Code

Unit 6 Case Study - Real Time Location

Rahn Lieberman, Jainy Selvakumar, Mike Sylvester, Shobana Sundaram

[1.1] Introduction

A current area of interest in many industries is determining an internet connected device's location. This could be a phone, a tablet or laptop, or something else with some sort of tracking chip embedded in it. By examining a devices name, and how it connects to different access points in a physical area with a known network topology, we can determine it's location.

Some practical uses of this may be to track someones location in a store to see where they are spending the most time. One could then examine what is in that area to determine what people are most interested in buying.

For this case study, we are using some known R code to read and analyze a device location in an office. See [1] for specific information.

We diverge from the code and analysis path given to examine two different MAC addresses and how the results differ based on whether they are included or not. In addition, the location prediction model using K nearest neighbors was explored to see if a weighted mean based on signal strength would improve accuracy over a simple mean.

The first portion of the code below is setting up and cleaning the dataset. The data exploration and detail analysis begins in section 1.3.

[1.2] Import Data and inital cleanup

First thing to do is load our data, and do some initial evaluation of it. We also clean up the data to put correct types on each column.

Hide

```
#Create a function to remove unwanted characters
processLine = function(x) {
  tokens = strsplit(x, "[;=,]")[[1]] #This is using a regex in the split to break as [
 ] = , characters
  if (length(tokens) <= 10)
      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), tm
p)
}


# Create a function that will make all the orientations line up to the closed "true" v
alue
# that we are interested in.  i.e. 0, 45, 90, etc.
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]
}

#Create a function to read data
readData =
function(filename = '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)

  #Remove the comment lines ,beginning with a #
  lines = txt[ substr(txt, 1, 1) != "#" ]

  # This line is to raise errors when warnings occur.
  options(error = recover, warn = 1)
  tmp = lapply(lines, processLine)
  # The above gives us a matrix of the values.
  # We want to combine all matrices together
  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", ]

  # 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")

  # convert orientation to angle
  offline$angle = roundOrientation(offline$orientation)

  # 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, ]


  return(offline)
}
```

[1.3] Cleaning the Data and Preparing for Analysis

Hide

```
  offline = readData()
dim(offline)
```

[1.3.1] Orientation of the devices

Each device should be orientated in one of 8 directions. These should be listed in degrees, and each direction is 45 degrees apart.

Hide

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

[1.3.2] Exploring MAC address

The original dataset contained 12 MAC addresses and 8 channels. In initial analysis, We observed that the MAC addresses have a one to one correspondence with channels. Therefore, we removed the redundant channel data in the readData function above. In addition, we were only expecting 6 access points. The additional MAC addresses had a low number of observations. Therefore, the code below will keep only the MAC addresses with top 7 readings. However, since we have already prepped the data by filtering to the MAC addresses of interest in the readData function above, running the code below will have no additional clean up impact.

Hide

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

table(offline$mac)

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

[1.3.3]Exploring the Position of the Hand-Held Device

Hide

```
locDF = with(offline,
    by(offline, list(posX, posY), function(x) x))
    length(locDF)
sum(sapply(locDF, is.null))
locDF = locDF[ !sapply(locDF, is.null) ]

#How many different locations do we have data for?
length(locDF)

locCounts = sapply(locDF, nrow)
locCounts = sapply(locDF,
function(df)
c(df[1, c("posX", "posY")], count = nrow(df)))
class(locCounts)
dim(locCounts)

#How many observations do we have for each location?
locCounts[ , 1:8]

#Create a plot of the number of observations for each location
locCounts = t(locCounts)
plot(locCounts, type = "n", xlab = "", ylab = "")
text(locCounts, labels = locCounts[,3], cex = .8, srt = 45)

# Find global variables that are needed if we run this in a new R session
library(codetools)
findGlobals(readData, merge=FALSE)$variables
```

[1.4] Signal Strength Analysis

[1.4.1] Distribution of Signal Strength

Factor containing all unique combinations of (x,y) pairs at 166 locations. And creating data frames for every combination of (x, y) angle and access point

Hide

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

byLocAngleAP = with(offline,
                    by(offline, list(posXY, angle, mac),
                       function(x) x))

#Function to summarize signal data by AP, location and angle
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
         })

## Create a new data frame of the summary statistics
offlineSummary = do.call("rbind", signalSummary)
```

[1.4.2] Examine the Relationship between Signal and Distance

Hide

```
#Create a function to draw the heat map by passing parameters
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)
}
```

Run the heatmap for both access points of interest

Hide

```
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)
#
# #MAC address of 00:0f:a3:39:e1:c0
# offlineSummary = subset(offlineSummary, mac = subMacs[1])
#
# #create a matrix for the access points on the floor plan
# AP = matrix( c( 7.5, 6.3, 2.5, -.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
#
# #MAC address of 00:0f:a3:39:dd:cd
# offlineSummary = subset(offlineSummary, mac = subMacs[2])
#
# #create a matrix for the access points on the floor plan
# AP = matrix( c( 7.5, 6.3, 2.5, -.8, 12.8, -2.8,
#                 1, 14, 33.5, 9.3,  33.5, 2.8),
#             ncol = 2, byrow = TRUE,
#             dimnames = list(subMacs[ -1 ], c("x", "y") ))
#
# AP

#MAC address of 00:0f:a3:39:dd:cd is dropped
offlineSummary = subset(offlineSummary, mac != subMacs[2])

#create a matrix for the access points on the floor plan
AP = matrix( c( 7.5, 6.3, 2.5, -.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
```

Exploring the relationship between signal strenth and distance from access point

Hide

```
#calculate the distance from device to access point
diffs = offlineSummary[ , c("posX", "posY")] -
          AP[ offlineSummary$mac, ]

#Calculate Euclidean distance betweeen device and access point
offlineSummary$dist = sqrt(diffs[ , 1]^2 + diffs[ , 2]^2)

# #draw scatterplots of each access point and 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))
# library(lattice)
# xyplot(signal ~ dist | factor(mac) + factor(angle),
#        data = offlineSummary, pch = 19, cex = 0.3,
#        xlab ="distance")
# par(oldPar)
# dev.off()
```

[1.5] K Nearest Neighbor Method to Predict Location

[1.5.1]Preparing the Test Data

Hide

```
#Use the readData function to create a dataframe using the test data
macs = unique(offlineSummary$mac)
online = readData("online.final.trace.txt", subMacs = macs)

#Create data frames for each unique location identifier for every combination of (x,
 y) angle and access point
online$posXY = paste(online$posX, online$posY, sep = "-")

#How many unique test locations do we have?
length(unique(online$posXY))

#How many observations do we have for each location?
tabonlineXYA = table(online$posXY, online$angle)
tabonlineXYA[1:6, ]

#create a dataframe of the signal strengths with each access point in its own column
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)
```

Hide

```
#Find the angles that are neighboring angles that can help filter the training data wh
en calculating the test data
#m = number of angles and angleNewObs = angle of the new observation
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 ]
}

#Remap the angles to be consistent to the values in refs
angles = angles + nearestAngle
angles[angles < 0] = angles[ angles < 0 ] + 360
angles[angles > 360] = angles[ angles > 360 ] - 360

#Selected the angles of interest from the training dataset
offlineSubset =
  offlineSummary[ offlineSummary$angle %in% angles, ]

#Create a function to create a dataframe of the filtered data similar to the summary t
able
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)
}

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

#Create a function to select the angles and reshape the dataframe given the parameters
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")
}
```

Hide

```
#test the function using m = 3 and angle of 130
# train130 = selectTrain(130, offlineSummary, m = 3)
#
# head(train130)
#
# length(train130[[1]])
```

[1.5.2] Finding the Nearest Neighbors

Hide

```
#Create a function to find the nearest neigbor of a new point based on the training da
ta (in terms of signal strngths)
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 ])
}
```

[1.5.2.1] Estimate Using a Simple Average

Hide

```
#Create a function to predict the location of the new point based on the nearest neigh
bors identified by findNN
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)
}
```

[1.5.2.2] Estimate Using a Weighted Average

Hide

```
# Weighted average function for k = 1
predXY_w = function(newSignals, newAngles, trainData,
                  numAngles = 1, k = 1){

  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)
  }

  nz=.001

  estXY = lapply(closeXY,
                 function(x) sapply(x[ , 2:3], function(x) weighted.mean(x[1:k],
c(1/(sum(x[1])+nz)))))

  estXY = do.call("rbind", estXY)
  return(estXY)
}

# Weighted average function for k = 3
predXY3_w = 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)
  }

  nz=.001

  estXY = lapply(closeXY,
                 function(x) sapply(x[ , 2:3], function(x) weighted.mean(x[1:k],
c(1/(sum(x[1])+nz), 1/(sum(x[2])+nz),1/(sum(x[3])+nz)))))

  estXY = do.call("rbind", estXY)
  return(estXY)
}


# Weighted average function for k = 5
predXY5_w = function(newSignals, newAngles, trainData,
                  numAngles = 1, k = 5){

  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)
  }

  nz=.001

  estXY = lapply(closeXY,
                 function(x) sapply(x[ , 2:3], function(x) weighted.mean(x[1:k],
c(1/(sum(x[1])+nz),
1/(sum(x[2])+nz),1/(sum(x[3])+nz),1/(sum(x[4])+nz),1/(sum(x[5])+nz)  ))))

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

[1.5.3] K = 1 and K = 3

Hide

```
#Predict using 3 nearest neighbors and 3 orientations
estXYk3 = predXY(newSignals = onlineSummary[ , 6:11],
                 newAngles = onlineSummary[ , 4],
                 offlineSummary, numAngles = 3, k = 3)

#Predict using k = 3 and weighted average
estXYk3w = predXY3_w(newSignals = onlineSummary[ , 6:11],
                 newAngles = onlineSummary[ , 4],
                 offlineSummary, numAngles = 3, k = 3)


#Predict using 1 nearest neighbors and 3 orientations
estXYk1 = predXY(newSignals = onlineSummary[ , 6:11],
                 newAngles = onlineSummary[ , 4],
                 offlineSummary, numAngles = 3, k = 1)

#Predict using k = 1 and weighted average
estXYk1w = predXY_w(newSignals = onlineSummary[ , 6:11],
                 newAngles = onlineSummary[ , 4],
                 offlineSummary, numAngles = 3, k = 1)
```

Hide

```
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, col = "blue", 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")]

floorErrorMap(estXYk3, onlineSummary[ , c("posX","posY")],
              trainPoints = trainPoints, AP = AP)


floorErrorMap(estXYk1, onlineSummary[ , c("posX","posY")],
              trainPoints = trainPoints, AP = AP)

floorErrorMap(estXYk3w, onlineSummary[ , c("posX","posY")],
              trainPoints = trainPoints, AP = AP)


floorErrorMap(estXYk1w, onlineSummary[ , c("posX","posY")],
              trainPoints = trainPoints, AP = AP)

#Create a function to calculate the sum of squared errors
calcError =
function(estXY, actualXY)
   sum( rowSums( (estXY - actualXY)^2) )

actualXY = onlineSummary[ , c("posX", "posY")]
sapply(list(estXYk1, estXYk3, estXYk1w, estXYk3w), calcError, actualXY)
```

Weighted sum analysis

The weights allow to give priorities to some positions of the area. And weight takes values between zero and one to vary the priority given to different zones.Weights are given by the function, (1/d)/{sum i=1 to k}(1/di) and is inversely proportional to distance.

Above results show the values of error for higher range of weights. We have iterated for weights ranging 0.001 to 0.9. As the weights get closer to zero errors are similar the results obtained for an unweighted function.So range of weights below 0.2 gives a better prdiction.

[1.5.4] Cross-validation

Hide

```
#Determine a number (v) of non-overlapping subsets of equal size
#Build a model for each subset using all the data except the selected subset
#Predict using the model for the selected subset
#Repeat for each subset and then aggregrate the prediction errors accross subsets
#v = Number of subsets
v = 11
permuteLocs = sample(unique(offlineSummary$posXY))
permuteLocs = matrix(permuteLocs, ncol = v,
                     nrow = floor(length(permuteLocs)/v))

onlineFold = subset(offlineSummary, posXY %in% permuteLocs[ , 1])

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)
}

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

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

onlineCVSummary = reshapeSS(offline, keepVars = keepVars,
                            sampleAngle = TRUE)

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)

estFold_w = predXY3_w(newSignals = onlineFold[ , 6:11],
                 newAngles = onlineFold[ , 4],
                 offlineFold, numAngles = 3, k = 3)
```

```
actualFold = onlineFold[ , c("posX", "posY")]
calcError(estFold, actualFold)
calcError(estFold_w, actualFold)
```

[1.5.5] Choice of k

But are we using the best number of k?

Hide

```
#Find the nearest neighbor estimates for each subset for k = 1, 2,...K
#Aggregate the errors over the number of subsets
#Should use k = 10 or 20 but taking too long to run
K = 10
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)
  }
}

#Plot the results to find the best k
plot(y = err, x = (1:K),  type = "l", lwd= 2,
     ylim = c(800, 1600),
     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)
text(x = kMin - 2, y = rmseMin + 40,
     label = as.character(round(rmseMin)), col = grey(0.4))

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

calcError(estXYk5, actualXY)
```

[1.6] Calculate for alternate MAC address

Hide

```
#MAC address of 00:0f:a3:39:e1:c0 is dropped
offlineSummary2 = do.call("rbind", signalSummary)
offlineSummary2 = subset(offlineSummary2, mac != subMacs[2])

table(offlineSummary2$mac)

#create a matrix for the access points on the floor plan
AP = matrix( c( 7.5, 6.3, 2.5, -.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

#calculate the distance from device to access point
diffs = offlineSummary2[ , c("posX", "posY")] -
          AP[ offlineSummary2$mac, ]

#Calculate Euclidean distance betweeen device and access point
offlineSummary2$dist = sqrt(diffs[ , 1]^2 + diffs[ , 2]^2)

#Use the readData function to create a dataframe using the test data
macs = unique(offlineSummary2$mac)
online2 = readData("online.final.trace.txt", subMacs = macs)

#Create data frames for each unique location identifier for every combination of (x,
 y) angle and access point
online2$posXY = paste(online2$posX, online2$posY, sep = "-")

#How many unique test locations do we have?
length(unique(online2$posXY))

#How many observations do we have for each location?
tabonline2XYA = table(online2$posXY, online2$angle)
tabonline2XYA[1:6, ]

#create a dataframe of the signal strengths with each access point in its own column
keepVars = c("posXY", "posX","posY", "orientation", "angle")
byLoc = with(online2,
             by(online2, 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)
                }))

onlineSummary2 = do.call("rbind", byLoc)

dim(onlineSummary2)

#Selected the angles of interest from the training dataset
offlineSubset2 =
  offlineSummary2[ offlineSummary2$angle %in% angles, ]

trainSS = reshapeSS(offlineSubset2, varSignal = "avgSignal")
```

[1.6.1] K = 1 and K = 3

Hide

```
#Predict using 3 nearest neighbors and 3 orientations
estXYk3_2 = predXY(newSignals = onlineSummary2[ , 6:11],
                 newAngles = onlineSummary2[ , 4],
                 offlineSummary2, numAngles = 3, k = 3)

#Predict using k = 3 and weighted average
estXYk3w_2 = predXY3_w(newSignals = onlineSummary2[ , 6:11],
                 newAngles = onlineSummary2[ , 4],
                 offlineSummary2, numAngles = 3, k = 3)

#Predict using 1 nearest neighbors and 3 orientations
estXYk1_2 = predXY(newSignals = onlineSummary2[ , 6:11],
                 newAngles = onlineSummary2[ , 4],
                 offlineSummary2, numAngles = 3, k = 1)

#Predict using k = 1 and weighted average
estXYk1w_2 = predXY_w(newSignals = onlineSummary2[ , 6:11],
                 newAngles = onlineSummary2[ , 4],
                 offlineSummary2, numAngles = 3, k = 1)
```

Hide

```
trainPoints = offlineSummary2[ offlineSummary2$angle == 0 &
                              offlineSummary2$mac == "00:0f:a3:39:dd:cd" ,
                        c("posX", "posY")]

floorErrorMap(estXYk3_2, onlineSummary2[ , c("posX","posY")],
              trainPoints = trainPoints, AP = AP)


floorErrorMap(estXYk1_2, onlineSummary2[ , c("posX","posY")],
              trainPoints = trainPoints, AP = AP)

floorErrorMap(estXYk3w_2, onlineSummary2[ , c("posX","posY")],
              trainPoints = trainPoints, AP = AP)


floorErrorMap(estXYk1w_2, onlineSummary2[ , c("posX","posY")],
              trainPoints = trainPoints, AP = AP)

actualXY = onlineSummary2[ , c("posX", "posY")]
sapply(list(estXYk1_2, estXYk3_2, estXYk1w_2, estXYk3w_2
            ), calcError, actualXY)
```

[1.6.2] Cross-validation

Hide

```
#Determine a number (v) of non-overlapping subsets of equal size
#Build a model for each subset using all the data except the selected subset
#Predict using the model for the selected subset
#Repeat for each subset and then aggregrate the prediction errors accross subsets
#v = Number of subsets
v = 11
permuteLocs = sample(unique(offlineSummary2$posXY))
permuteLocs = matrix(permuteLocs, ncol = v,
                     nrow = floor(length(permuteLocs)/v))

onlineFold2 = subset(offlineSummary2, posXY %in% permuteLocs[ , 1])

offline = readData()
offline = offline[ offline$mac != "00:0f:a3:39:e1:c0", ]
offline$posXY = paste(offline$posX, offline$posY, sep = "-")

onlineCVSummary2 = reshapeSS(offline, keepVars = keepVars,
                            sampleAngle = TRUE)

onlineFold2 = subset(onlineCVSummary2,
                    posXY %in% permuteLocs[ , 1])

offlineFold2 = subset(offlineSummary2,
                     posXY %in% permuteLocs[ , -1])

estFold2 = predXY(newSignals = onlineFold2[ , 6:11],
                 newAngles = onlineFold2[ , 4],
                 offlineFold2, numAngles = 3, k = 3)

estFold2_w = predXY3_w(newSignals = onlineFold2[ , 6:11],
                 newAngles = onlineFold2[ , 4],
                 offlineFold2, numAngles = 3, k = 3)

actualFold = onlineFold[ , c("posX", "posY")]
calcError(estFold2, actualFold)
calcError(estFold2_w, actualFold)
```

[1.6.3] Choice of k

But are we using the best number of k?

Hide

```
#Find the nearest neighbor estimates for each subset for k = 1, 2,...K
#Aggregate the errors over the number of subsets
#Should use k = 10 or 20 but taking too long to run
K = 10
err = rep(0, K)

for (j in 1:v) {
  onlineFold2 = subset(onlineCVSummary2,
                       posXY %in% permuteLocs[ , j])
  offlineFold2 = subset(offlineSummary2,
                        posXY %in% permuteLocs[ , -j])
  actualFold2 = onlineFold2[ , c("posX", "posY")]

  for (k in 1:K) {
    estFold2 = predXY(newSignals = onlineFold2[ , 6:11],
                      newAngles = onlineFold2[ , 4],
                      offlineFold2, numAngles = 3, k = k)
    err[k] = err[k] + calcError(estFold2, actualFold2)
  }
}

#Plot the results to find the best k
plot(y = err, x = (1:K),  type = "l", lwd= 2,
     ylim = c(1000, 2000),
     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)
text(x = kMin - 2, y = rmseMin + 40,
     label = as.character(round(rmseMin)), col = grey(0.4))

estXYk4_2 = predXY(newSignals = onlineSummary2[ , 6:11],
                 newAngles = onlineSummary2[ , 4],
                 offlineSummary2, numAngles = 3, k = 4)

calcError(estXYk4_2, actualXY)
```

[1.7] Results

Generally based on the results in the previous two sections, it appears as if the unweighted mean for predicting location performs better than the weighted mean. The least square error equals unweighted function as the weight nears zero.

Furthermore, there appears to be a significant difference between the two MAC addresses. Least square error obtained for MAC address of 00:0f:a3:39:dd:cd has lesser error than the MAC address of 00:0f:a3:39:e1:c0 with error being 270 an 306.7 for unweighted function and 273 and 445.4 for weighted function for the same k value and same weights.

For further comparison, the k = 5 was held stable and the sum of square errors was calculated for all 4 scenarios. As seen in the results below, the MAC address of 00:0f:a3:39:dd:cd should be kept in the data model and used for RTLS in order to increase accuracy.

Though error obtained using both MAC addresses were lower compared to individual MAC address of 00:0f:a3:39:dd:cd or 00:0f:a3:39:e1:c0. It can be hypothesized that using both MAC addresses simultaneously would skew the data and provide less accurate predictions of location.

Hide

```
estXYk5 = predXY(newSignals = onlineSummary[ , 6:11],
                 newAngles = onlineSummary[ , 4],
                 offlineSummary, numAngles = 3, k = 5)

estXYk5_2 = predXY(newSignals = onlineSummary2[ , 6:11],
                 newAngles = onlineSummary2[ , 4],
                 offlineSummary2, numAngles = 3, k = 5)

estXYk5w = predXY5_w(newSignals = onlineSummary[ , 6:11],
                 newAngles = onlineSummary[ , 4],
                 offlineSummary, numAngles = 3, k = 5)

estXYk5w_2 = predXY5_w(newSignals = onlineSummary2[ , 6:11],
                 newAngles = onlineSummary2[ , 4],
                 offlineSummary2, numAngles = 3, k = 5)

sapply(list(estXYk5, estXYk5w, estXYk5_2, estXYk5w_2), calcError, actualXY)
```

References

Nolan and Lang, Data Science in R, A Case Studies Approach, CRC Press