Real Time Location Using KNN

Ananth Subramanian, Vitaly Briker,

Jackson Au & Richard Farrow

September 8, 2019

Abstract

Real Time location Services (RTLS) technology can be used to determine the location of objects in real time and is used across various industries including manufacturing, retail, healthcare, and many more. This case study covers the methodologies adopted in predicting location using K-Nearest Neighbors' principles. In addition to using K-Nearest Neighbor, a Weighted Average method is also studied and compared.

1. Introduction

K-Nearest Neighbor (KNN) is a non-parametric supervised learning technique that is often used for classification and regression. Since KNN does not employ a training phase in its predictions it is referred to a "lazy algorithm." When applying KNN, a classifier is developed to assign class label from an existing set to an unknown variety based on the class of its nearest neighbors. This class membership is determined by proximity, which can be calculated using a variety of distance functions. Few of these functions include 1. Euclidean 2. Manhattan 3. Minkowski. The mathematics supporting these functions are out of scope for this study; however, additional details can be obtained from machine learning websites.

The data provided for this case study include, among others, the signal strengths captured by a hand-held device. The signals emanate from readers or access points at vantage positions within a selected floor of a university building. The signals are measured at various positions and orientations of the hand-held device. The data study warrants some knowledge in the field of telecommunication as signal strength needs to be analyzed at various positions of the hand-held device to know how different parameters like distance and positions impact the signal reach.

Each of the readers or access points are associated with a Media Access Control (MAC) address. Even though there are six (6) access points/MAC addresses, the data covers measurements from a few hidden access points, which are discovered by analyzing the data in detail.

This case study provides a methodology to determine an unknown location of a hidden device based on the signal strength measured in that position from different access points. KNN is used to arrive at the location based on the signal strengths of the nearby locations as provided in the offline data. A comparison is also done to find out which MAC addresses provide a better prediction.

The analysis was completed using R programming and selected custom and standard functions for data cleansing and manipulations.

2. Methods

Offline data was read into memory using R in order to manipulate the data for the purposes of this analysis. Data manipulation includes, among others, adding appropriate and meaningful feature names (e.g., time, signal, mac-address, positions, orientation, etc.). In addition, the angle of orientation is rounded to the nearest whole number to align the data with the description provided in the study. Functions are defined for optimal reuse and readability.

The core of the analysis was to explore signal strengths from various access points and to select MAC addresses that contribute to signal strength measurements. Eliminating access points that reported very few observations, the group deduced the number of Mac addresses to seven (7) from a list of 166 locations in the building floor plan.

All created functions were summarized and entered within a custom function for future use. Signal strengths between MAC addresses and access points were plotted using heat maps. Finally, a matrix was generated recording positions of MAC addresses and calculated distances at the positions emitting a signal.

The location of a new observation is predicted using KNN. Using signal strengths from different access points, the signal strength of the new observation is compared with that of the signal strength of known location and based on the closeness of the value of the signal strength the location of the new observation is arrived at.

The formula to calculate proximity is Euclidean distance as shown below (1)

$$\sqrt{(S_{1*} - S_1)^2 + \ldots + (S_{6*} - S_6)^2} \tag{1}$$

“where S_i is the signal strength between the hand-held device and the i-th access point for a training observation, while S_{i*} is the signal strength between the same access point and the new observation whose (x,y) position we are trying to predict.[1]”

It is important to be wary of the orientation of the new observation. The rounding function is utilized to round-off the orientation of the new observation and another custom function is made use of to arrive at the prediction of the location and this is based on number of nearest angles that are provided.

3. Results

The analysis was split into three (3) cases as described below:

	Case 1 Including all available devices	**Case 2** excluding 00:0f:a3:39:dd:cd	**Case 3** excluding 00:0f:a3:39:e1:c0
Common addresses for all models	00:14:bf:3b:c7:c6 00:14:bf:b1:97:81 00:14:bf:b1:97:8a 00:14:bf:b1:97:8d 00:14:bf:b1:97:90	00:14:bf:3b:c7:c6 00:14:bf:b1:97:81 00:14:bf:b1:97:8a 00:14:bf:b1:97:8d 00:14:bf:b1:97:90	00:14:bf:3b:c7:c6 00:14:bf:b1:97:81 00:14:bf:b1:97:8a 00:14:bf:b1:97:8d 00:14:bf:b1:97:90
Additional MAC addresses	00:0f:a3:39:dd:cd 00:0f:a3:39:e1:c0	00:0f:a3:39:e1:c0	00:0f:a3:39:dd:cd

Table 1: Cases studied

Model 1: Predicting location of mobile devices using different combinations of MAC addresses. Calculating estimated location using simple average of the distances between two sets of signal strengths with Euclidean distance. Figures 1.1-1.3 show a sum of square error as a function of neighbors.

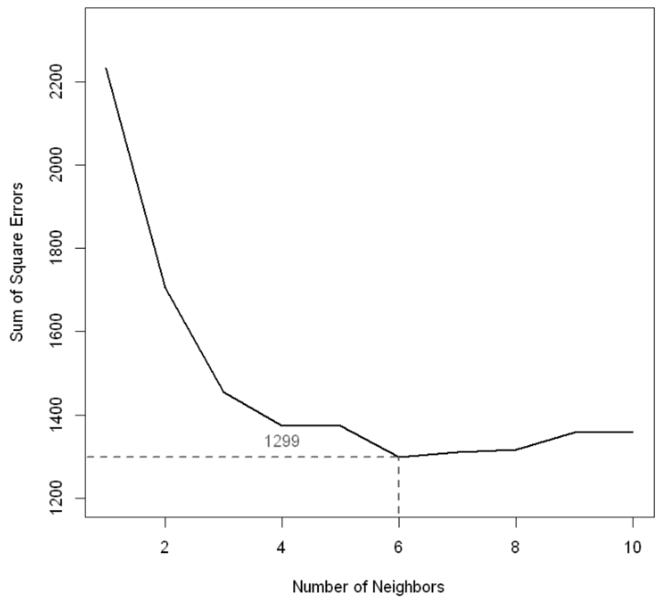


Figure 1.1: Case 1 - Cross Validated Selection of *k*

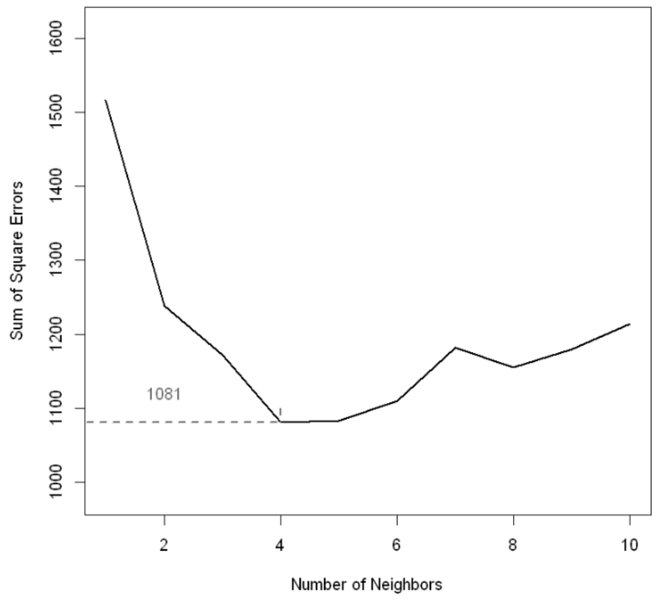


Figure 1.2: Case 2 - Cross Validated Selection of *k*

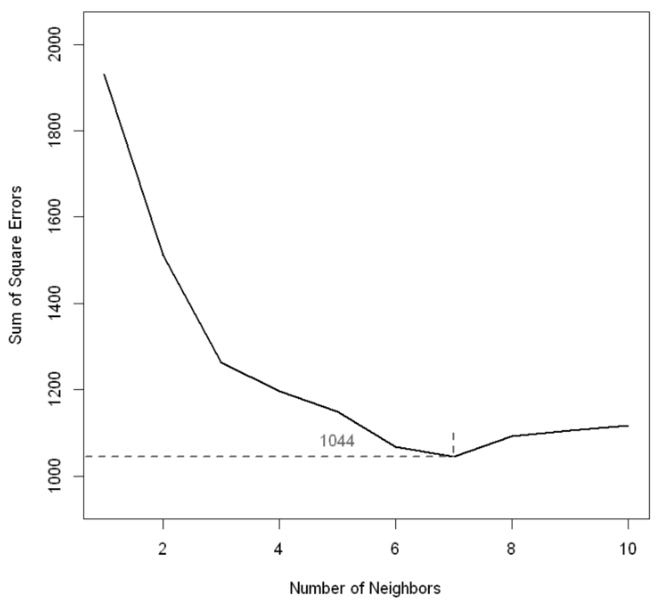


Figure 1.3: Case 3 - Cross Validated Selection of *k*

Based on cross validation result the optimal k value obtained using offline data.

Case	K value	Number of Angles	Estimated error
1	1	3	478.64
	3	3	249.43
	6	3	**211.17**
2	1	3	659.40
	3	3	302.45
	4	3	**301.57**
3	1	3	411.64
	3	3	270.49
	7	3	**258.07**

Table 2: Estimated model error

Based on the results from table 2, MAC 00:0f:a3:39:dd:cd gives slightly better results than MAC 00:0f:a3:39:e1:c0. The results also show that by using all 7 access points gives the lowest calculated error.

Model 2: Predicting location of mobile device using the following MAC addresses. Calculating estimated location using averaged weight that is inversely proportional to the distance (in signal strength) from the test observation. Objective is to increase effect of closer neighbors and improve the accuracy of the results. The weighting factor is calculated as shown below (2).

$$\frac{1/d_i}{\sum_{i=1}^{k} 1/d_i} \qquad (2)$$

The plots show a sum of square error of weighed average model as a function of neighbors.

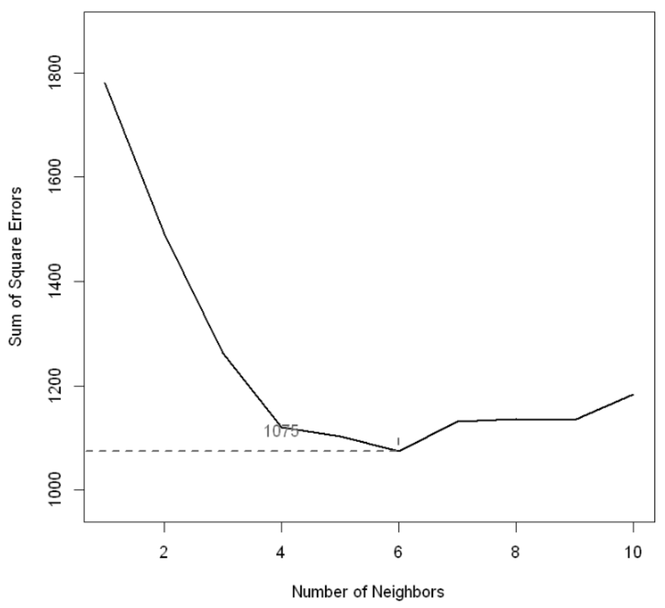


Figure 2.1: Case 1 - Cross Validated Selection of *k*

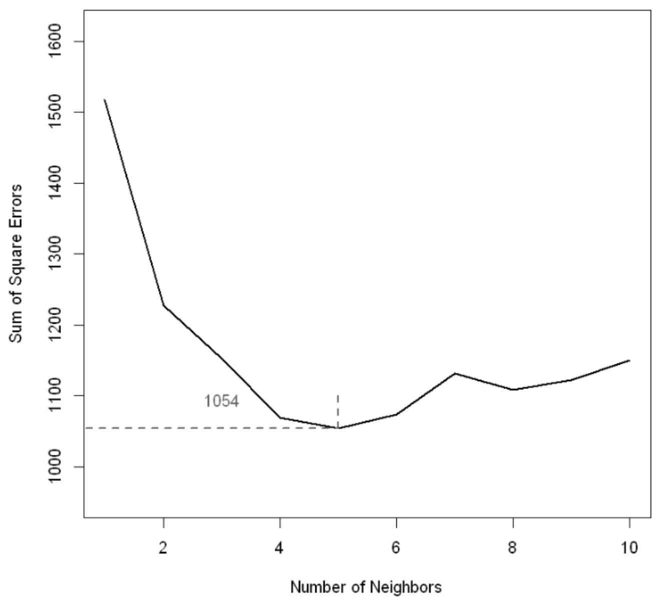


Figure 2.2: Case 2 - Cross Validated Selection of *k*

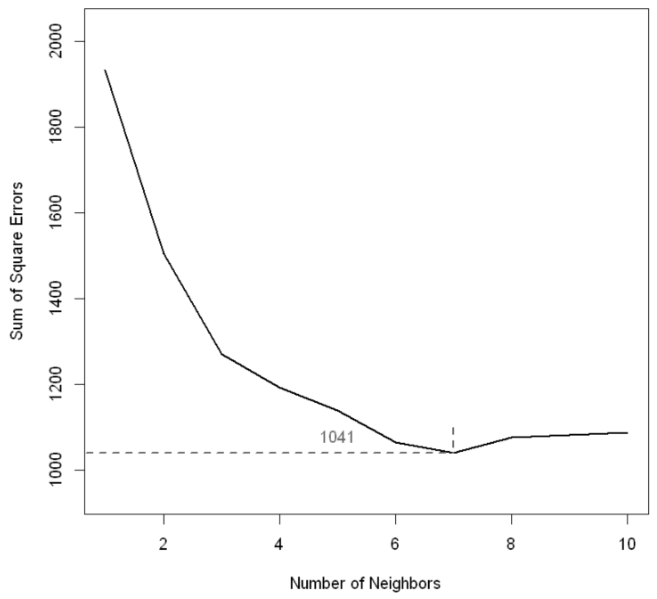


Figure 2.3: Case 3 - Cross Validated Selection of *k*

Based on cross validation result the optimal k values obtained using offline data.

Case	K value	Number of Angles	Estimated error
1	1	3	478.64
	3	3	249.97
	6	3	**207.25**
2	1	3	659.40
	3	3	302.46
	5	3	**273.12**
3	1	3	411.64
	3	3	267.11
	7	3	**248.35**

Table 3: Estimated model error using weighting factor

Based on the results from Table 3, MAC 00:0f:a3:39:dd:cd gives slightly better results than MAC 00:0f:a3:39:e1:c0. The results also suggest that using all seven (7) access points' results in the lowest calculated error.

Best performed model	Estimated error
Model 1 (case1) – KNN simple average	**211.17**
Model 2 (case 1) - Weighted Average	**207.25**

Table 4: Estimated error model comparison

From the results included in Table 4, we can see that this alternative model with weighted average offers a more accurate prediction result. Every case shows a benefit of using weighting factor and not a simple average calculation.

4. Conclusion

It appears that using access point with MAC 00:0f:a3:39:dd:cd gives slightly better result than using MAC 00:0f:a3:39:e1:c0. Moreover the analysis reveals that using all seven (7) access point provides better accuracy. The analysis also indicate that k-value in range 5-7 neighbors provides better accuracy for both models.

The alternative approach using weighted average inversely proportional to distance provides most accurate result but the difference between two models is marginal.

The implementation of our internal positioning system will depend on what is available to our disposal. If our priority is to have the best predictive ability for indoor tracking and our budget allows for us to move forward with both MAC addresses, it would make sense to include both because that produces the best signal strength at each location.

5. References

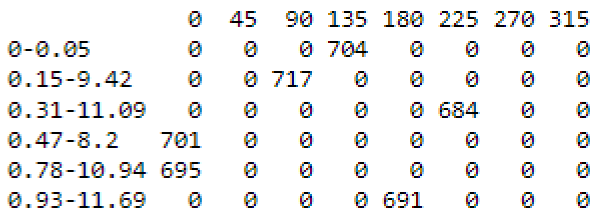
1. Nolan, D. and Lang, D. T. “Data Science in R.” CRC Press, 2015 (Chapter 1).

6. Appendix – Code

```
# Function to read the data file and pull necessary information in right format
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)
    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)
  }
################################################################################
# put it all together to process a line as a function
# note that the if statement handles null values to remove warnings
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)
}

# create a function that will round off to the nearest major angle
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]
}
#set path to the file, the data file should be in the folder as a script
setwd(dirname(rstudioapi::getSourceEditorContext()$path))
#read the data file
offline = readData()

#Setup all the data using the data summary
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
        })
#concatanate rows
offlineSummary = do.call("rbind", signalSummary)
#create vector of all MACS
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")

#create new variable that holds length of the MAC number
mac_number=length(subMacs)

# signal strength, get online data matches offline
macs = unique(offlineSummary$mac)
online = readData("online.final.trace.txt", subMacs = macs)
online$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$mac, mean)
                  print(avgSS)
                  y = matrix(avgSS, nrow = 1, ncol = mac_number,
                             dimnames = list(ans$posXY, names(avgSS)))
                  cbind(ans, y)
                }))
```

```
#position range
onlineSummary = do.call("rbind", byLoc)
```



```
               0  45  90 135 180 225 270 315
0-0.05         0   0   0 704   0   0   0   0
0.15-9.42      0   0 717   0   0   0   0   0
0.31-11.09     0   0   0   0   0 684   0   0
0.47-8.2     701   0   0   0   0   0   0   0
0.78-10.94   695   0   0   0   0   0   0   0
0.93-11.69     0   0   0   0 691   0   0   0
```

```
# new variable holds the maximum column number, 12 - all Macs, 11 – other 2 cases
max_col=ncol(onlineSummary)


# new variable holds the angles number trough the analysis
numAngles=3


# create data frame and functions to aggregate/select data with similar angles
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, ]
```

```
60  12

'posXY'  'posX'  'posY'  'orientation'  'angle'  '00:0f:a3:39:dd:cd'  '00:0f:a3:39:e1:c0'  '00:14:bf:3b:c7:c6'  '00:14:bf:b1:97:81'  '00:14:bf:b1:97:8a'
'00:14:bf:b1:97:8d'  '00:14:bf:b1:97:90'
```

```
#only angles 225+-45

unique(offlineSubset$angle)


#convert offline data by taking a mean

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 = mac_number,

                               dimnames = list(ans$posXY,

                                               names(avgSS)))

                    cbind(ans, y)

                  }))


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

    show(newDataSS)

  return(newDataSS)

}


#trainSS - function converts offline to match online table

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")
}

# NN function no weight
findNN = function(newSignal, trainSubset) {
  diffs = apply(trainSubset[ , 4:(max_col-2)], 1,
                function(x) x - newSignal)
  dists = apply(diffs, 2, function(x) sqrt(sum(x^2)) )
  closest = order(dists)
  return(trainSubset[closest, 1:3 ])
}

# NN function with weight calculation
findNNW = function(newSignal, trainSubset) {
  diffs = apply(trainSubset[ , 4:(max_col-2)], 1,
                function(x) x - newSignal)
  dists = apply(diffs, 2, function(x) sqrt(sum(x^2)) )
     # order by distance
 closest = order(dists)
     # Position index
    closeXY = trainSubset[closest, 1:3 ]
     #1/distance
    weight1d = as.numeric(1/dists[closest])
   return(cbind(closeXY, weight1d))
}

# predict Weighted Average X-Y based on the the nearest k neighbors (default 3)
predXYW = 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)



      weight1d =
          findNNW(newSignal = as.numeric(newSignals[i, ]), trainSS)
    weight = weight1d[1:k, 4]/sum(weight1d[1:k, 4])
# add zeros to more than k members
    weight= append(weight,replicate(  nrow(weight1d)-k, 0))
# multiply position  by weight
    weight1d[ , 2:3] = weight1d[ , 2:3] * weight
    closeXY[[i]] = weight1d[ ,1:3]
  }
  estXY = lapply(closeXY,
                 function(x) sapply(x[ , 2:3],
                                    function(x) sum(x)))
  estXY = do.call("rbind", estXY)
return(estXY)
}

# predict X-Y based on the nearest k neighbors (default 3)
predXY = function(newSignals, newAngles, trainData,
                  numAngles = 1, k = 3){
  closeXY = list(length = nrow(newSignals))

  for (i in 1:nrow(newSignals)) {
# get all data matches the angle
    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],
#average of k locations
                    function(x) mean(x[1:k])))
        estXY = do.call("rbind", estXY)
  return(estXY)
}
```

```
# calculation of nearest 1 and 3 neighbors
estXYk3 = predXY(newSignals = onlineSummary[ , 6:max_col],
                 newAngles = onlineSummary[ , 4],
                 offlineSummary, numAngles = numAngles, k = 3)
estXYk1 = predXY(newSignals = onlineSummary[ , 6:max_col],
                 newAngles = onlineSummary[ , 4],
                 offlineSummary, numAngles = numAngles, k = 1)

# Error calculation 1 and 3 neighbors
calcError =
  function(estXY, actualXY) {
     #show(estXY )
     #show(actualXY)
     return (sum( rowSums( (estXY - actualXY)^2) ))
}
actualXY = onlineSummary[ , c("posX", "posY")]
sapply(list(estXYk1, estXYk3), calcError, actualXY)
```

```
478.6403 249.426966666667
```

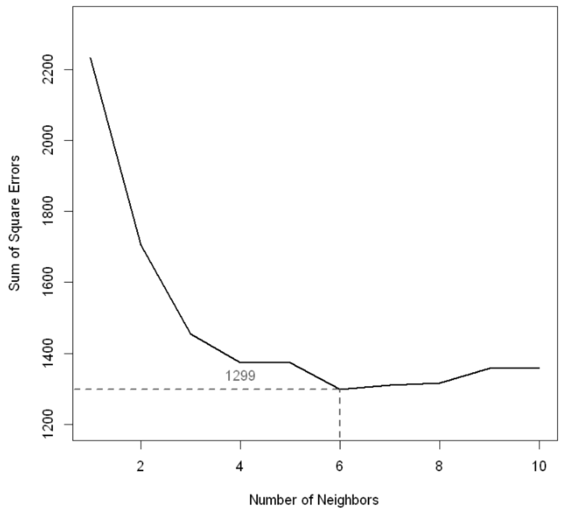
```
# calculation nearest 1 and 3 neighbors with weight factor
estXYk3 = predXYW(newSignals = onlineSummary[ , 6:max_col],
                 newAngles = onlineSummary[ , 4],
                 offlineSummary, numAngles = numAngles, k = 3)
estXYk1 = predXYW(newSignals = onlineSummary[ , 6:max_col],
                 newAngles = onlineSummary[ , 4],
                 offlineSummary, numAngles = numAngles, k = 1)
# Error calculation for weighted average 1 and 3 neighbors
calcError =
  function(estXY, actualXY) {
     return (sum( rowSums( (estXY - actualXY)^2) ))
}
actualXY = onlineSummary[ , c("posX", "posY")]
sapply(list(estXYk1, estXYk3), calcError, actualXY)
```

```
478.6403   249.971581430096
```

```
# adding 11 folds
v = 11
permuteLocs = sample(unique(offlineSummary$posXY))
permuteLocs = matrix(permuteLocs, ncol = v,
                     nrow = floor(length(permuteLocs)/v))
onlineFold = subset(offlineSummary, posXY %in% permuteLocs[ , 1])
reshapeSS1 = function(data, varSignal = "signal",
                     keepVars = c("posXY", "posX","posY"),
                     sampleAngle = FALSE,
                     refs = seq(0, 315, by = 45)) {
  byLocation =
    with(data, by(data, list(posXY),
                  function(x) {
                    if (sampleAngle) {
                      x = x[x$angle == sample(refs, size = 1), ]}
                    ans = x[1, keepVars]
                    avgSS = tapply(x[ , varSignal ], x$mac, mean)
                    y = matrix(avgSS, nrow = 1, ncol = mac_number,
                               dimnames = list(ans$posXY,
                                               names(avgSS)))
                    cbind(ans, y)
                  }))
  newDataSS = do.call("rbind", byLocation)
  return(newDataSS)
}
# Due to long run time examine up to 10 neighbors, 11 folds
keepVars = c("posXY", "posX","posY", "orientation", "angle")
onlineCVSummary = reshapeSS1(offline, keepVars = keepVars,
                            sampleAngle = TRUE)
onlineFold = subset(onlineCVSummary,
                    posXY %in% permuteLocs[ , 1])
offlineFold = subset(offlineSummary,
                     posXY %in% permuteLocs[ , -1])

estFold = predXY(newSignals = onlineFold[ , 6:max_col],
                 newAngles = onlineFold[ , 4],
                 offlineFold, numAngles = numAngles, k = 3)
```

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

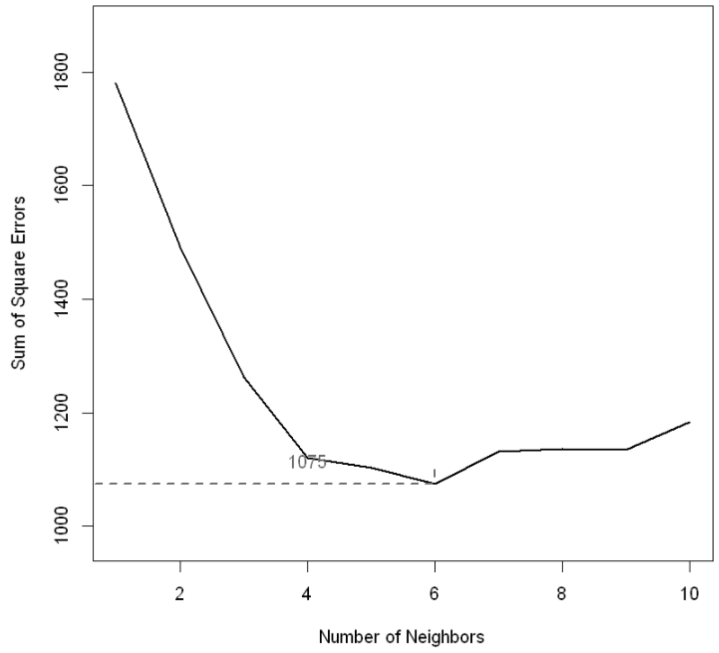
```
#cross validation calculation
K = 10
err = rep(0, K)

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:max_col],
                     newAngles = onlineFold[ , 4],
                     offlineFold, numAngles = numAngles, k = k)
    err[k] = err[k] + calcError(estFold, actualFold)
  }
}
#plot cross validation results
plot(y = err, x = (1:K),  type = "l", lwd= 2,
     ylim = c(min(err)-100, max(err)+100),
     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))

# all MAC no weight K optimal
kMin
6
# Error calculation for weighted average with optimal k
estXYk_best = predXY(newSignals = onlineSummary[ , 6:max_col],
                 newAngles = onlineSummary[ , 4],
                 offlineSummary, numAngles = numAngles, k = kMin)

allMac_Best=calcError(estXYk_best, actualXY)

# all MAC no weight
allMac_Best
211.173633333333

# same analysis adding weight factor
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 = predXYW(newSignals = onlineFold[ , 6:max_col],
                     newAngles = onlineFold[ , 4],
                     offlineFold, numAngles = numAngles, k = k)
    err[k] = err[k] + calcError(estFold, actualFold)
  }
}
kMin
7
```

#Plot CV result

```
plot(y = err, x = (1:K),  type = "l", lwd= 2,
     ylim = c(min(err)-100, max(err)+100),
     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))
```

# Error calculation with weight factor

```
estXYk_best = predXYW(newSignals = onlineSummary[ , 6:max_col],
                  newAngles = onlineSummary[ , 4],
                  offlineSummary, numAngles = numAngles, k = kMin)
```

```
allMac_Best_weight=calcError(estXYk_best, actualXY)


# all MAC with weight
allMac_Best_weight
```

207.250770141227

```
########### finish analysis for all MAC case


######  00:0f:a3:39:e1:c0  ########### remove 00:0f:a3:39:dd:cd
# making a copy of offlineSummary , can reuse later
offlineSummary_copy=offlineSummary
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")
#remove MAC from data
offlineSummary = subset(offlineSummary_copy, mac !=subMacs[2])
mac_number = length(unique(offlineSummary$mac))


# signal strength matching offline table
macs = unique(offlineSummary$mac)
online = readData("online.final.trace.txt", subMacs = macs)
online$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$mac, mean)
                  print(avgSS)
                  y = matrix(avgSS, nrow = 1, ncol = mac_number,
                             dimnames = list(ans$posXY, names(avgSS)))
                  cbind(ans, y)
                }))
onlineSummary = do.call("rbind", byLoc)
```

```
#create new variable that holds maximum number of columns
 max_col=ncol(onlineSummary)
 max_col
 11

 # Error calculation 1 and 3 neighbors

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

 estXYk1 = predXY(newSignals = onlineSummary[ , 6:max_col],
                  newAngles = onlineSummary[ , 4],
                  offlineSummary, numAngles = numAngles, k = 1)
 calcError =
   function(estXY, actualXY)
     sum( rowSums( (estXY - actualXY)^2) )

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

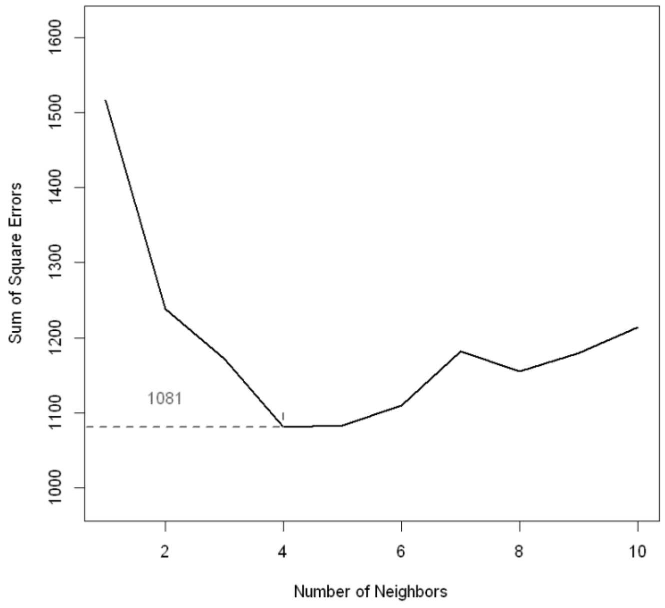
659.4003   306.702522222222

```
 # Error calculation for weighted average 1 and 3 neighbors
 estXYk3 = predXYW(newSignals = onlineSummary[ , 6:max_col],
                  newAngles = onlineSummary[ , 4],
                  offlineSummary, numAngles = numAngles, k = 3)

 estXYk1 = predXYW(newSignals = onlineSummary[ , 6:max_col],
                  newAngles = onlineSummary[ , 4],
                  offlineSummary, numAngles = numAngles, k = 1)
 calcError =
   function(estXY, actualXY)
     sum( rowSums( (estXY - actualXY)^2) )
 actualXY = onlineSummary[ , c("posX", "posY")]
 sapply(list(estXYk1, estXYk3), calcError, actualXY)
```

659.4003 302.456031328407

```
# Fold creation up to 10 neighbors, 11 folds
# create offline_copy for later use
offline_copy=offline
offline = offline_copy[ offline_copy$mac != "00:0f:a3:39:dd:cd", ]
keepVars = c("posXY", "posX","posY", "orientation", "angle")
onlineCVSummary = reshapeSS1(offline, keepVars = keepVars,
                            sampleAngle = TRUE)
onlineFold = subset(onlineCVSummary,
                    posXY %in% permuteLocs[ , 1])
offlineFold = subset(offlineSummary,
                     posXY %in% permuteLocs[ , -1])
estFold = predXY(newSignals = onlineFold[ , 6:max_col],
                 newAngles = onlineFold[ , 4],
                 offlineFold, numAngles = numAngles, k = 3)
# CV calculation
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:max_col],
                     newAngles = onlineFold[ , 4],
                     offlineFold, numAngles = numAngles, k = k)
    err[k] = err[k] + calcError(estFold, actualFold)
  }
}
#Plot result
plot(y = err, x = (1:K),  type = "l", lwd= 2,
     ylim = c(min(err)-100, max(err)+100),
     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))
```



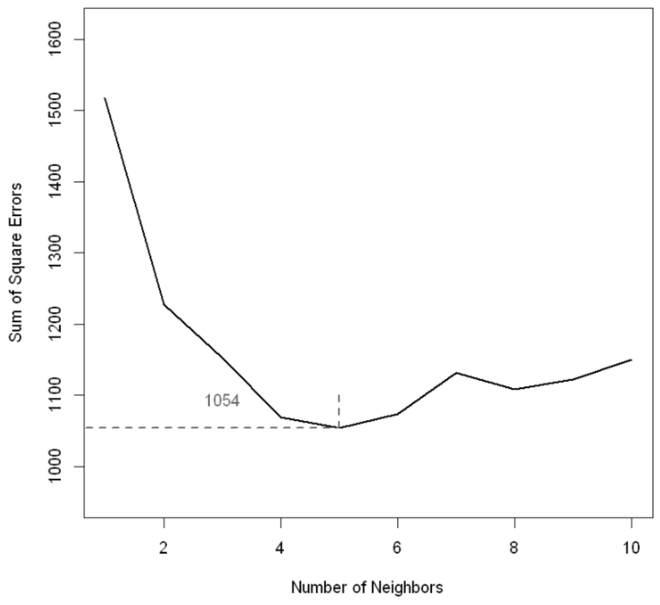
# Error calculation for 00:0f:a3:39:e1:c0

```
estXYk_best = predXY(newSignals = onlineSummary[ , 6:max_col],
                 newAngles = onlineSummary[ , 4],
                 offlineSummary, numAngles = numAngles, k = kMin)

no_cd_mac_best=calcError(estXYk_best, actualXY)
no_cd_mac_best
```

301.5728

# same as above adding weight

```
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 = predXYW(newSignals = onlineFold[ , 6:max_col],
                      newAngles = onlineFold[ , 4],
                      offlineFold, numAngles = numAngles, k = k)
    err[k] = err[k] + calcError(estFold, actualFold)
  }
}
plot(y = err, x = (1:K),  type = "l", lwd= 2,
     ylim = c(min(err)-100, max(err)+100),
     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))
```



```
# Error calculation for 00:0f:a3:39:e1:c0  with weighted factor:
estXYk_best = predXYW(newSignals = onlineSummary[ , 6:max_col],
                 newAngles = onlineSummary[ , 4],
                 offlineSummary, numAngles = numAngles, k = kMin)
no_cd_mac_best_weight =calcError(estXYk_best, actualXY)

no_cd_mac_best_weight
```

273.121241611466

```
########  finish 00:0f:a3:39:e1:c0 analysis ################

######### 00:0f:a3:39:dd:cd #####  remove 00:0f:a3:39:e1:c0
# Copy of offlineSummary for later use
offlineSummary_copy=offlineSummary
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")
# remove 00:0f:a3:39:e1:c0 MAC
offlineSummary = subset(offlineSummary_copy, mac !=subMacs[1])
mac_number = length(unique(offlineSummary$mac))
# Matching online data with offline
macs = unique(offlineSummary$mac)
online = readData("online.final.trace.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)
                  print(avgSS)
                  y = matrix(avgSS, nrow = 1, ncol = mac_number,
                             dimnames = list(ans$posXY, names(avgSS)))
                  cbind(ans, y)
                }))
```

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


max_col=ncol(onlineSummary)
max_col
```
11

```
#Error calculation for 00:0f:a3:39:dd:cd
estXYk3 = predXY(newSignals = onlineSummary[ , 6:max_col],
                 newAngles = onlineSummary[ , 4],
                 offlineSummary, numAngles = numAngles, k = 3)
estXYk1 = predXY(newSignals = onlineSummary[ , 6:max_col],
                 newAngles = onlineSummary[ , 4],
                 offlineSummary, numAngles = numAngles, k = 1)
calcError =
  function(estXY, actualXY)
    sum( rowSums( (estXY - actualXY)^2) )
actualXY = onlineSummary[ , c("posX", "posY")]
sapply(list(estXYk1, estXYk3), calcError, actualXY)
```

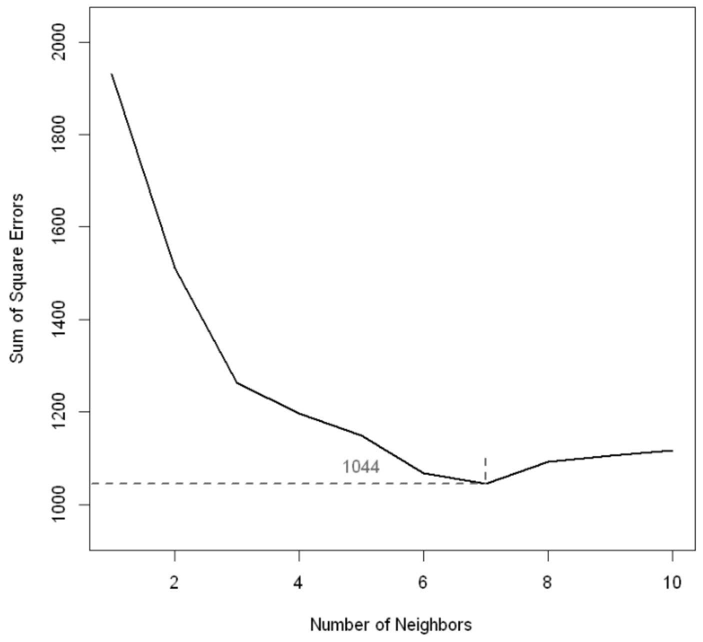
411.6403   270.458077777778

```
# Error calculation for 00:0f:a3:39:dd:cd with weight
estXYk3 = predXYW(newSignals = onlineSummary[ , 6:max_col],
                 newAngles = onlineSummary[ , 4],
                 offlineSummary, numAngles = numAngles, k = 3)
estXYk1 = predXYW(newSignals = onlineSummary[ , 6:max_col],
                 newAngles = onlineSummary[ , 4],
                 offlineSummary, numAngles = numAngles, k = 1)
calcError =
  function(estXY, actualXY)
    sum( rowSums( (estXY - actualXY)^2) )
actualXY = onlineSummary[ , c("posX", "posY")]
sapply(list(estXYk1, estXYk3), calcError, actualXY)
```

411.6403 267.111562167323

```
# Folding creation 11 folds
offline = offline_copy[ offline_copy$mac != "00:0f:a3:39:e1:c0", ]
keepVars = c("posXY", "posX","posY", "orientation", "angle")
onlineCVSummary = reshapeSS1(offline, keepVars = keepVars,
                             sampleAngle = TRUE)
onlineFold = subset(onlineCVSummary,
                    posXY %in% permuteLocs[ , 1])
offlineFold = subset(offlineSummary,
                     posXY %in% permuteLocs[ , -1])
estFold = predXY(newSignals = onlineFold[ , 6:max_col],
                 newAngles = onlineFold[ , 4],
                 offlineFold, numAngles = numAngles, k = 3)

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

#CV calculation
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:max_col],
                     newAngles = onlineFold[ , 4],
                     offlineFold, numAngles = numAngles, k = k)
    err[k] = err[k] + calcError(estFold, actualFold)
  }
}
```

```
plot(y = err, x = (1:K),  type = "l", lwd= 2,
     ylim = c(min(err)-100, max(err)+100),
     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))
```



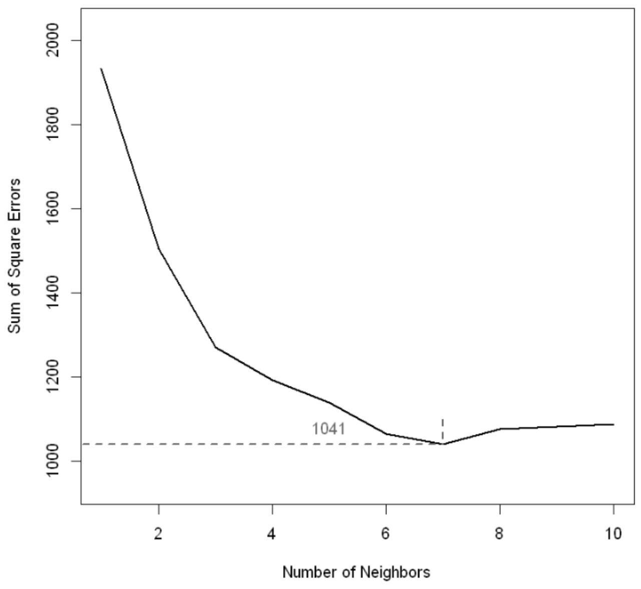
```
kMin
```

7

```
estXYk_best = predXY(newSignals = onlineSummary[ , 6:max_col],
                 newAngles = onlineSummary[ , 4],
                 offlineSummary, numAngles = numAngles, k = kMin)

noc0_mac_best=calcError(estXYk_best, actualXY)
noc0_mac_best
```

258.074993877551

```
# CV calculation adding weight factor
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 = predXYW(newSignals = onlineFold[ , 6:max_col],
                     newAngles = onlineFold[ , 4],
                     offlineFold, numAngles = numAngles, k = k)
    err[k] = err[k] + calcError(estFold, actualFold)
  }
}

plot(y = err, x = (1:K),  type = "l", lwd= 2,
     ylim = c(min(err)-100, max(err)+100),
     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))
```



```
# error calculation with weight factor
estXYk_best = predXYW(newSignals = onlineSummary[ , 6:max_col],
                 newAngles = onlineSummary[ , 4],
                 offlineSummary, numAngles = numAngles, k = kMin)
noc0_mac_weight_best=calcError(estXYk_best, actualXY)
noc0_mac_weight_best
```

248.353546031523

####### finish 00:0f:a3:39:dd:cd analysis ############3