Unit 2 Case Study

MSDS Fall ‘19

7333 Quantify the World

Daniel Serna, Bruce Granger, and Brandon de la Houssaye

October 8, 2019

**Abstract**

In this paper we provide an in-depth analysis of a machine learning (ML) method that programmatically identifying the difference between emails that are unwanted (spam) from email that are wanted (ham). We will touch upon the history of unwanted emails, the financial impact of malicious spam emails, and who is at risk to spam.

**1 Introduction**

In today’s email reality, people subscribe to email newsletters, forums, alerts, etc. as a means of having information pushed to them automatically. What began as a helpful feature for users slowly turned into a marketing bonanza, as a new form of target marketing. The marketing firms then began to resell email address as they learned more about the users, thus creating value to retailers. Not to be left out of the bonanza, criminals (cyber-criminals) use email as means of attacking new targets through what is known as “phishing” expeditions. The basic aim of a phishing attack is to convince the email receiver to click on an active link that is embedded within the email, which then leads the victim to a fake website that closely resembles a legitimate company and where the user is asked to provide username and password and or other information. The phishing game is very easy for scammers (criminals) to replicate over and over and as a result is one of the easiest forms of cyber-attack for a cyber-criminal to carry out[[1]](#footnote-1). The latest form of attack often begins from email is called “ransomware”. The majority of ransomware is spread via gigantic spam email campaigns that involving hundreds of thousands of emails that are sent daily. In ransomware attack, the receiver is again prompted to click on embedded URLs within the email. Once a user clicks the link, malicious code is downloaded to the user’s machine that encrypts the users hard drive. Once encrypted, the cyber-criminal will lock out the user, then demand a ransom be paid, typically through bitcoin, before the cyber-criminal will unlock the user’s data. Often times malware will pose as ransomware and when victims pay the ransom to the cyber-criminal, the victims’ data is never “unlocked” by the cyber-criminal, this is known as “wiper” malware[[2]](#footnote-2). These attacks do not typically target an individual or organization, they are attacks of opportunity through the economies of scale[[3]](#footnote-3). A report from Cybersecurity Ventures estimated that damages from ransomware attacks cost as much as $8 billion globally in 2018[[4]](#footnote-4). One agent to combat this phenomenon is to make users aware of the dangers of spam email and educate users what to look for and what not to do. Another response to this is to phenomenon is to invest in technologies that stop spam email from ever entering the users “in box”, such as implementing machine learning to route spam to the spam folder and the good email to the inbox, which is the point of this exercise. Given what is at stake, the machine learning algorithm should error on the side of caution and false positives, which means we want to focus on “precision” as measurement to model effectiveness. Worst case, the user can check the emails in their spam folder for an email they are expected, in which case is “ham” email.

**2 Data**

The email message that will be at the core of this analysis is made available SpamAssassin (<http://spamassassin.apache.org>) and for repeatability and convenience these email examples have been bundled into the RSpamData package in the R programming language. The emails have been organized into five (5) subdirectories, three of which identify the good emails (ham) and two identify the unwanted emails (spam). The names of the spam folders are:

* spam
* spam\_2

The names of the ham folders are:

* easy\_ham
* easy\_ham\_2
* hard\_ham

There are a total of nine-thousand three-hundred and fifty-three (9,353) emails and the distribution of spam to ham is as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Folder Name | Email Type | Email Count | Type Total | Type % of Total |
| spam | Spam | 1,001 | 2,399 | 25% |
| spam\_2 | Spam | 1,398 |
| easy\_ham | Ham | 5,052 | 6,954 | 75% |
| easy\_ham\_2 | Ham | 1,401 |
| hard\_ham | Ham | 501 |

An email message has two parts, the email header and email body, which mirrors the concept of a standard letter that is sent through the United States Postage system. The email header is similar to the envelope that encapsulates the letter and contains metadata about where the email is going, additional recipients that will also receive the email, routing information. This information is contained in a key: value pair. The body of the email message is analogous to the letter being sent within the postal envelope. The body of the email message is separated from the email header by a single empty line. When an email attachment is added to the email message, the attachment is included in the body of the email message[[5]](#footnote-5). Within the metadata, the Content-Type key communicates if the message has an attachment or not. If the message does not have an attachment, with the value of “TEXT/PLAIN” is provided. If the message has an attachment, the value of “multipart”. In the case where there is an attachment, there is a boundary string. After the initial “boundary” is provided, subsequent boundary string is preceded by two hyphens, except the final boundary where it is preceded and followed by two hyphens.

**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:e1: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 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).

**5 Results**

For the first analysis, which focused on MAC addresses 00:0f:a3:39:dd:cd and MAC address 00:0f:a3:39:31:c0, the team plotted the signal strength versus distance to each of the seven access points. The team would conclude that using 00:0f:a3:39:e1:c0 would produce less accurate results based on the visual image of the plots. In particular, MAC address 00:0f:a3:39:e1:c0 shows a flatter line slope than MAC address 00:0f:a3:39:dd:cd. This implies that regardless distance, for MAC address 00:0f:a3:39:e1:c0, the signal strength would be less likely to fluctuate and would therefore be more difficult to point the communicating device in terms of plot location. In simple terms, if the volume remains the same whether the person yelling is 5 feet or 100 feet away, then volume would not be a good measure of distance.

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.

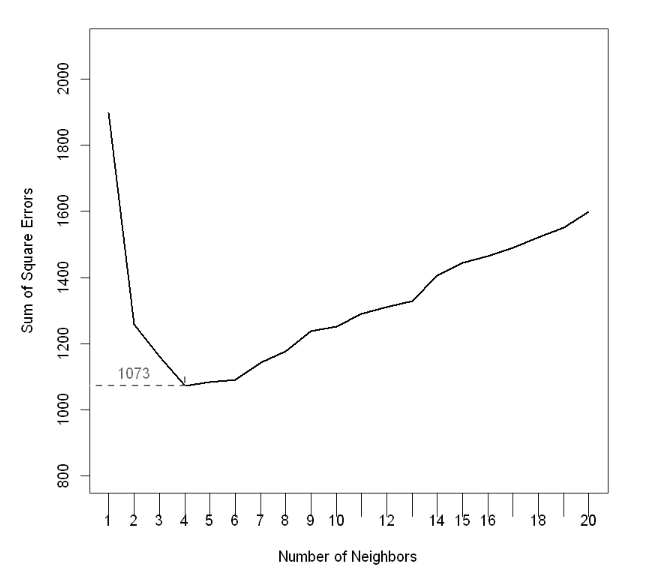


Figure - Original KNN Elbow Plot

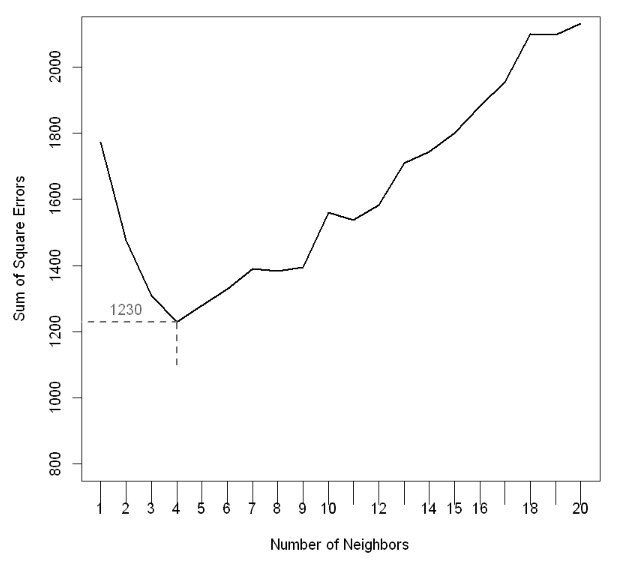


Figure - Weighted KNN Elbow Plot

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.

**6 Conclusion**

Our team was able to estimate the MAC address that was more likely to produce accurate results in order establish an indoor positioning system. The team reached its conclusion by using visual plot that compared the signal strength versus distance to access point for each of the seven access points (and each device). The team relied upon the visual image for purposes of this analysis but would potentially, in the future, run additional analysis by comparing the absolute accuracy of the system when employing one MAC address versus the other.

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**

**Required Libraries**

library(lattice)

library(fields)

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**

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**

*# 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**

*# 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**

*# 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***

*# implement our function*

offline **=** readData()

**1.4 Signal Strength**

**1.4.1 Distribution of Signal Strength**

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

*#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***

oneAPAngle **=** subset(offlineSummary,

mac **==** subMacs[5] **&** angle **==** 0)

​

​

*# library(fields)*

smoothSS **=** Tps(oneAPAngle[, c("posX","posY")], oneAPAngle**$**avgSignal)

​

vizSmooth **=** predictSurface(smoothSS)

​

In [ ]:

​

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**

*# 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 = 'None'

#eliminate = '00:0f:a3:39:dd:cd'

#eliminate = '00:0f:a3:39:e1:c0'

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

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

col = 6

matrix\_end = 11

findNN\_end = 9

offlineSummary = subset(offlineSummary, mac != eliminate)

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

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

col = 7

matrix\_end = 11

findNN\_end = 9

offlineSummary = subset(offlineSummary, mac != eliminate)

} else if(eliminate == 'None'){

not\_eliminate <- 'None'

col = 7

matrix\_end = 12

findNN\_end = 10

}

*# Look at the access points*

*# signal strength vs distance*

# CODE CHANGE

if(eliminate == 'None'){

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[], c("x", "y") ))

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

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") ))

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

AP = matrix( c( 7.5, 6.3, 2.5, -.8, 12.8, -2.8,

1, 14, 33.5, 9.3, 33.5, 2.8),

ncol = 2, byrow = TRUE,

dimnames = list(subMacs[ -1 ], c("x", "y") ))

}

AP

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")

​

#AP[ offlineSummary$mac, ]

unique(offlineSummary$mac)

**Tally Signal Strength**

*# 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:col, ]

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 = col, # CODE CHANGE

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

cbind(ans, y)

}))

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

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

*# 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**

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**

*# 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)

returnVal **=** trainSubset[closest, 1**:**3 ]

returnVal**$**weight **=** 1**/**closest

return(returnVal)

}

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

nearest\_neighbor **=** 3

**Predict and Map Errors**

*# 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)

*# 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)

​

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

1. 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]"

neighbors **=** 20

*# 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)

​

91

417.1843

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))

​

133.666666666667

275.5083

[1] 276

**Weighted nearest neighbor algorithm**

*# new predXY function*

​

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

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])))*

*# print("Original estXY")*

*# print(estXY)*

estXY **=** list(length **=** length(closeXY))

**for** (i **in** 1**:**length(closeXY)){

currentCloseXY **<-** closeXY[[i]]

*# currentMean <- sapply(currentCloseXY[ , 2:3],*

*# function(x) ((1/currentCloseXY[1,4]) / (sum(1/currentCloseXY[1:k,4]))) \* mean(x[1:k]))*

currentMean **<-** sapply(currentCloseXY[ , 2**:**3],

**function**(x) (weighted.mean(x[1**:**k], currentCloseXY[1**:**k,4])))

estXY[[i]] **<-** currentMean

}

*# print("New estXY")*

*# print(estXY)*

estXY **=** do.call("rbind", estXY)

*# print(estXY)*

return(estXY)

}

*# nearest 3 neighbors*

estXYk3 **=** predXY2(newSignals **=** onlineSummary[ , 6**:**11],

newAngles **=** onlineSummary[ , 4],

offlineSummary, numAngles **=** 3, k **=** 3)

​

*# nearest neighbor*

estXYk1 **=** predXY2(newSignals **=** onlineSummary[ , 6**:**11],

newAngles **=** onlineSummary[ , 4],

offlineSummary, numAngles **=** 3, k **=** 1)

*# 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)

​

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

1. 346.283420631847

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]"

neighbors **=** 20

*# 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 **=** predXY2(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 **=** predXY2(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 **=** predXY2(newSignals **=** onlineSummary[ , 6**:**11],

newAngles **=** onlineSummary[ , 4],

offlineSummary, numAngles **=** 1, k **=** 5)

​

calcError(estXYk5, actualXY)

​

​

209.010824139128

468.408342594425

onlineFold **=** subset(onlineCVSummary,

posXY **%in%** permuteLocs[ , 1])

​

offlineFold **=** subset(offlineSummary,

posXY **%in%** permuteLocs[ , **-**1])

​

estFold **=** predXY2(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 **=** predXY2(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 **=** predXY2(newSignals **=** onlineSummary[ , 6**:**11],

newAngles **=** onlineSummary[ , 4],

offlineSummary, numAngles **=** 3, k **=** 5)

​

calcError(estXYk5, actualXY)

print(calcError(estXYk5, actualXY))

​

123.573724883592

311.25235381449

[1] 311

1. https://www.forbes.com/sites/kateoflahertyuk/2018/08/17/how-to-survive-a-ransomware-attack-and-not-get-hit-again/#4807db156cd3 [↑](#footnote-ref-1)
2. Ibid [↑](#footnote-ref-2)
3. Ibid [↑](#footnote-ref-3)
4. https://www.newsweek.com/texas-ransomware-bitcoin-hackers-1454865 [↑](#footnote-ref-4)
5. Data Science in R, A Case Studies Approach to Computational Reasoning and Problem Solving Deborah Nolan, Duncan Temple Lang page 107 [↑](#footnote-ref-5)