Unit 2 Case Study

MSDS Fall ‘19

7333 Quantify the World

Daniel Serna, Bruce Granger, Herber Nielsen, Brandon de la Houssaye

September 9, 2019

**Abstract**

In this paper we provide a Real Time Location System (RTLS) analysis in order to guide in the build of an indoor positioning system. Positioning systems are not as reliable when the device (such as a cell phone) is within a structure (i.e., indoors). An indoor positioning system could be used in a variety of structures, such as hospitals, travel transit structures (airports, train stations, etc.), museums, and other structures. The benefits of such a system range from increased ability to provide emergency services; crime prevention; business process optimization; social connection; and other related uses. While indoor positioning systems are not necessarily novel in form, this paper does present a model that utilizes k-nearest neighbors with assigned weighting.

**1 Introduction**

Global Positioning Systems (GPS) is a radionavigation system. This system enables the pinpointed location of an object on Earth and it works by the transmittal of signals between a device (receiver) and a number of satellites (orbiting earth). With the satellites having a known location, the time delay of the received signal (from the receiver) is measured. With the signals to the multiple satellites recorded, the relative time differences in the signals (each to a known satellite) is compared to approximate where the receiver is located (by longitude and latitude). i.e., the shorter the time delay of the signal, the nearer the receiver.

The issue with GPS centers around signal disruption. That is, a mountain or a building, can interfere with a signal either by prohibiting the transmission or causing increased delay, thereby decreasing the accuracy of the location. Thus, GPS works quite well in situations where there is unobstructed signal transmission between satellites and a receiver.

However, there is an increased demand and need to have accurate geolocations of devices where there is a high likelihood of obstructions. For example, the ability to pinpoint map a device within a larger distribution center at any given point in time is of great interest to any number of companies looking to increase supply chain efficiency. This gives rise to an opportunity to develop analogous indoor positioning systems; where the output of GPS is still realized by users in an obstructed environment.

In this paper, we discuss such a system – an indoor positioning system – that can meet such a demand in a way that leverages the core concepts of GPS as well as modern network hardware and technology.

Specifically, this paper provides a system that is a Wi-Fi based positioning system (WPS) where signal strength at known MAC addresses is analyzed using a statistical approach called k-nearest neighbor. The remaining sections of this paper cover the data utilized; data cleansing steps; statistical methods applied; and, finally, conclusions. This paper also provides the relevant code and references.

**2 Data**

A floor plan containing eight routers was leveraged. The eight routers collected signals from devices that communicated with the eight routers. This forms the raw data. The eight routers (within the structure) as well as the individual devices were tracked based on exact location using a grid system. Thus, the routers locations were known as well as the individual devices. The signals between the devices and the routers were then collected (where the x,y coordinates were already known) into the raw data.

The raw data considered was provided from the following linked text file:

<http://rdatasciencecases.org/Data/offline.final.trace.txt>

The raw data is organized into lines by router, and the variables in a line are defined as follows:[[1]](#footnote-1)

* t = timestamp in ms since 12:00 am, January 1, 1970 UTC
* id = MAC address of the scanning device (i.e., router)
* pos = physical coordinate of the scanning device
* degree = orientation (direction pointed) of the user carrying the scanning device and expressed in degrees
* MAC = “MAC address of a responding peer (e.g. an access point or a device in adhoc mode) with the corresponding values for signal strength in dBm (Decibel-millwatts), the channel frequency and its mode (access point = 3, device in adhoc mode = 1)”

**3 Data Cleansing**

The raw data contains eight lines of arrays. We separate the information into matrixes and introduce features in order to isolate data for statistical method application.

The first step, after reading in the raw data, is processing the lines of the text file. Once that is done a function is created to remove unwanted or not needed information. Additionally, the function should establish usable names to data of interest and convert time variables to numeric. We also remove unwanted to devices to only keep ‘access point’ devices.

Once those steps are complete, new features are developed to tally signal strength (i.e., calculate mean, median, length, etc.) Additionally, a feature is created to assign one angle (degree) based on an access point.

What remains, after the steps above are complete (using the code provided in Section A of this paper), is a dataset limited to the correct observations (access points by MAC addresses) with data around signal strength and degree of signal.

**4 Methods**

With the data cleansed into a usable format, the team performs two distinct analyses.

The first analysis is centered around MAC address 00:0f:a3:39:dd:cd and MAC address 00:0f:a3:39:31:c0. Both MAC addresses were analyzed in isolation to determine which MAC address should be retained for the system; whether using one or the other (or both) yielded the most accurate prediction of location (for devices). The statistical model used to perform this analysis was k-nearest neighbor.

The second analysis performed was to alter the weighting of the signal strength to account for the distance observed. In this case, the statistical method utilized is still k-nearest neighbor, but the variable is adjusted to account for both signal strength and distance. The team utilized a weighting mechanism whereby the datapoints closest to the access points received greater weight. The team achieved this weight utilizing an inverse distance calculation (1/distance).

**3 Results**

MAC Analysis Results: TBD

For the weighted KNN algorithm, the team did not achieve improved results. The sum of squares error was greater than the original KNN implementation. Figure 1 provides an elbow plot of the original implementation for comparison to Figure 2 which provides an elbow plot of the modified KNN implementation.

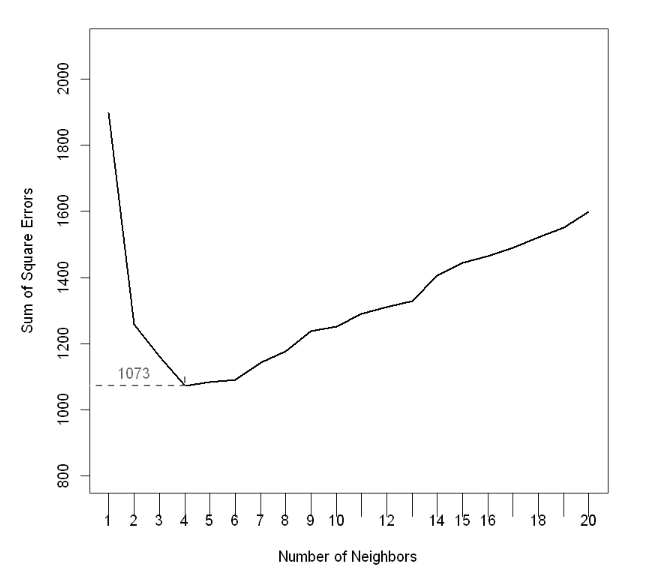


Figure - Original KNN Elbow Plot

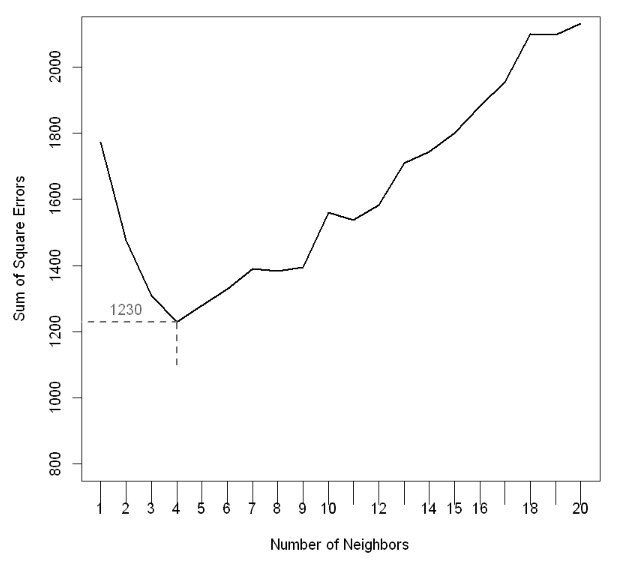


Figure - Weighted KNN Elbow Plot

As can be seen in the above figures, the weighted KNN algorithm had an increased Sum of Squares Error but maintained a similar optimal number of neighbors. For future work, a different weighting algorithm could be tried to achieve better results. Perhaps utilizing radius as a distance metric instead of signal strength could yield improvements.

**4 Conclusion**

Mac Analysis Conclusion: TBD

Our team was unable to improve upon the original KNN implementation. It is possible the signal strength data involved too much variability to be useful as a weighting mechanism. It is also possible we did not utilize this metric in an optimal manner for weighting. Future work is warranted to determine a useful weighting mechanism.

**A Code**

**Unit 2 - Predicting Locations via Indoor Positioning Systems**

**Required Libraries**

In [48]:

**library**(lattice)

**library**(fields)

In [49]:

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")

**1.2 Read Raw Data**

In [50]:

options(digits = 2)

*# read in the entire file into a variable txt*

*# each line will be its own element*

txt = readLines("offline.final.trace.txt")

**Process File**

**Process Lines of file**

In [51]:

*# 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)

}

**Grouping of Angles**

In [52]:

*# 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]

}

**1.3 Cleaning Data & Building Representation for Analysis**

**1.3.4 Creating Function to Prepare the Data**

In [53]:

*# re do our data read combining all the anlysis we did here there are 7 SEVEN macids*

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)

}

***Develop offline Dataset***

In [54]:

*# implement our function*

offline = readData()

**1.4 Signal Strength**

**1.4.1 Distribution of Signal Strength**

***Create posXY feature and Offline Signal Summary***

In [55]:

*#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

})

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

***Develop oneAPAngle***

In [56]:

oneAPAngle = subset(offlineSummary,

mac == subMacs[5] & 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)*

In [ ]:

In [57]:

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)

}

**DROP MAC Address and prime eliminate not\_eliminate logic**

In [77]:

*# here is where we drop a macid*

*# here is where we drop a macid*

*# subMacs[2] == 00:0f:a3:39:dd:cd*

*# subMacs[1] == 00:0f:a3:39:e1:c0*

eliminate = subMacs[2]

**if**(eliminate == '00:0f:a3:39:dd:cd'){

not\_eliminate <- '00:0f:a3:39:e1:c0'

} **else** **if**(eliminate == '00:0f:a3:39:e1:c0'){

not\_eliminate <- '00:0f:a3:39:dd:cd'

}

offlineSummary = subset(offlineSummary, mac != eliminate)

In [78]:

*# Look at the access points*

*# signal strength vs distance*

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

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")*

|  | **x** | **y** |
| --- | --- | --- |
| **00:0f:a3:39:e1:c0** | 7.5 | 6.3 |
| **00:14:bf:b1:97:8a** | 2.5 | -0.8 |
| **00:14:bf:3b:c7:c6** | 12.8 | -2.8 |
| **00:14:bf:b1:97:90** | 1.0 | 14.0 |
| **00:14:bf:b1:97:8d** | 33.5 | 9.3 |
| **00:14:bf:b1:97:81** | 33.5 | 2.8 |

**Tally Signal Strength**

In [79]:

*# tally signal strength*

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)

y = matrix(avgSS, nrow = 1, ncol = 6,

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

cbind(ans, y)

}))

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

60

0 45 90 135 180 225 270 315

0-0.05 0 0 0 593 0 0 0 0

0.15-9.42 0 0 606 0 0 0 0 0

0.31-11.09 0 0 0 0 0 573 0 0

0.47-8.2 590 0 0 0 0 0 0 0

0.78-10.94 586 0 0 0 0 0 0 0

0.93-11.69 0 0 0 0 583 0 0 0

**Create Data Frame and Functions to Aggregate/Select Data with Similar Angles**

In [80]:

*# 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, ]

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)

}

**Train Signal Strength**

In [81]:

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]])

166

**Nearest Neighbor Function**

In [82]:

*# here is our NN function.*

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 ])

}

**Predict X-Y Based on the Neasest k Neighbors (default 3)**

In [83]:

nearest\_neighbor = 3

**Predict and Map Errors**

In [84]:

*# predict X-Y based on the the neasest k neighbors (default 3)*

predXY = **function**(newSignals, newAngles, trainData,

numAngles = 1, k = nearest\_neighbor){

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)

}

*# nearest 3 neighbors*

estXYk3 = predXY(newSignals = onlineSummary[ , 6:11],

newAngles = onlineSummary[ , 4],

offlineSummary, numAngles = 3, k = nearest\_neighbor)

*# nearest neighbor*

estXYk1 = predXY(newSignals = onlineSummary[ , 6:11],

newAngles = onlineSummary[ , 4],

offlineSummary, numAngles = 3, k = 1)

In [85]:

*# predict and map errors*

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**,

main = "\* = Estimate, Solid Circle = Actual, Solid Square = AP")

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, col = "red")

}

*# offlineSummary$mac == "00:0f:a3:39:e1:c0"*

trainPoints = offlineSummary[ offlineSummary$angle == 0 &

offlineSummary$mac == not\_eliminate ,

c("posX", "posY")]

*# 3 NN*

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

trainPoints = trainPoints, AP = AP)

*# 1 NN*

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

trainPoints = trainPoints, AP = AP)

In [86]:

options(error = recover, warn = 1)

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)

}

1. 659.4003
2. 306.702522222222

Warning message in matrix(permuteLocs, ncol = v, nrow = floor(length(permuteLocs)/v)):

"data length [166] is not a sub-multiple or multiple of the number of rows [15]"

In [87]:

neighbors = 20

In [88]:

*# up to 20 neighbors, 11 folds*

*# this one can run for a while (5-10 mins)*

*# this cell and the next are the same, but the angles change slightly!!*

offline = offline[ offline$mac != eliminate, ]

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 = 1, k = 3)

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

calcError(estFold, actualFold)

K = neighbors

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 = 1, k = k)

err[k] = err[k] + calcError(estFold, actualFold)

}

}

plot(y = err, x = (1:K), type = "l", lwd= 2,

ylim = c(800, 2100),

xlab = "Number of Neighbors",

ylab = "Sum of Square Errors")

axis(side = 1,

at = round(seq(from=1, to=20, by=1), 0),

*# labels = v2,*

tck=-.05)

rmseMin = min(err)

kMin = which(err == rmseMin)[1]

segments(x0 = 0, x1 = kMin, y0 = rmseMin, col = gray(0.4),

lty = 2, lwd = 2)

segments(x0 = kMin, x1 = kMin, y0 = 1100, y1 = rmseMin,

col = grey(0.4), lty = 2, lwd = 2)

*#mtext(kMin, side = 1, line = 1, at = kMin, col = grey(0.4))*

text(x = kMin - 2, y = rmseMin + 40,

label = as.character(round(rmseMin)), col = grey(0.4))

estXYk5 = predXY(newSignals = onlineSummary[ , 6:11],

newAngles = onlineSummary[ , 4],

offlineSummary, numAngles = 1, k = 5)

calcError(estXYk5, actualXY)

144.777777777778

417.1843

In [89]:

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)

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

calcError(estFold, actualFold)

K = neighbors

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 = 1, k = k)

err[k] = err[k] + calcError(estFold, actualFold)

}

}

plot(y = err, x = (1:K), type = "l", lwd= 2,

ylim = c(1200, 2100),

xlab = "Number of Neighbors",

ylab = "Sum of Square Errors")

axis(side = 1,

at = round(seq(from=1, to=20, by=1), 0),

*# labels = v2,*

tck=-.05)

rmseMin = min(err)

kMin = which(err == rmseMin)[1]

segments(x0 = 0, x1 = kMin, y0 = rmseMin, col = gray(0.4),

lty = 2, lwd = 2)

segments(x0 = kMin, x1 = kMin, y0 = 1100, y1 = rmseMin,

col = grey(0.4), lty = 2, lwd = 2)

*#mtext(kMin, side = 1, line = 1, at = kMin, col = grey(0.4))*

text(x = kMin - 2, y = rmseMin + 40,

label = as.character(round(rmseMin)), col = grey(0.4))

estXYk5 = predXY(newSignals = onlineSummary[ , 6:11],

newAngles = onlineSummary[ , 4],

offlineSummary, numAngles = 3, k = 5)

calcError(estXYk5, actualXY)

print(calcError(estXYk5, actualXY))

87.3333333333333

275.5083

[1] 276

In [90]:

predXY2 = **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. Data Science in R, A Case Studies Approach to Computational Reasoning and Problem Solving by Deborah Nolan and Duncan Temple Lang [↑](#footnote-ref-1)