Stephan & Haenselmann, Thomas & Lubberger, Christian & Effelsberg, Wolfgang. (2006). COMPASS: A Probabilistic Indoor Positioning System Based on 802.11 and Digital Compasses. 34-40. 10.1145/1160987.1160995.
3. Arya, Sunil & Mount, David, & Kemp, Samuel & Jefferis, Gregory. (2008). RANN: Fast Nearest Neighbor Search (Wraps ANN Library). R package version 2.6.1.

## **Appendix - R Code**

#Functions needed by readData

# split each line on ; = , and return 1 line each for each observed signal (redundant time, mac, x, y, z, orientation)

processLine =

function(x)

{

tokens = strsplit(x, "[;=,]")[[1]]

if (length(tokens) == 10) #added to fix those observations where no signal detected

return(NULL)

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)

}

# set angles to proper increments

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]

}

#Main data load function for first format

readData = function(filename){

# load data, drop all comment rows (beginning with '#')

txt = readLines(filename)

lines = txt[ substr(txt, 1, 1) != "#" ] # remove lines that start with #, comments

# process each line via process line function, combine all matrices that are returned

tmp = lapply(lines, processLine)

output = as.data.frame(do.call("rbind", tmp), stringsAsFactors = FALSE)

# set column names and numeric data types

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

numVars = c("time", "posX", "posY", "posZ", "orientation", "signal")

output[ numVars ] = lapply(output[ numVars ], as.numeric)

# remove rows that aren't type == 3, drop the type column

output = output[ output$type == "3", ]

output = output[ , "type" != names(output) ]

# copy raw time, convert time from ms, set proper time data types

output$rawTime = output$time

output$time = output$time/1000

class(output$time) = c("POSIXt", "POSIXct")

# convert character variables to factors

sapply(output[ , c("mac", "channel", "scanMac")], as.factor)

# drop scanmac and posZ

output = output[ , !(names(output) %in% c("scanMac", "posZ"))]

# set angles to proper rounded increments

output$angle = roundOrientation(output$orientation)

# drop all rows that aren't in the top 7 MACs by count...(this is the subMacs component from the book)

subMacs = names(sort(table(output$mac), decreasing = TRUE))[1:7]

output = output[ output$mac %in% subMacs, ]

return(output)

}

# load offline data

offline = readData("offline.final.txt")

# code to create offline summary

processOfflineSummary = function(offline){

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

byLocAngleAP = with(offline,

by(offline, list(posXY, angle, mac), function(x) x))

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

})

return(do.call("rbind", signalSummary))

}

# create offline summary

offlineSummary = processOfflineSummary(offline)

# load online data

online = readData("online.final.txt")

# code to create online summary

processOnlineSummary = function(online){

online$posXY = paste(online$posX, online$posY, sep="-") # create new feature combining posX with posY for each row

online$posXY <- factor(online$posXY) #let's make posXY a factor THIS IS WHAT WAS MISSING IN THE BOOK!

keepVars = c("posXY", "posX","posY", "orientation", "angle")

byLoc = with(online, # with applies an expression to the dataset "with(data, expression)", in this case applies "by"

by(online, list(posXY), # by applies a function to each level of a factor "by(data, factorlist, function)"

function(x) {

ans = x[1, keepVars]

avgSS = tapply(x$signal, x$mac, mean)

y = matrix(avgSS, nrow = 1, ncol = 7, dimnames = list(ans$posXY, names(avgSS))) # THIS NEEDS TO BE 7 COL, NOT 6 LIKE THE BOOK..BECAUSE WE HAVEN'T DROPPED EXTRA MAC YET

cbind(ans, y)

}))

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

}

# create online summary

onlineSummary = processOnlineSummary(online)

# code to take data shaped like the online summary, and make it the format of the offline summary

reshapeSS = function(data, varSignal = "signal", keepVars = c("posXY", "posX","posY")) {

data$posXY = factor(data$posXY) ## added this here as well, since it needs to be a factor for the by function below

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)

}

# function to reduce the training data by range of angles specified

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, ]

return(reshapeSS(offlineSubset, varSignal = "avgSignal"))

}

# function to calculate total error between estimate and actual x and y

calcError = function(estXY, actualXY){

sum(rowSums((estXY - actualXY)^2))

}

# function to get index and distances of k nearest neighbors using RANN library

library("RANN")

findNN = function(newSignal, trainSubset, k = 3) {

knn <- nn2(trainSubset[,4:9], newSignal, k = k)

nn <- trainSubset[knn$nn.idx , ]

nn$dist <- (array(knn$nn.dists))

return(nn)

}

# function to predict X and Y of new observation (uses the average of the specified nearest neighbors)

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)

tmp = findNN(newSignal = newSignals[i, ], trainSS, k=k)

estXY = colMeans(tmp[sapply(tmp[1:3], is.numeric)])[1:2]

closeXY[[i]] = estXY

}

return(do.call(rbind, closeXY))

}

# create actual X + Y for comparison in calcError

actualXY = onlineSummary[ , c("posX", "posY")]

# this runs excluding 00:0f:a3:39:dd:cd per the book

offlineSummary1 = offlineSummary[offlineSummary$mac != '00:0f:a3:39:dd:cd',] # drop rows containing the extra MAC

onlineSummary1 = onlineSummary[, !(names(onlineSummary) %in% '00:0f:a3:39:dd:cd')] # drop column containing the extra MAC

estXYk5 = predXY(newSignals = onlineSummary1[ , 6:11] , newAngles = onlineSummary1[ , 4], offlineSummary1, numAngles = 3, k = 5)

estXYk3 = predXY(newSignals = onlineSummary1[ , 6:11] , newAngles = onlineSummary1[ , 4], offlineSummary1, numAngles = 3, k = 3)

estXYk1 = predXY(newSignals = onlineSummary1[ , 6:11] , newAngles = onlineSummary1[ , 4], offlineSummary1, numAngles = 3, k = 1)

sapply(list(estXYk1, estXYk3, estXYk5), calcError, actualXY)

# this runs including 00:0f:a3:39:dd:cd, but excluding 00:0f:a3:39:e1:c0 (the one the book decided to keep)

offlineSummary2 = offlineSummary[offlineSummary$mac != '00:0f:a3:39:e1:c0',] # drop rows containing the extra MAC

onlineSummary2 = onlineSummary[, !(names(onlineSummary) %in% '00:0f:a3:39:e1:c0')] # drop column containing the extra MAC

estXYk5 = predXY(newSignals = onlineSummary2[ , 6:11] , newAngles = onlineSummary2[ , 4], offlineSummary2, numAngles = 3, k = 5)

estXYk3 = predXY(newSignals = onlineSummary2[ , 6:11] , newAngles = onlineSummary2[ , 4], offlineSummary2, numAngles = 3, k = 3)

estXYk1 = predXY(newSignals = onlineSummary2[ , 6:11] , newAngles = onlineSummary2[ , 4], offlineSummary2, numAngles = 3, k = 1)

sapply(list(estXYk1, estXYk3, estXYk5), calcError, actualXY)

# this runs including 00:0f:a3:39:dd:cd AND 00:0f:a3:39:e1:c0 (both MAC addresses)

# have to update reshape to accomodate 7 MACs versus 6

reshapeSS = function(data, varSignal = "signal", keepVars = c("posXY", "posX","posY")) {

data$posXY = factor(data$posXY) ## added this here as well, since it needs to be a factor for the by function below

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 = 7, dimnames = list(ans$posXY, names(avgSS)))

cbind(ans, y)

}))

newDataSS = do.call("rbind", byLocation)

return(newDataSS)

}

# have to update findNN() to accept 7 MACs in the training subset

findNN = function(newSignal, trainSubset, k = 3) {

knn <- nn2(trainSubset[,4:10], newSignal, k = k)

nn <- trainSubset[knn$nn.idx , ]

nn$dist <- (array(knn$nn.dists))

return(nn)

}

# have to update index in onlineSummary to include 7 MACs

estXYk5 = predXY(newSignals = onlineSummary[ , 6:12] , newAngles = onlineSummary[ , 4], offlineSummary, numAngles = 3, k = 5)

estXYk3 = predXY(newSignals = onlineSummary[ , 6:12] , newAngles = onlineSummary[ , 4], offlineSummary, numAngles = 3, k = 3)

estXYk1 = predXY(newSignals = onlineSummary[ , 6:12] , newAngles = onlineSummary[ , 4], offlineSummary, numAngles = 3, k = 1)

sapply(list(estXYk1, estXYk3, estXYk5), calcError, actualXY)

# function to predict X and Y of new observation (uses the WEIGHTED average of the specified nearest neighbors)

weightedPredXY = 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)

tmp = findNN(newSignal = newSignals[i, ], trainSS, k=k)

estXY = colSums(tmp[,c('weightX','weightY')])

names(estXY) = c('posX','posY')

closeXY[[i]] = estXY

}

return(do.call(rbind, closeXY))

}

## plots of actual vs predicted with distance line segments

plotDistance = function(compare){

p <- ggplot(compare, aes(x=posX, y=posY)) +

geom\_point(aes(colour = 'Actual' )) +

geom\_point(aes(x=estX, y=estY, colour='Estimated')) +

geom\_segment(data = compare,

aes(x = posX, xend = estX,

y = posY, yend = estY), alpha = 0.25) +

labs(title = 'Distance between Actual vs Predicted', x= 'Position X', y= 'Position Y', colour = "Groups") +

theme(legend.position="bottom", plot.title = element\_text(hjust = 0.5))

return(p)

}

## Loop through a range of k's and store the errors for both raw and weighted

kTesting = function(newSignals, newAngles, trainData, numAngles = 3, range = 3){

error = list(range)

for (i in 1:range){

estXY = predXY(newSignals = onlineSummary[ , 6:12],newAngles = onlineSummary[ , 4]

,offlineSummary, numAngles = 3, k = i)

weightedEstXY = weightedPredXY(newSignals = onlineSummary[ , 6:12],newAngles = onlineSummary[ , 4]

,offlineSummary, numAngles = 3, k = i)

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

actualXY = onlineSummary[ , c("posX", "posY")]

error[[i]] = sapply(list(estXY, weightedEstXY), calcError, actualXY)

print(error[[i]])

}

return(error)

}

## calculate error and plot graph comparing

error = kTesting(onlineSummary[ , 6:12], newAngles = onlineSummary[ , 4] ,offlineSummary, numAngles = 3, range = 10)

errorDF = data.frame(do.call(rbind, error))

errorDF$k <- as.numeric(row.names(errorDF))

pdf(paste0("ErrorPlotsAccrossRange.pdf"))

ggplot(errorDF, aes(k)) +

geom\_line(aes(y = X1, colour = "Raw Average")) +

geom\_line(aes(y = X2, colour = "Weighted Average")) +

labs(title = 'Raw Average vs Weighted Average', x= 'K Nearest Neighbors', y= 'Error', colour = "Metric") +

theme(legend.position="bottom",plot.title = element\_text(hjust = 0.5))

dev.off()

###Below creates the plots showing the distances between k=1 & k=6

## plot distances for k=1

estXY = predXY(newSignals = onlineSummary[ , 6:12],newAngles = onlineSummary[ , 4]

,offlineSummary, numAngles = 3, k = 1)

actualXY$estX <- estXY[,1]

actualXY$estY <- estXY[,2]

pdf(paste0("Distance\_Plot\_Weighted\_k1.pdf"))

plotDistance(actualXY)

dev.off()

## plot distance plots for k=6 weighted

weightedEstXY = weightedPredXY(newSignals = onlineSummary[ , 6:12],newAngles = onlineSummary[ , 4]

,offlineSummary, numAngles = 3, k = 6)

actualXY$estX <- weightedEstXY[,1]

actualXY$estY <- weightedEstXY[,2]

pdf(paste0("Distance\_Plot\_Weighted\_k6.pdf"))

plotDistance(actualXY)

dev.off()