Unit 6 Case Study - Predicting Location Via Indoor Positioning Systems and KNN

June 22, 2016

Carl Bryant, Sujatha Chinnappa, Carrie Garcia, Brian Mente

Abstract - In order to develop an indoor positioning system that used signal strength measurements from wi-fi access points on the floor of a building, a team of researchers collected a set of training data and applied a K-nearest neighbors algorithm to the data, then evaluated their model with a subsequent test set of data. In cleaning of the data, it was discovered that one of the access point locations had two access points with different identifying addresses and different sets of signal strength readings. The first access point was chosen for the model and the second discarded. We explored whether that choice was correct by conducting two more sets of analysis: one with the discarded access point in place of the one that was kept, and one with both access points included in the data set. Based on our chosen evaluation criteria, the models created with data from both access points in the center of the building were more accurate than the two scenarios with only one of the two access points at for all values of K. Between the two scenarios with only one access point, the scenario that included the previously discarded access point resulted in better prediction performance for every value of K.

1. Introduction and Project Approach

In recent years, the Internet of Things (IoT) has expanded at a rapid rate. Using internet technologies for communication with mobile devices, activity-tracking health monitoring bands and other smart devices create a wealth of data. While all these devices can be tracked with some kind of GPS-based navigation systems, challenges arise when the devices are within a building. Ability to locate IoT devices inside a building using a system are categorized as “Indoor Positioning System (IPS). There are several technologies available to build an IPS system which includes using nearest “Wifi Access Points”. Real-time location prediction models for locating an IoT device inside a building have many applications.

In this project, we will recreate an indoor positioning system using the k-nearest neighbor prediction algorithm using the training dataset provided to predict the locations of the new mobile devices based on the signal strength for the devices to the nearest access points. Our case study builds on the one in the text by optimize prediction accuracy through exploration of three different training data set scenarios.

2. About the Data

In our text, the author introduces a training dataset and case study for an indoor positioning system. As illustrated in below Figure 1, the training dataset includes 166 grid points, 110 signal strengths to each of the 6 access points for 8 orientation combinations. (166 grids * 8 angles * 110 recordings = 146,080 total data points). The training dataset also had 5312 comment lines which were removed as part of data preparation).

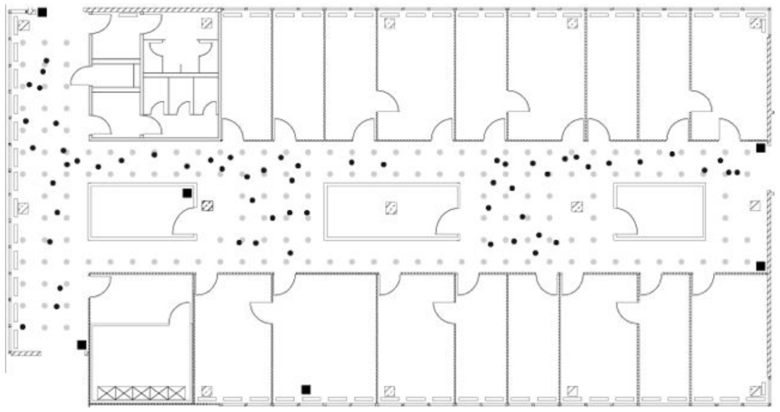

Figure 1: Floor plan for the test environment (6 access points, 166 test grids)

This figure shows the location of each of the signal strength observations and the location of the 6 access points overlaid on a floorplan of the floor where the data were collected. The large black square markers are the access points, while the smaller gray and black markers are the signal strength observations. The gray observations show the regularly spaced training data set measurements, while the more randomly occuring black markers are the test set observations.

The datasets (training and testing) includes the following variables:

t	:timestamp when the signal strength was observed
Id	:MAC address of the mobile device
pos	:x, y and z coordinates. (z represents the levels of the building)
degrees	:orientation of the mobile device. For this study, each device's signal strength is measured at 8 orientations with 45 degrees apart
mac	:mac address of the access points

Dr. McGee provided the R markdown file that we used to clean, explore and analyze the data. Because the data cleansing is covered extensively in the text, we will not spend time in this paper restating it (refer to Nolan and Temple Lang chapter 1 for more information).

We do, however, need to point out a key data cleansing component. There are two datasets used in this case study. One is used to train the model and the other to test. The authors decided to eliminate one of the MAC addresses in the training dataset because the heatmap of the median signal for two access points were very similar (see Figure 2).

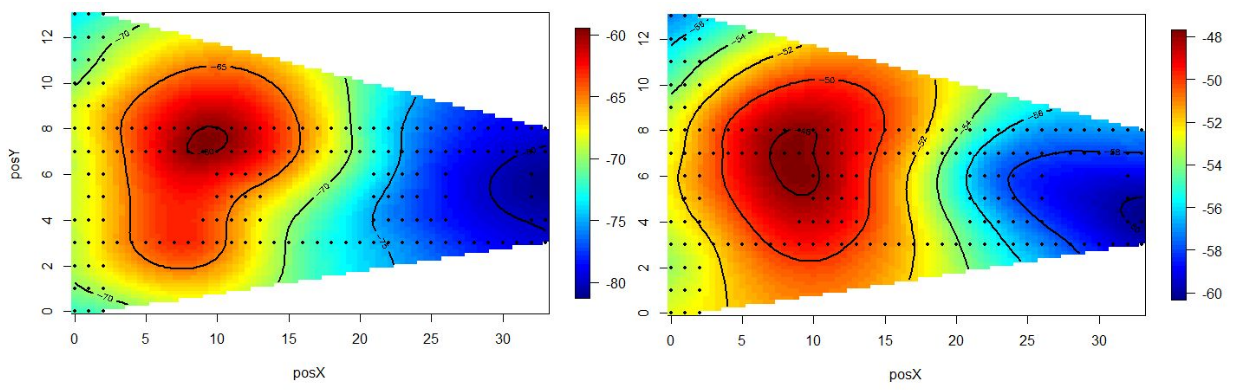

Figure 2: Heat map of median signals at two access points that are similar

This figure shows the two access points that were in similar locations in the building. The y-axis shows the y-coordinate of the location of the access point, and the x-axis shows the x-coordinate of the location of the access point. The color shows the median signal strength of the readings measured at the training locations shown by black dots. The authors deleted the access point "00:0f:a3:39:dd:cd" from their KNN model, which had the signal strength pattern shown on the left. Access point "00:0f:a3:39:e1:c0", whose signal strength pattern is shown on the right, was kept in the data used to train the K-nearest neighbors model. While the measured signal strengths differ significantly between the two access points (the deleted access point has much lower median signal strength readings), the general pattern of the signal strengths is very similar.

We duplicated their training dataset using the same six access points for our first KNN model (as shown below as original training dataset). Next we swapped out the MAC address they kept from the similar pair for the one they deleted to see how it affected the results (referred to as training dataset 2). And for training dataset 3 we kept all seven MAC addresses.

MAC Address	Original Training Dataset & Model 1	Training Dataset & Model 2	Training Dataset & Model 3
00:0f:a3:39:e1:c0	Yes	No	Yes
00:0f:a3:39:dd:cd	No	Yes	Yes
00:14:bf:b1:97:8a	Yes	Yes	Yes
00:14:bf:3b:c7:c6	Yes	Yes	Yes
00:14:bf:b1:97:90	Yes	Yes	Yes
00:14:bf:b1:97:8d	Yes	Yes	Yes
00:14:bf:b1:97:81	Yes	Yes	Yes

Table 1: Identification of MAC Addresses Including in the Training Datasets

The table above summarizes which MAC addresses were included in which training dataset and model pairs. The left hand column shows the MAC addresses of each access point. The other columns show whether that access point was included in a particular model.

3. Analysis

We used a K-Nearest Neighbor (KNN) algorithm to build the indoor positioning system model. A KNN model is a classification model that works by finding the set of k vectors in the training data that are closest in distance to the test point being classified. Once the vectors in the training data are identified, the point classification is determined by the classification of the majority of the k training vectors. The purpose of this KNN analysis is to predict the location of a device based on the signal strength between that same device and the access points in the building (Nolan and Temple Lang). The data discussed thus far in this analysis (referred to as the offline data) is the training data; the text introduces a new dataset on page 31 for testing that is referred to as the online data. The test data consists of signal strength measurements, locations, and device orientations for 60 additional locations on the floor.

As mentioned in section 2: project approach, there are three different scenarios of training datasets are used to analyses which access point helps the most to predict the location of a new device.

Scenario 1:	Keep access point with mac:00:0f:a3:39:e1:c0 and eliminate data with mac:00:0f:a3:39:dd:cd
Scenario 2:	Eliminate access point with mac:00:0f:a3:39:e1:c0 and keep the data with mac:00:0f:a3:39:dd:cd
Scenario 3:	Keep both access points

To determine which scenario resulted in the best model, we created KNN models from the data using a range of values for the K parameter and evaluating them with 11-fold cross-validation. We felt it was important to use a range for the K-parameter in order to ensure the scenario that performed best was not chosen based on an arbitrary model choice. The evaluation criteria examined to compare the model output was the sum of the squared deviations from the actual positions (also known as the sum of squared errors) from each fold of the cross validation of the models. By selecting the model with the minimum sum of squared errors we will be selecting the model that predicted the location of the test points most closely to their accurate values. The results of these cross-validation exercises are included in Figure 3.

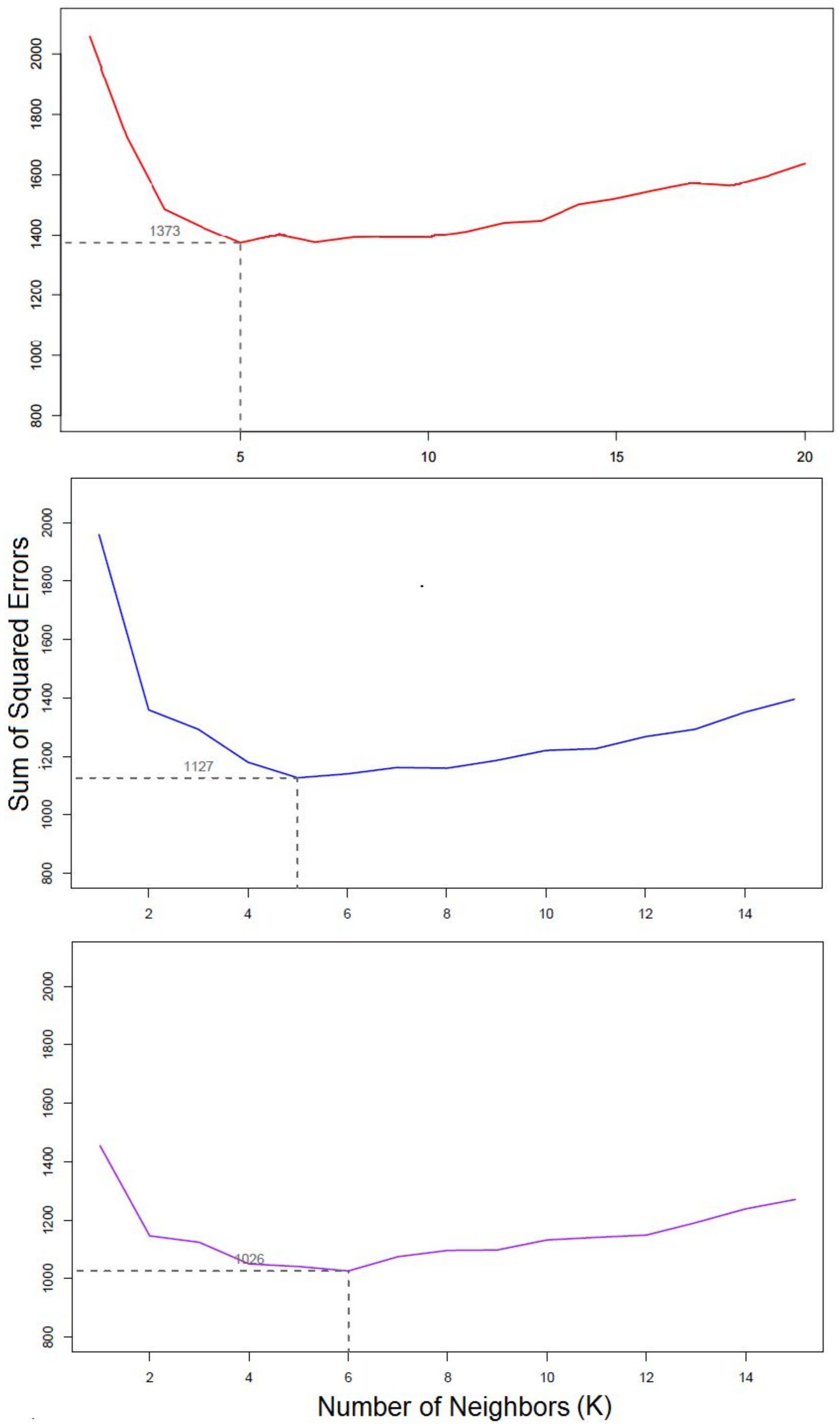

Figure 3: Results of Model Cross Validation

Figure 3: This figure shows the cross-validation results from the array of models created for each scenario. The x-axis of each plot shows the K-parameter of the K-nearest neighbors model, and y-axis shows the resulting sum of squared errors of the each fold of the 11-fold cross validation. In each plot the minimum sum of squared errors is shown for comparison to the other scenarios. The plot in red at the top of the figure shows the original test data used in the text, the plot in blue in the middle shows the scenario where the data eliminated access point has been substituted for the data from the one that was kept, and the plot in purple at the bottom shows the scenario where data from both access points were used. The lowest sum of squared errors among the three scenarios, 1026, was achieved by using the data set with both access points and a K-parameter of 6.

Scenario	Description	Minimum Sum of Squared Errors	K Parameter
1	Discarded data from access point 00:0f:a3:39:dd:cd	1373	5
2	Discarded data from access point 00:0f:a3:39:e1:c0	1127	5
3	Kept data from both access points	1026	6

Table 2: Results of Cross-Validation Exercise

The table above summarizes the results of the cross validation process for each scenario.The third column shows the minimum sum of squared errors for each. The fourth column indicates the most optimal k for each training dataset.

After finding the optimal k via cross-validation for each training dataset, we decided to use k=5 to test all three models using the test data set. Although the minimum sum of squared errors was at K=6 for scenario 3, the value at K = 5 was nearly identical, and we felt it was important to compare models with the same K parameter. The error plots from these final predictions are shown below in Figure 4.

The calculated sum of squared errors for Scenario 1 when K equaled 5 was 276.0, and the sum of square errors for Scenario 2 when K equaled 5 was 249. The sum of squared errors for Scenario 3 when K equaled 5 was the lowest, 210.0.

The lowest error total is for the model trained with data from the highest number of access points (scenario 3). This makes sense because there are more measurements in the training dataset from which to classify the test data set. Therefore, the model in scenario 3 is the most accurate.

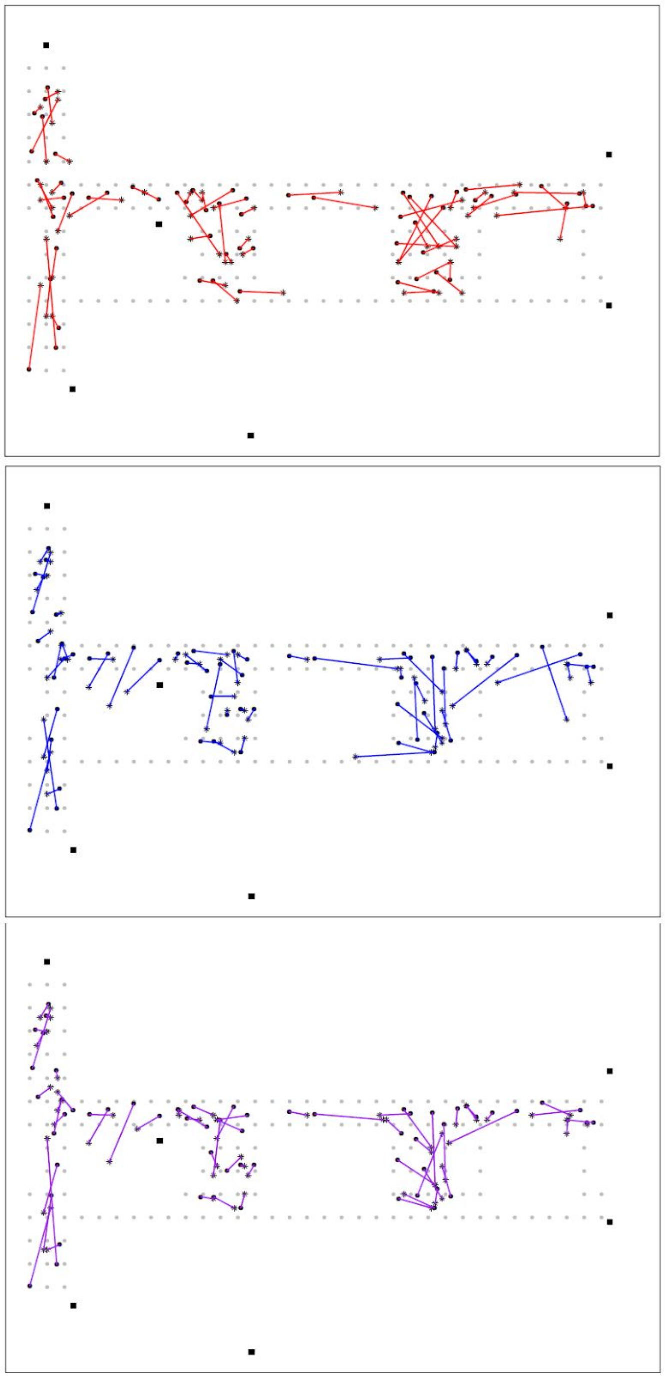

Figure 4: Error Plots For K = 5 Models

Figure 4: This figure shows the prediction results in each scenario for the K-nearest neighbors model with K=5 . Each plot's coordinates mimic the x and y axis of the floorplan. The colored line segments show the error in the predicted locations (the asterisks) and the actual location (the small black squares). The plot in red at the top of the figure shows the original test data used in the text, the plot in blue in the middle shows the scenario where the data eliminated access point has been substituted for the data from the one that was kept, and the plot in purple at the bottom shows the scenario where data from both access points were used.

Conclusion

Based on our chosen evaluation criteria, the models created with data from both access points in the center of the building were more accurate than the two scenarios with only one of the two access points at for all values of K. Surprisingly, between the two scenarios with only one access point, the scenario with the access point that resulted in weaker signal strength measurements (00:0f:a3:39:dd:cd) resulted in better prediction performance for every value of K. This is somewhat counterintuitive, and although we cannot be sure of the mechanism of this increased performance, we theorized that the lower signal strength resulted in fewer multipath readings (defined as bounces off of walls and floors) or other signal interference.

The additional data not only resulted in an improvement in accuracy, but also performed better at lower values of K. This may be an important consideration depending on the application of the indoor positioning system. If lower values of K allow shorter processing time between measurement and location estimation, it may be a better design choice than a value of K that is more accurate but cannot estimate as quickly. Application specific testing of the system should be performed to determine if this is necessary.

Because IoT data collection and resulting models generally require real-time or near real-time and frequent analyses, the model design and application should be scrutinized carefully. Future research with this dataset could focus on isolating hardware and software variables while running the analyses to determine the speed and memory usage for varying designs.

References

1. Nolan, Deborah and Duncan Temple Lang. *Data Science in R: A Case Study Approach to Computational Reasoning and Problem Solving.*
2. McGee, Monnie. R code provided to class via 2U.
3. R: K-Nearest Neighbour Classification. (n.d.). Retrieved June 18, 2016, from https://stat.ethz.ch/R-manual/R-devel/library/class/html/knn.html

```
Code Appendix
#  Unit 6 Case Study - Predicting Location Via Indoor Positioning Systems and
KNN
# June 22, 2016
# Carl Bryant, Sujatha Chinnappa, Carrie Garcia, Brian Mente
#
#  This code processes the data according to the procedures given in Chapter 1
#  of the Nolan Temple Lang Text.  Three versions of this code were used to
#  create each scenario compared in the paper.  At various points the code
#  where each version differed, the differences are called out in comments
#  below to show the reader the variation in versions, and color coded
#  according to the plots in the paper.

#  Only code relevant to the graphical outputs and results presented in the
#  paper above is included in the following code.

options(digits = 2)

# This readData function processes the offline training data
readData =
  function(filename = 'offlinefinaltrace.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")

    # 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)
  }

offline = readData()

summary(offline$signal)



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
         })


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

# Scenario 1 version (graph showing MAC address "00:0f:a3:39:e1:c0") :
oneAPAngle = subset(offlineSummary,
                    mac == subMacs[1] & angle == 0)
# Scenario 2 version (graph showing MAC address "00:0f:a3:39:dd:cd"):
#oneAPAngle = subset(offlineSummary,
#                    mac == subMacs[2] & 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)


# Scenario 1 version (removing mac address "00:0f:a3:39:dd:cd") :
offlineSummary = subset(offlineSummary, mac != subMacs[2])
# Scenario 2 version (removing mac address "00:0f:a3:39:e1:c0") :
# offlineSummary = subset(offlineSummary, mac != subMacs[1])
# Scenario 3 version (keeping all MAC addresses):
# offlineSummary = offlineSummary

# Scenario 1 & 2 version:
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") ))
# Scenario 3 version:
#AP = matrix( c( 7.5, 6.3,7.5, 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
```

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

offlineSummary$dist = sqrt(diffs[ , 1]^2 + diffs[ , 2]^2)

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

macs = unique(offlineSummary$mac)
online = readData("onlinefinaltrace.txt", subMacs = macs)

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

length(unique(online$posXY))

tabonlineXYA = table(online$posXY, online$angle)
tabonlineXYA[1:6, ]

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)

names(onlineSummary)
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 ]
}
angles = angles + nearestAngle
angles[angles < 0] = angles[ angles < 0 ] + 360
angles[angles > 360] = angles[ angles > 360 ] - 360

offlineSubset =
  offlineSummary[ offlineSummary$angle %in% angles, ]

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")

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)

length(train130[[1]])

# Scenario 1 & 2 version:
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 ])
}

# Scenario 3 version (added extra column to account for additional address)
#findNN = function(newSignal, trainSubset) {
#  diffs = apply(trainSubset[ , 4:10], 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)
}
# Scenario 1 & 2 version:
estXYk5 = predXY(newSignals = onlineSummary[ , 6:11],
                  newAngles = onlineSummary[ , 4],
                  offlineSummary, numAngles = 3, k = 5)
# Scenario 3 version (added extra column to account for additional address)
# estXYk5 = predXY(newSignals = onlineSummary[ , 6:12],
#                  newAngles = onlineSummary[ , 4],
#                  offlineSummary, numAngles = 3, k = 5)


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,
           # Scenario 1 Version:
           col='red')
           # Scenario 2 Version:
           #col='blue')
           # Scenario 3 Version:
           #col='purple')

}

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


pdf(file="GEO_FloorPlanK5Errors.pdf", width = 10, height = 7)
oldPar = par(mar = c(1, 1, 1, 1))
floorErrorMap(estXYk5, onlineSummary[ , c("posX","posY")],
              trainPoints = trainPoints, AP = AP)
par(oldPar)
dev.off()

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

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


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)
}

# Scenario 1 Version (removing mac address "00:0f:a3:39:dd:cd"):
offline = offline[ offline$mac != "00:0f:a3:39:dd:cd", ]
# Scenario 2 Version  (removing mac address "00:0f:a3:39:e1:c0"):
# offline = offline[ offline$mac != "00:0f:a3:39:e1:c0", ]
# Scenario 3 Version (keeping all MAC addresses):
# offline = offline[ offline$mac , ]

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])

# Scenario 1 & 2 Version:
estFold = predXY(newSignals = onlineFold[ , 6:11],
                 newAngles = onlineFold[ , 4],
                 offlineFold, numAngles = 3, k = 3)

# Scenario 3 Version (added extra column to account for additional MAC
address):
#estFold = predXY(newSignals = onlineFold[ , 6:12],
#                 newAngles = onlineFold[ , 4],
#                 offlineFold, numAngles = 3, k = 3)


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

K = 15
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) {
    # Scenario 1 & 2 Version:
    estFold = predXY(newSignals = onlineFold[ , 6:11],
                     newAngles = onlineFold[ , 4],
                     offlineFold, numAngles = 3, k = k)
    # Scenario 3 Version (added extra column to account for additional MAC
address):
    #estFold = predXY(newSignals = onlineFold[ , 6:12],
    #                 newAngles = onlineFold[ , 4],
    #                 offlineFold, numAngles = 3, k = k)
    err[k] = err[k] + calcError(estFold, actualFold)
  }
}

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(800, 2100),
     xlab = "Number of Neighbors",
     ylab = "Sum of Square Errors",
     # Scenario 1 Version:
     col='red')
     # Scenario 2 Version:
     #col='blue')
     # Scenario 3 Version:
     #col='purple')

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 = 600,  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()

#Scenario 1 & 2 Version
estXYk5 = predXY(newSignals = onlineSummary[ , 6:11],
                 newAngles = onlineSummary[ , 4],
                 offlineSummary, numAngles = 3, k = 5)
# Scenario 3 Version (added extra column to account for additional MAC
address):
# estXYk5 = predXY(newSignals = onlineSummary[ , 6:12],
#                  newAngles = onlineSummary[ , 4],
#                  offlineSummary, numAngles = 3, k = 5)

calcError(estXYk5, actualXY)

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)}
```