Authors: Justin Beer, Harsha Kommanapalli, Anthony Dalton

Date: 2/20/2018

Predicting Device Location

Using a Real Time Location Services Dataset

## Abstract

In today’s world of internet connected devices, the IoTs, and real-time location services (RTLS) it is paramount for the Data Scientist to be able to understand, munge, evaluate, and analyze RTLS data. In this case study, we seek to understand a RTLS dataset through analyzing the access points where the data from devices was received. For an access point that has two MAC addresses, do signals received at either one or both MACs yield a better prediction of location? Also, does utilizing the signal strength or inverse distance as a weight yield a better KNN model for location prediction?

We will utilize the data munging techniques as prescribed in Chapter 1 of Data Science in R (Nolan, Lang 2015) for importing, munging, and analyzing the RTLS data. In order to determine which MAC address yields better location prediction we will analyze the signal strengths of requests received at both MACs along using K Nearest Neighbors analysis for location prediction. Next, we will assess the accuracy of the KNN location prediction by comparing between the use of mean signal strength and inverse distance weighting.

We find that the signal strength for both MAC addresses at the single access point are nearly identical and that using MAC 00:0f:a3:39:e1:c0 as the predictor of location results in a lower error than using MAC 00:0f:a3:39:dd:cd does. However, utilizing both MACs at the same time to predict location results in a magnitudes lower error than using either one alone.

Finally, we also want to find out whether using the inverse of the signal strength as a weight is better in KNN than using the average distance away from an observation. We find that using the inverse signal strength weighting yields a lower error than the typical KNN implementation using just the mean.

## Introduction

With the ever growing world of the Internet of Things, we have more devices coming online and sharing their location than ever before. As device usage grows so does the capabilities of Real-Time Location Services (RTLS). From being able to track packages, warehouse shipments, to equipment locations, RTLS has grown to being able to pin-point exact locations down to the inch.

For this case study, we are tasked with using an RTLS dataset as set forth in Chapter 1 of Data Science in R (Nolan, Lang 2015). This dataset is a collection of WiFi signals received at 6 stationary access points located on the same floor of a building at the University of Mannheim. The floor layout and access points can be seen in Figure 1.

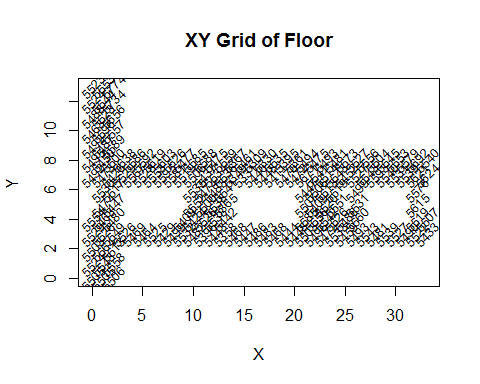


Figure 1. XY Grid of Floor, a mapping of the floor grid for the building based upon where the signal originated

One of these access points, the one located in the first room from the left on the above floor plan, has two different MAC addresses, 00:0f:a3:39:dd:cd and 00:0f:a3:39:e1:c0. In the text, Nolan and Lang decide to only use signals received from 00:0f:a3:39:e1:c0. We are tasked with verifying the assumption that 00:0f:a3:39:e1:c0 is the best MAC to use from this access point and to assess the location predicitive ability of using one or both MAC signal strengths data.

Furthermore, we are tasked with assessing two different KNN models. The first, which is provided in the text, utilizes the mean signal strength to determine the nearest neighbors to access points. The second, we are tasked with creating, which involves using a weighted method of inverse distance. We will compare the accuracy of both models to see which one yields the lowest error in predicting a devices location on the floor.

## Literature Review

The text we are utilizing, “Data Science in R: A Case Studies Approach to Computational Reasoning and Problem Solving” (Nolan, Lang 2015), teaches us how to access, download, import, munge, and analyze the RTLS dataset. The software used is R and the methods are standard for R coding. The implementation of KNN that they use is that of manual coding of the logical structure. While it is good for the student to be able to see the code behind KNN, there are other pre-built R packages and functions that could be utilized by the student in later KNN analysis. For example, the package “class” (Ripley 2015) contains a robust KNN function and can be utilized for more concise analysis in the future.

## Methods

In order to wrap our minds around the RTLS data we need to be able to look at the signals received by the access points and analyze them. To accomplish this, the textbook shows us how to calculate and summarize the signals by their received strength and the direction relative to the access point that the device was facing. The following code creates the summary of our signals for the Online and Offline datasets.

We can better understand what these summary tables are showing us by creating a topographical heatmap of the received signal strengths and overlaying the floor plan grid. The below graphs, Figure 2 and Figure 3, show a plot of the floor plan with a heatmap of the signal strength where red is the strongest signal received by the access points. The stronger the signal the closer the device is to the access point.

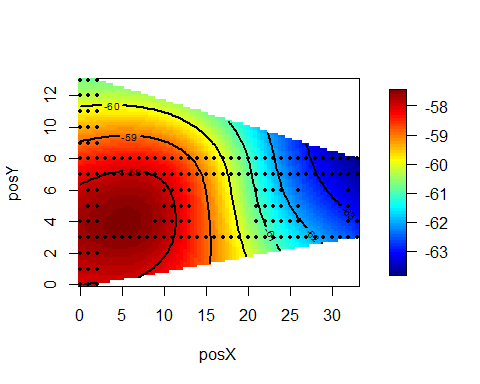


Figure 2. Signal strength versus XY coordinate position.

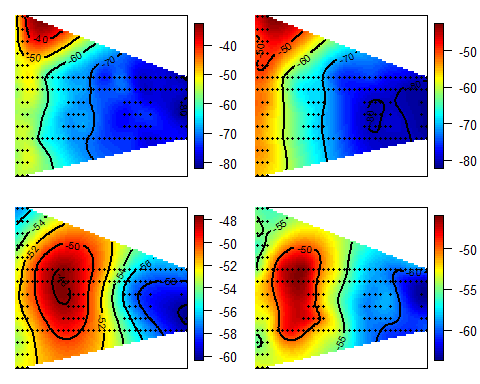


Figure 3. Signal strength versus position for different access points.

Now that we have a better understanding of the signal strengths at different locations we also need to assess the direction the device was facing at the time of the received signal. This will aid us later in creating our KNN function for predicting the device location based upon received signal strength.

Up to this point, we have calculated the signals location, strength, and orientation. We have all of the information needed to be able to build our KNN model. In order to build the KNN model we will need to calculate the signal strength, or distance, of a training set against a subset of our online data. We will use the offline data to train our KNN model and use a subset of our online data to attempt to predict the location of the new signal. Since we have collected the signal strengths at the different access points from the offline device at grid points spaced out by one meter, we will be able to create a KNN model that predicts the new signals location.

The KNN model calculates the distance, which is the difference between the signal strength mapping and the new signal we are trying to predict, and uses this value to find the nearest neighbors of our new signal. From there we use an average to locate an approximate location, our prediction of the new signals location.

We then assess the accuracy of our model using calcError() and compare our predicted values to the actual X and Y coordinates from our offline dataset. Here is an example where we compare the accuracy between using K = 1 and K = 3. We can see that using more neighbors yields a smaller error by more than half. We obtain the following errors respectively:

## [1] 510.4003 244.2070

Through the use of cross-validation, where we use many values of K, we can arrive at a K value that yields us the most favorable and lowest error. That value for this dataset is approximately K = 8.

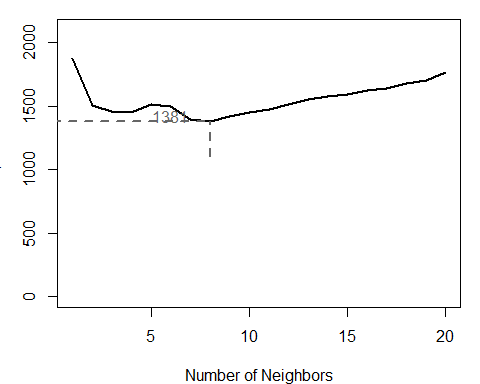


Figure 4. Number of neighbors versus error.

## Results

For our first task of evaluating which of the two MACs, 00:0f:a3:39:dd:cd and 00:0f:a3:39:e1:c0, to utilize for the one access point, we start by analyzing our datasets including the two MACs separate and then together. This way we can use our KNN model to find out which of the MACs have a lower error when predicting location and whether or using both yields a lower error than using the MACs separately.

The textbook made the decision to only use 00:0f:a3:39:e1:c0 and omitted the signals received by 00:0f:a3:39:dd:cd. However, we need to analyze if that is the best choice. Figure 5 provides boxplots of the MAC addresses received signal strength by device orientation. While the values for signal strength are negative, the lower the number the weaker the received signal strength.

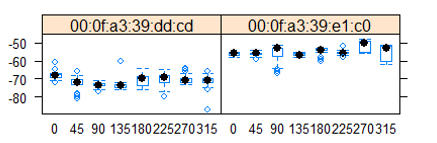


Figure 5. Boxplots of Signal Strength versus Device Orientation for each MAC

Also, we can plot the heatmap for both MACs next to each other so that we can easily compare the signal strengths received. Figure 6 has two plots for 00:0f:a3:39:dd:cd at the top and two plots for 00:0f:a3:39:e1:c0 at the bottom. Nolan and Lang were correct in their assumption that the MAC 00:0f:a3:39:e1:c0 was superior than 00:0f:a3:39:dd:cd as we can see that it received stronger signals for a wider area while having a better gradient for predictive ability. We still need to assess if using both of these MACs will yield a better location prediction.

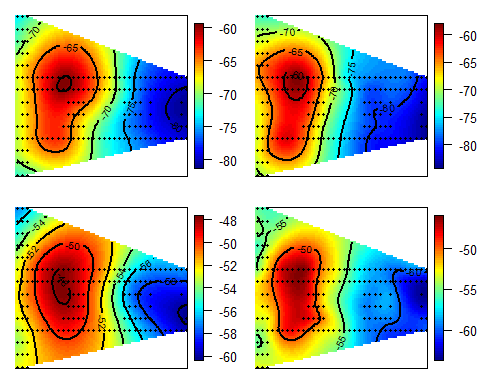


Figure 6. X & Y Coordinates of received Signal Strength.

Using our KNN model that we built we use a value of K = 8 and evaluate using each MAC separately and then both together. We take a subset of our larger datasets extracting only the MACs we want to assess and then feed that into our KNN model. Taking MAC 00:0f:a3:39:dd:cd and excluding the other we see that the model has an error of 20073.72:

estXYk5ddcd = predXY(newSignals = onlineSummary[ , c(6,8:11)],

newAngles = onlineSummary[ , 4], offlineSummary, numAngles = 3, k = 8)  
calcError(estXYk5ddcd, actualXY)

## [1] 20073.72

While taking MAC 00:0f:a3:39:e1:c0 and excluding the previous we see that the model has an error of 13086.57:

estXYk5e1c0 = predXY(newSignals = onlineSummary[ , 7:11],

newAngles = onlineSummary[ , 4], offlineSummary, numAngles = 3, k = 8)  
calcError(estXYk5e1c0, actualXY)

## [1] 13086.57

Using both MACs together yields an error of 235.1959:

estXYk5ddcde1c0 = predXY(newSignals = onlineSummary[ , 6:11],

newAngles = onlineSummary[ , 4], offlineSummary, numAngles = 3, k = 8)  
calcError(estXYk5ddcde1c0, actualXY)

## [1] 235.1959

We can see that the MAC address 00:0f:a3:39:dd:cd yields an error of 20073.72while the MAC address of 00:0f:a3:39:e1:c0 yields an error of only 13086.57. Therefore, it is better to use 00:0f:a3:39:e1:c0 as Lang and Nolan did in the text. However, and intuitively, using both MACs together yields the best error of 235.1959.

Now that we have assessed that it is better to use both MACs together in predicting a devices location, we want to move onto assessing using the mean versus the inverse distance weighting in our KNN model. We must first create another model that uses and passes along the inverse of the signal strength, distance, so that we can use the weighting in predicting the location. We have accomplished this through the following code:

findNNweight = function(newSignal, trainSubset) {  
 # We are subtracting the newsignal from the trainss  
 diffs <- apply(trainSubset[ , 4:9], 1, function(x) x - newSignal)  
 dists <- apply(diffs, 2, function(x) sqrt(sum(x^2)))  
 closest <- order(dists)  
 distsClosest <- dists[closest]  
 # This is our inverse of the distance  
 w1 <- as.numeric(1/distsClosest)  
 # Grab the XY, X, and Y coordinates  
 train2 <- trainSubset[closest, 1:3 ]  
 # Combine our coordinates with the inverse  
 return(cbind(train2,w1))  
 }  
 # Function for predicting signal strengths close to the locations in the training data  
 predXYweight = function(newSignals, newAngles, trainData,   
 numAngles = 1, k = 8){  
 closeXY = list(length = nrow(newSignals))  
 for (i in 1:nrow(newSignals)) {  
 trainSS = selectTrain(newAngles[i], trainData, m = numAngles)  
 findNNreturn = findNNweight(newSignal = as.numeric(newSignals[i, ]), trainSS)  
 # Using the inverse signal strength (distance) we calculate the actual weight for each observation per k  
 weighting = append(findNNreturn[1:k, 4]/sum(findNNreturn[1:k, 4]), rep(0, nrow(findNNret urn)-k))  
 findNNreturn[, 2:3] = findNNreturn[, 2:3]\*weighting  
 # Grab our final data for predicting location  
 closeXY[[i]] = findNNreturn[,1:3]  
 }  
 # Here we attempt to predict the location of the newSignals using the weighting we created  
 estXY = lapply(closeXY, function(x) sapply(x[ , 2:3], function(x) sum(x)))  
 estXY = do.call("rbind", estXY)  
 return(estXY)  
 }

We have to calculate the distances, just as we did in the textbook version of KNN, but this time we need to take its inverse and bind it will the X & Y coordinates. Then we pass this bound data frame through to our predXYweight function. In this function, we actually build our weighting for each observation by taking the inverse we passed through and dividing it by the sum of all of the inverses. This yields our final data frame that take and sum over the columns to acheive an overall weighting for each position. Finally, we can take this and compare it against the actual XY values we built from our offline dataset and see if we are accurate in predicting the new signal devices location.

Here we calculate an estimate using first our old method of the mean and secondly our new method of inverse weighting.

estXYk3weight = predXYweight(newSignals = onlineSummary[ , 6:11],

newAngles = onlineSummary[ , 4], offlineSummary, numAngles = 1, k = 8)  
calcError(estXYk3weight, actualXY)

## [1] 249.1513

estXYk3 = predXY(newSignals = onlineSummary[ , 6:11], newAngles = onlineSummary[ , 4], offlineSummary, numAngles = 1, k = 8)  
calcError(estXYk3, actualXY)

## [1] 267.4947

The older method of using the mean to find the nearest neighbors and predict yields an error of 268.4947 while using the new method of the inverse of the signal strength, distance, yields an error of 249.1513 t is clear that the inverse weighting KNN implementation yields the lowest error in comparison and should be utilized for future predictions of device location.

## Conclusion

In conclusion, we find that using both MAC addresses 00:0f:a3:39:dd:cd and 00:0f:a3:39:e1:c0 along with the other wanted MACs yields the lowest error for our optimum K value of 8. Also, we find that using the inverse weighting of the signal strength is superior to using just the mean of the distance when predicting a devices location.

## Appendix

### Reference

1. Nolan, D., & Lang, D. T. (2015). Data Science in R: A Case Studies Approach to Computational Reasoning and Problem Solving. (Data science in R.) London: CRC Press.
2. Ripley, B. (2015). Package ‘class’. Retrieved from <https://cran.r-project.org/web/packages/class/class.pdf>

### Complete Code Base

####################\*  
# Unit 6 Case Study  
####################\*  
  
# Reference for code used: http://www.rdatasciencecases.org/GeoLoc/code.R  
# Here is the readData function stated in 1.2  
# In the raw data set there are 12 MACs and 8 Channels.   
# Unwanted MACs: "00:0f:a3:39:e0:4b" "00:0f:a3:39:e2:10" "00:04:0e:5c:23:fc" "00:30:bd:f8:7f:c5" "00:e0:63:82:8b:a9"  
# Wanted MACs: "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"  
#############################################  
# Functions  
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")  
  
# Function for munging data  
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)  
}  
  
# Function for adjusting angles  
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]  
}  
  
# Function for selecting training obs  
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")  
}  
  
# Function for plotting heatmaps  
surfaceSS = function(data, mac, angle = 45) {  
 require(fields)  
 oneAPAngleMACc0 = data[ data$mac == mac & data$angle == angle, ]  
 smoothSS = Tps(oneAPAngleMACc0[, c("posX","posY")], oneAPAngleMACc0$avgSignal)  
 vizSmooth = predictSurface(smoothSS)  
 plot.surface(vizSmooth, type = "C", xlab = "", ylab = "", xaxt = "n", yaxt = "n")  
 points(oneAPAngleMACc0$posX, oneAPAngleMACc0$posY, pch=19, cex = 0.5)   
}  
  
# Function for reading in the data and munging  
readData =   
 function(filename = "http://rdatasciencecases.org/Data/offline.final.trace.txt", subMacs)  
 {  
 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")  
   
 # 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)  
   
 # Create unique location identifier  
 offline$posXY = paste(offline$posX, offline$posY, sep = "-")  
   
 return(offline)  
 }  
  
#############################################  
# Prepare the offline and online data  
offline = readData("http://rdatasciencecases.org/Data/offline.final.trace.txt", subMacs)  
online = readData("http://rdatasciencecases.org/Data/online.final.trace.txt", subMacs)  
# Make sure we have 10 MAC addresses for each data set  
unique(offline$mac)  
unique(online$mac)  
# Sort by mac address  
offline = offline[with(offline,order(mac)),]  
  
#############################################  
# Create position for hand-held device for mapping.  
locDF = with(offline,  
 by(offline, list(posX, posY), function(x) x))  
locDF = locDF[ !sapply(locDF, is.null) ] #drop unwanted nulls  
# Determine # of obs at each location  
locCounts = sapply(locDF, nrow)  
locCounts = sapply(locDF,  
 function(df)  
 c(df[1, c("posX", "posY")], count = nrow(df)))  
# Plot out the locations for a visualization of the rooms and hallways  
locCounts = t(locCounts)  
plot(locCounts, type = "n", xlab = "", ylab = "")  
text(locCounts, labels = locCounts[,3], cex = .8, srt = 45)  
  
#############################################  
# Create offline summary table of the signals  
byLocAngleAP = with(offline,  
 by(offline, list(posXY, angle, mac),  
 function(x) x))  
summary(byLocAngleAP)  
byLocAngleAP = byLocAngleAP[ !sapply(byLocAngleAP, is.null) ]  
# Create offlineSummary  
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)  
# Create onlineSummary  
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 = 7,  
 dimnames = list(ans$posXY, names(avgSS)))  
 cbind(ans, y)  
 }))  
onlineSummary = do.call("rbind", byLoc)   
  
#############################################  
# The relationship between Signal and Distance  
# Select one MAC and one orientation to examine  
oneAPAngle = subset(offline, angle == 0)  
# Make topographical map  
oneAPAngle = subset(offlineSummary, angle == 0)  
library(fields)  
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))  
par(parCur)  
  
#############################################  
# Choosing our MAC address 00:0f:a3:39:dd:cd  
subMacs = unique(offline$mac)  
offlineSummaryMAC = subset(offlineSummary, mac == "00:0f:a3:39:dd:cd")  
# Create matrix with relevant positions  
AP = matrix(c(7.5, 6.3, 7.5, 6.3, 12.8, -2.8, 33.5, 2.8, 2.5, -0.8, 33.5, 9.3, 1.0, 14.0), ncol = 2, byrow = TRUE,   
 dimnames = list(subMacs, c("x", "y") ))  
# Compute distances from the locations of the device emitting the signal to the access point  
diffs = offlineSummaryMAC[ , c("posX", "posY")] - AP[ offlineSummaryMAC$mac, ]  
# Find the Euclidean distance  
offlineSummaryMAC$dist = sqrt(diffs[ , 1]^2 + diffs[ , 2]^2)  
# Plot the scatter plots  
library(lattice)  
xyplot(signal ~ dist | factor(mac) + factor(angle),  
 data = offlineSummaryMAC, pch = 19, cex = 0.3,  
 xlab ="distance")  
  
#############################################  
# 1.5.2 Choice of Orientation  
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 ]  
}  
# Map and adjust angles  
angles = angles + nearestAngle  
angles[angles < 0] = angles[ angles < 0 ] + 360  
angles[angles > 360] = angles[ angles > 360 ] - 360  
# Select obs to analyze  
offlineSubset = offlineSummary[ offlineSummary$angle %in% angles, ]  
# Aggregate signal strengths from the angles and create data structure  
reshapeSS = function(data, varSignal = "signal",   
 keepVars = c("posXY", "posX","posY"),  
 sampleAngle = FALSE,   
 refs = seq(0, 315, by = 45)) {  
 byLocation =  
 with(data, by(data, list(posXY),   
 function(x) {  
 if (sampleAngle) {  
 x = x[x$angle == sample(refs, size = 1), ]}  
 ans = x[1, keepVars]  
 avgSS = tapply(x[ , varSignal ], x$mac, mean)  
 y = matrix(avgSS, nrow = 1, ncol = 7,  
 dimnames = list(ans$posXY, names(avgSS)))  
 cbind(ans, y)  
 }))  
   
 newDataSS = do.call("rbind", byLocation)  
 return(newDataSS)  
}  
trainSS = reshapeSS(offlineSubset, varSignal = "avgSignal")  
  
#############################################  
# use selectTrain function for selecting training set  
train130 = selectTrain(130, offlineSummary, m = 3)  
  
#############################################  
# 1.5.3 Finding the Nearest Neighbors  
# Calculate distances and neighbors  
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 ])  
}  
# Function for predicting signal strengths close to the locations in the training data  
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)  
}  
# Use the pred function  
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)  
# ASsess fit by calculating error  
calcError =   
 function(estXY, actualXY)  
 sum( rowSums( (estXY - actualXY)^2) )  
# Apply function to our two sets of errors  
actualXY = onlineSummary[ , c("posX", "posY")]  
sapply(list(estXYk1, estXYk3), calcError, actualXY)  
  
#############################################  
# 1.5.4 Cross-Validation and Choice of k  
# Where v = # of validation folds to use  
v = 11  
permuteLocs = sample(unique(offlineSummary$posXY))  
permuteLocs = matrix(permuteLocs, ncol = v, nrow = floor(length(permuteLocs)/v))  
onlineFold = subset(offlineSummary, posXY %in% permuteLocs[ , 1])  
keepVars = c("posXY", "posX","posY", "orientation", "angle")  
onlineCVSummary = reshapeSS(offline, keepVars = keepVars, sampleAngle = TRUE)  
# First fold  
onlineFold = subset(onlineCVSummary,   
 posXY %in% permuteLocs[ , 1])  
offlineFold = subset(offlineSummary,  
 posXY %in% permuteLocs[ , -1])  
# Use our predXY function w/ our cross-validated version of online and offline data  
estFold = predXY(newSignals = onlineFold[ , 6:11],  
 newAngles = onlineFold[ , 4],  
 offlineFold, numAngles = 3, k = 3)  
# Find error in estimates  
actualFold = onlineFold[ , c("posX", "posY")]  
calcError(estFold, actualFold)  
  
#############################################  
# For each fold we want to find the KNN estimate and aggregate the errors over "v" folds.  
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)  
 }  
}  
calcError(estFold, actualFold)  
  
#############################################  
# Export results to a PDF  
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(0, 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()  
# Using the above export, which is an graph of the errors (sum of squares) received as a function of k increasing, we come to use k=5  
estXYk5 = predXY(newSignals = onlineSummary[ , 6:11], newAngles = onlineSummary[ , 4], offlineSummary, numAngles = 3, k = 5)  
# Check our errors for k=5  
calcError(estXYk5, actualXY)  
  
#############################################