Location Prediction with Indoor Positioning Systems

Megan Riley, Jaclyn Coate, Reagan Meagher

1. Introduction

There are around 5 billion unique smartphones in the world today. We often don't reflect on how we have the ability to know where we are, where our friends/family are, and even our lost devices at our fingertips. These are a few of the many reasons there is an unprecedented growth of wireless networks. With this growth comes the interest in being able to track individuals and devices both in the world and within standing structures. Global Position Systems (GPS) are very effective in tracking different things outdoors, however, are not reliable for tracking people/things within structures. Indoor positioning systems (IPS) have started to leverage wireless local area networks (LANs) and WiFi signals to address challenges such as: Where is this printer? Where is the ventilator in the hospital? What level is this rental car parked so I can go grab it for this customer?

In the following case study, we assessed if it is feasible to predict the location of specific items, contained within an indoor structure, based on the measurement of WiFi signals from different fixed access points (AP). We use a k-nearest neighbors technique with both a non-weighted and weighted methodology in our trained model for measuring the success of the IPS bouncing on WiFi networks. After deploying our models it became clear that the more AP reference points used the better a predictor model becomes. Even though we had two access point references in what appeared to be the exact same spot, including both of them increased our accuracy and reduced overall error. Our weighted k-NN model performed the best as a predictor and was compared to all other models using a RMSE. Even if they are not measured and known points and are just errant measurements from extra hand held devices. Leaving us to conclude that not only using WiFi signals but all signals in an indoor structure (including those mobile phones, prints, watches) can effectively increase the accuracy of using IPS both historically and in real-time.

Below we begin the analysis by loading packages of use and the data files labeled offline and online.

```
library(tidyverse)
library(stringr)
library(magrittr)
library(caret)
library(kknn)


#Loading data locally
offline_txt =
  readLines("/Users/zmartygirl/Documents/MSDS_SMU/QTW_7333/cs2_data/offline.final.trace.txt")
online_txt =
  readLines("/Users/zmartygirl/Documents/MSDS_SMU/QTW_7333/cs2_data/online.final.trace.txt")
```

2. Data

In order to test if WiFi signals are a good way to power IPS', we first need to have access to a set of reference data. This data is available via the CRAWDAD website. The reference data will be termed our "offline" data. It contains the varying signal strengths observed using a hand-held device within an office building. The signal strength tests were performed a single meter apart within a single floor office building at the University of Mannheim. The locations reference in the offline data allow us a base-line set of signal strength for the model builds. We will use this data set to construct our model, then leveraging it to predict the locations of other devices when it's position is unknown within the building.

To accompany our offline data, we will also use a set of data that is recordings, deemed the "online" data. This data set is in order to test the model we build with our offline data set. Our testing data contains 60 locations and orientations that are randomly chosen with 110 signals measured from those locations.

To load, clean and organize our data, we use a few helper functions defined below. processLine is a function from the given code that breaks the lines of data from the files into rows. RoundOrientation rounds orientation angles to the nearest fifteen degrees. Finally load_data uses both functions to take the files from loaded form into a full data frame.

```
#Functions from Provided Code
processLine = function(x)
{
tokens = strsplit(x, "[;=,]")[[1]]
if (length(tokens) == 10)
    return(NULL)
tmp = matrix(tokens[ - (1:10) ], , 4, byrow = TRUE)
#build a 4 column matrix mac,signal,scan,type
cbind(matrix(tokens[c(2, 4, 6:8, 10)], nrow(tmp), 6,
#builds a 6 column matrix t,scanMac,x,y,z,angle
byrow = TRUE), tmp) #binds them together
}

#Round Orientation rounds angles to nearest 45 degrees
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]
}

#Loading Data from file to dataframe using processLine and Round Orientation Functions
load_data  = function(txt){
  lines = txt[ substr(txt, 1, 1) != "#" ]
  #Run process line function
  df = lapply(lines,processLine)
  df = as.data.frame(do.call("rbind", df),stringsAsFactors = FALSE)
  #Give column names
  names(df) =c("time", "scanMac", "posX", "posY", "posZ",
"orientation", "mac", "signal", "channel", "type")
  #Create an XY postiion column
  df$posXY = paste(df$posX, df$posY, sep = "-")
  #Create angle column
  df$orientation = as.integer(df$orientation)
  df$angle = roundOrientation(df$orientation)
  return(df)
}


#using helper functions, load online and offline data into dataframes
offline = load_data(offline_txt)

online = load_data(online_txt)
```

3. Expoloratory Data Analysis

The raw data file is a combination of the measurement logs with the MAC addresses of each scanning device and the corresponding: timestamp of measurement positions, orientations, and the MAC address with AP signal strength on a single line [Table 2.1]. The unique identifier for the hardware of a device is the media access control (MAC). The MAC variable allows for a network card of an equipment to be identified on a network.

Variable Units T timestamp in milliseconds since midnight, January 1, 1970 UTC *Id* MAC address of the scanning device *Pos* the physical coordinate of the scanning device *Degree* orientation of the user carrying the scanning device 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 dB (Decibel-milliwatts), the channel frequency and its mode (access point = 3, device in adhoc mode = 1)

We leveraged the functions provided in Nolan and Lang (2015): processLine, roundOrientation, and readData. Thus allowing us to easily parse and read the data into a usable data frame format with R. The output of these functions present the data to have the same initial variables of the hand-held device: MAC address, location, orientation, and time with the additional 4 variables: MAC address of device the signal was received, the channel, the signal, and the type of device. Mostly importantly this data frame structure allows us to perform group-by operations on the MAC addresses.

With our data frame in a readable format we were able to determine all the variable units contained within the data and label them accordingly: time, scanMAC, posX, posY, posZ, orientation, mac, signal, channel, and type. This process allowed us to complete large amounts of data cleaning. For instance, dropping irrelevant variables we say kept constant between MAC addresses: posZ and scanMAC. Additionally, only keeping those mac addresses that were relevant to our APs; the ones labeled with 3 and not the adhocs labeled with 1.

The last few levels of our EDA contained evaluations on the data to assure the data fit the expected trends outlined in the data documentation. This includes rounding our orientation measurements to the identified 8 using the roundOrientation function. Exploring the MAC addresses to identify those that should be used in our model build, quickly seeing there are 2 that could potentially identify with a single AP, thus creating our need for multiple iterations of models. Lastly, assessing signal strength and the relationships between signal and distance to determine and apply any smoothing techniques needed to allow us to model the behaviors we observe: such as, lack of consistency between standard deviations of weak and strong signal strengths, which is required as an assumption for the data. We leveraged topography maps as described in Nolan and Lang (2015) in order to evaluate the relationships between distance and signal strength.

```
#Turning offline dataframe into pivot table
subMacs = names(sort(table(offline$mac), decreasing = TRUE))[1:7]
subMacs
```

```
## [1] "00:0f:a3:39:e1:c0" "00:0f:a3:39:dd:cd" "00:14:bf:b1:97:8a"
## [4] "00:14:bf:3b:c7:c6" "00:14:bf:b1:97:90" "00:14:bf:b1:97:8d"
## [7] "00:14:bf:b1:97:81"
```

```
offline1 = offline[ offline$mac %in% subMacs, ]

offline1$signal %<>% as.integer
offline_pivot<-select(offline1, -c(channel,scanMac)) %>%
  pivot_wider(names_from = mac,values_from = signal, values_fn = list(signal=mean))


offline_pivot$nas<-rowSums(is.na(offline_pivot))
offline_pivot = offline_pivot[offline_pivot$nas==0,]
```

```
#Turning online dataframe into pivot table
subMacs = names(sort(table(online$mac), decreasing = TRUE))[1:7]
subMacs
```

```
## [1] "00:0f:a3:39:dd:cd" "00:0f:a3:39:e1:c0" "00:14:bf:b1:97:8a"
## [4] "00:14:bf:b1:97:90" "00:14:bf:3b:c7:c6" "00:14:bf:b1:97:8d"
## [7] "00:14:bf:b1:97:81"
```

```
online1 = online[ online$mac %in% subMacs, ]

online1$signal %<>% as.integer
online_pivot<-select(online1, -c(channel,scanMac)) %>%
  pivot_wider(names_from = mac,values_from = signal, values_fn = list(signal=mean))

online_pivot$nas<-rowSums(is.na(online_pivot))
online_pivot = online_pivot[online_pivot$nas==0,]
```

Below we created several subsets to manage different k-NN approaches. An original subset using the six MAC addresses explored by the Nolan and Lang(2015). An alternate six addresses that drops the one in use and uses the one originally dropped. And finally a full subset with the seven most used MAC addresses including the two found to be in similar locations.

```
offline_subset <- select(offline_pivot, c("posX", "posY", 'posXY', "angle",
                                          "00:14:bf:b1:97:8a", "00:14:bf:b1:97:90",
                                          "00:0f:a3:39:e1:c0",  "00:14:bf:b1:97:8d",
                                          "00:14:bf:b1:97:81","00:14:bf:3b:c7:c6",
                                          "00:0f:a3:39:dd:cd"))
#Various Offline subsets to build KNN Models

#With all available mac addresses
seven_mac_offline = offline_subset


#With the original mac address removed as in the book
original_six_mac_offline = offline_subset %>% select(-"00:0f:a3:39:dd:cd")

#With the other mac address removed
alternate_six_mac_offline = offline_subset %>% select(-"00:0f:a3:39:e1:c0")
```

4. Model Type: k-NN

There are many statistical methods to estimate a location based on test data. We leveraged the k-nearest neighbors (k-NN), it is an intuitive approach to location based prediction. The k-NN prediction technique is as follows: researchers obtain a set of training data where a signal strength is measured across several APs from known positions within a structure, when any new signal observation is obtained and the position is unknown, the researchers find the observation in their training data closest to the new signal observation. Then a researcher can predict the position of the new observation as the position of the closest training observation. Since we are working within standing structures and signals can pass through solid barriers, we use the euclidean distance to determine the closest signal of the new observation with the location unknown. Additionally, we tried a weighted k-NN model. The idea of a k-NN model is to give more weights to the points which are nearby and less weight to the points which are farther away. Effectively smoothing the data and assuring those close points are more heavily contributing to the prediction model.

When using the k-NN method of prediction there is always the need to determine what the best k for prediction will be. The k is the number of neighbors to include in comparing the known location observation to the unknown location observation.

Since we are trying to find the best predictors for our model building and we have an 'extra' MAC address. We have divided our data set into three different groups: Original 6 MAC, Alternate 6 MAC, and 7 MAC

addresses to see if the extra MAC address can contribute to better predicting unknown signals within the building.

First below we build a data frame to hold the results of each attempted model.

```
#Set up a final results dataframe to work with
Final_results = data.frame("Model_Type" = c("Original Six Mac Addresses",
                                            "Alternate Six Mac Addresses",
                                            "All Seven Mac Addresses",
                                            "Weighted KNN Model"),
                           "K" = rep(0,4), "Accuracy"= rep(0,4), "Time" = rep(0,4),
                           "RMSE" = rep(0,4))
```

5. Model Validation

Selection of the k for our k-NN model can only be completed by using a cross validation method that ensures there is no overfitting of our model and leaving data to test the model with our trained data set. Instead of manually holding out data to train and test our k number options we leveraged the k-fold technique which does this for us automatically. The k-fold tools in R can use a selected ratio; in our case 9:1 (10 fold), to test the k selection on 9 folds of the data leaving 1 fold to test the determination on. Rotating through the entire data set for the training and test ratios. Effectively, training our model on all data available but at the same time cross validating our k selection for our k-NN model.

With this cross validation method we built 3 models. The first model we built leveraged the data set with the Original 6 MAC addresses contained within it. The second model we built contained the Alternate 6 MAC addresses. Thirdly, we built a model that contained all 7 MAC addressee for prediction. Against these f-fold cross validated models we used a traditional 90:10 ratio split of data and built a weighted k-NN model to see how it competes against our other 3.

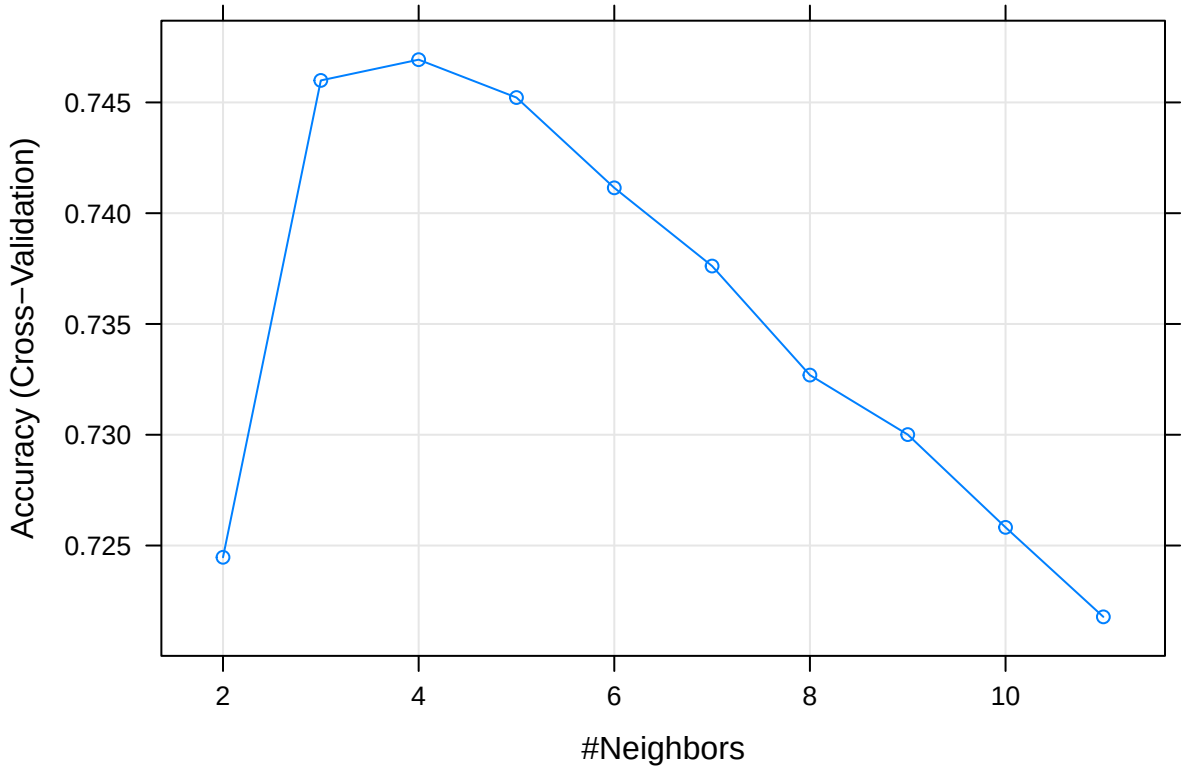
5.1 Original Six MAC Addresses

Below is the first analysis using the six original MAC addresses. We use a 10 fold cross validation to train the method and find accuracy from within the training data alone.

```
train.control <- trainControl(method = 'cv', number = 10)
knn.grid <- expand.grid(k = seq(2,11))

knnFit <- train(posXY ~ ., data = original_six_mac_offline %>%
                  select(-c("posX","posY")), method = "knn", trControl = train.control,
                tuneGrid = knn.grid)


best_k = 3
Final_results[1,2] = best_k
Final_results[1,3] = knnFit$results$Accuracy[best_k] #need to report this accurayc
#Full_Accuracy = knnFit$results #in appendix for reference
Final_results[1,4] = knnFit$times$everything[3]
plot(knnFit)
```



With this fitted kNN model we build predictions on the entire online test set, the performance in this model is laid out in the chunk below.

```
#original six addresses predict
knnPredict <- predict(knnFit,newdata = online_pivot %>%
                        select(-c("posX","posY","posXY","00:0f:a3:39:dd:cd")))
temp_res = data.frame(predictions = knnPredict,
                      TrueX = online_pivot$posX, TrueY = online_pivot$posY)

#using predictions, split into X and Y
split_x = list()
split_y = list()
for(item in seq(1:dim(temp_res)[[1]])){
  tmp = unlist(strsplit(as.character(temp_res[item,1]), "[-]"))
  split_x[item] = tmp[1]
  split_y[item] = tmp[2]
}

orig_mac_residuals = data.frame(predictions = knnPredict,
                                PredX = as.numeric(unlist(split_x)),
                                PredY = as.numeric(unlist(split_y)),
                                TrueX = as.numeric(online_pivot$posX),
                                TrueY = as.numeric(online_pivot$posY))


diff1 =  as.numeric(orig_mac_residuals$TrueX) - as.numeric(orig_mac_residuals$PredX)
diff2 = as.numeric(orig_mac_residuals$TrueY) - as.numeric(orig_mac_residuals$PredY)
diffxy = abs(diff1) + abs(diff2)
orig_mac_residuals$diffXY = diffxy
```

```
RMSE = sqrt(sum(orig_mac_residuals$diffXY^2) / dim(orig_mac_residuals)[1])

Final_results[1,5] = RMSE
print(Final_results[1,])
```

```
##                    Model_Type K  Accuracy    Time    RMSE
## 1 Original Six Mac Addresses 3 0.7469289 510.356 4.89083
```

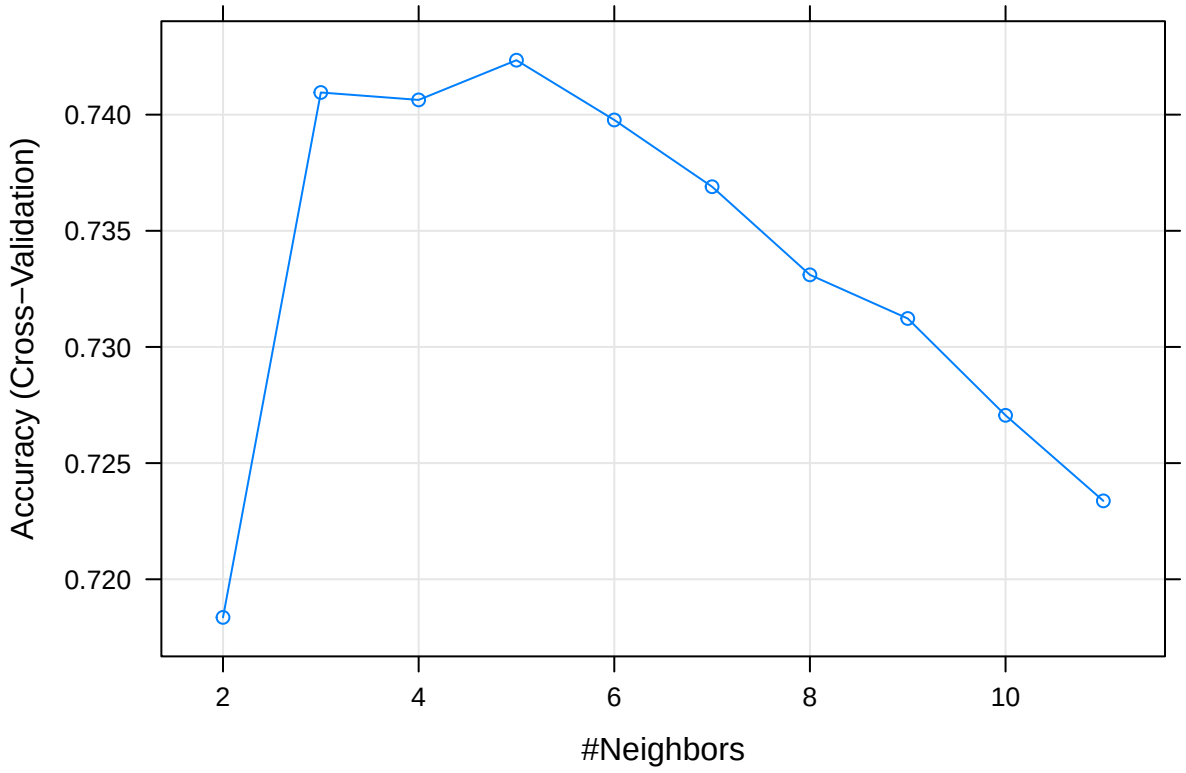
5.2 Alternate Six MAC Addresses

Using the same method as before we build a new kNN model using the alternate MAC address data set, with the originally chosen MAC address dropped and the other used in it's place.

```
train.control <- trainControl(method = 'cv', number = 10)
knn.grid <- expand.grid(k = seq(2,11))

knnFit1 <- train(posXY ~ ., data = alternate_six_mac_offline %>%
                   select(-c("posX","posY")), method = "knn", trControl = train.control,
                 tuneGrid = knn.grid)


best_k = 5
Final_results[2,2] = best_k
Final_results[2,3] = knnFit1$results$Accuracy[best_k] #need to report this accurayc
#Full_Accuracy = knnFit1$results #in appendix for reference
Final_results[2,4] = knnFit1$times$everything[3]
plot(knnFit1)
```



The results for this analysis are shown below.

```
#Alternate six addresses predict
knnPredict <- predict(knnFit1,newdata = online_pivot %>%
                        select(-c("posX","posY","posXY","00:0f:a3:39:e1:c0")))
temp_res = data.frame(predictions = knnPredict, TrueX = online_pivot$posX,
                      TrueY = online_pivot$posY)

#using predictions, split into X and Y
split_x = list()
split_y = list()
for(item in seq(1:dim(temp_res)[[1]])){
  tmp = unlist(strsplit(as.character(temp_res[item,1]), "[-]"))
  split_x[item] = tmp[1]
  split_y[item] = tmp[2]
}

orig_mac_residuals = data.frame(predictions = knnPredict,
                                PredX = as.numeric(unlist(split_x)),
                                PredY = as.numeric(unlist(split_y)),
                                TrueX = as.numeric(online_pivot$posX),
                                TrueY = as.numeric(online_pivot$posY))


diff1 =  as.numeric(orig_mac_residuals$TrueX) - as.numeric(orig_mac_residuals$PredX)
diff2 = as.numeric(orig_mac_residuals$TrueY) - as.numeric(orig_mac_residuals$PredY)
diffxy = abs(diff1) + abs(diff2)
orig_mac_residuals$diffXY = diffxy


RMSE = sqrt(sum(orig_mac_residuals$diffXY^2) / dim(orig_mac_residuals)[1])

Final_results[2,5] = RMSE
print(Final_results[2,])
```

```
##                     Model_Type K  Accuracy   Time     RMSE
## 2 Alternate Six Mac Addresses 5 0.7397714 637.84 4.435598
```

5.3 Seven MAC Addresses

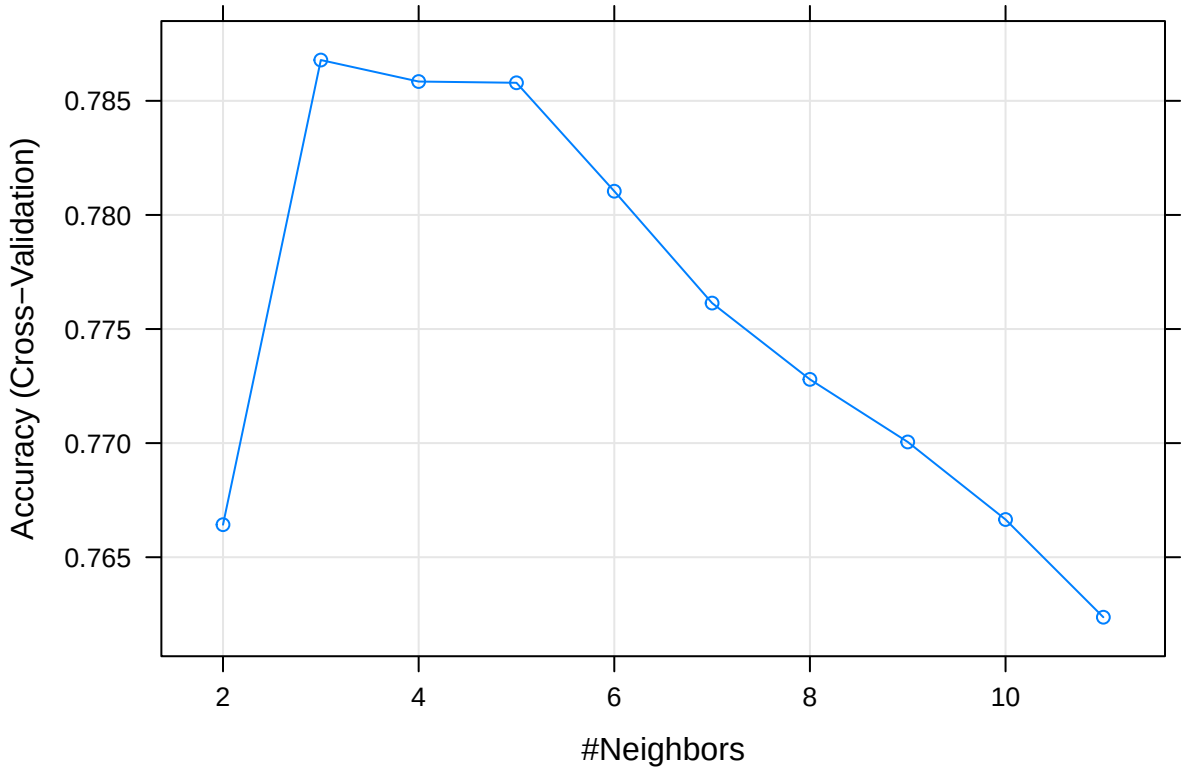
Finally we use all the seven major MAC addresses to build a kNN model to detect location.

```
train.control <- trainControl(method = 'cv', number = 10)
knn.grid <- expand.grid(k = seq(2,11))

knnFit2 <- train(posXY ~ ., data = seven_mac_offline %>%
                   select(-c("posX","posY")), method = "knn", trControl = train.control,
                 tuneGrid = knn.grid)


best_k = 3
Final_results[3,2] = best_k
Final_results[3,3] = knnFit2$results$Accuracy[best_k] #need to report this accurayc
#Full_Accuracy = knnFit$results #in appendix for reference
Final_results[3,4] = knnFit2$times$everything[3]
```

```
plot(knnFit2)
```



With the seven MAC address kNN model built we can use it to predict the test data set. The results are laid out below the code chunk. So far, this is the most accurate model we have.

```
#SEven addresses predict
knnPredict <- predict(knnFit2,newdata = online_pivot %>%
                        select(-c("posX","posY","posXY")))
temp_res = data.frame(predictions = knnPredict,
                      TrueX = online_pivot$posX, TrueY = online_pivot$posY)

#using predictions, split into X and Y
split_x = list()
split_y = list()
for(item in seq(1:dim(temp_res)[[1]])){
  tmp = unlist(strsplit(as.character(temp_res[item,1]), "[-]"))
  split_x[item] = tmp[1]
  split_y[item] = tmp[2]
}

orig_mac_residuals = data.frame(predictions = knnPredict,
                                PredX = as.numeric(unlist(split_x)),
                                PredY = as.numeric(unlist(split_y)),
                                TrueX = as.numeric(online_pivot$posX),
                                TrueY = as.numeric(online_pivot$posY))


diff1 =  as.numeric(orig_mac_residuals$TrueX) - as.numeric(orig_mac_residuals$PredX)
diff2 = as.numeric(orig_mac_residuals$TrueY) - as.numeric(orig_mac_residuals$PredY)
diffxy = abs(diff1) + abs(diff2)
```

```
orig_mac_residuals$diffXY = diffxy


RMSE = sqrt(sum(orig_mac_residuals$diffXY^2) / dim(orig_mac_residuals)[1])

Final_results[3,5] = RMSE
print(Final_results[3,])
```

```
##                 Model_Type K  Accuracy    Time     RMSE
## 3 All Seven Mac Addresses 3 0.7858465 586.287 4.354741
```

5.4 Weighted Seven MAC Addresses

To build the weighted kNN model we used the best performing model, the kNN using all seven MAC addresses. This weighted kNN model was built with a weight measure using the inverted value of distance of the nearest neighbors, and an ideal k of 3.

```
training = seven_mac_offline
training$posXY = as.factor(training$posXY)

leavout_idx = sample(seq(1,dim(training)[1]), round(.8 *dim(training)[1]))
leavout_train = training[leavout_idx,]
leavout_test = training[-leavout_idx,]

#Documentation of kknn specificially defines inv as:  if (kernel=="inv") W <- 1/W
# 1 / distance or 1 / weights is what was requested by prof.

train_weightedKNN = kknn(posXY ~ ., train = leavout_train %>% select(-c("posX","posY")),
                         test = leavout_test %>% select(-c("posX","posY")),
                         distance = 1, k = 3, kernel = "inv")

cM = confusionMatrix(train_weightedKNN$fitted.values, leavout_test$posXY)

Final_results[4,3] = cM$overall[1]
```

With the Weighted kNN model built we can retrain it on the entire training data set to predict the test data set. The results are laid out below the code chunk.

```
testing = online_pivot %>%
  select(-c("posX","posY","posXY","posZ","orientation","type","time"))


#Distance is minkowski =1
#Using the parameter of K = 3 from earlier work
weightedKNN = kknn(posXY ~ ., train = training %>% select(-c("posX","posY")),
                   test = testing, distance = 1, k = 3, kernel = "inv")

#K VAlue
Final_results[4,2] = 3
```

```
#Final KNN predict
knnPredict <- weightedKNN$fitted.values
temp_res = data.frame(predictions = knnPredict,
                      TrueX = online_pivot$posX, TrueY = online_pivot$posY)
```

```
#using predictions, split into X and Y
split_x = list()
split_y = list()
for(item in seq(1:dim(temp_res)[[1]])){
  tmp = unlist(strsplit(as.character(temp_res[item,1]), "[-]"))
  split_x[item] = tmp[1]
  split_y[item] = tmp[2]
}

orig_mac_residuals = data.frame(predictions = knnPredict,
                                PredX = as.numeric(unlist(split_x)),
                                PredY = as.numeric(unlist(split_y)),
                                TrueX = as.numeric(online_pivot$posX),
                                TrueY = as.numeric(online_pivot$posY))


diff1 =  as.numeric(orig_mac_residuals$TrueX) - as.numeric(orig_mac_residuals$PredX)
diff2 = as.numeric(orig_mac_residuals$TrueY) - as.numeric(orig_mac_residuals$PredY)
diffxy = abs(diff1) + abs(diff2)
orig_mac_residuals$diffXY = diffxy


RMSE = sqrt(sum(orig_mac_residuals$diffXY^2) / dim(orig_mac_residuals)[1])

Final_results[4,5] = RMSE
Final_results[4,4] = NA
print(Final_results[4,])
```

```
##           Model_Type K  Accuracy Time     RMSE
## 4 Weighted KNN Model 3 0.8042825   NA 4.231039
```

6. Results

In Table 6.1 we have summarized the model performance across all datasets allowing us to see which MAC ids were the most effective in predicting unknown locations. Our weighted k-NN out performed all other modeling methods in both accuracy and RMSE.

```
print( Final_results)
```

```
##                    Model_Type K  Accuracy    Time     RMSE
## 1  Original Six Mac Addresses 3 0.7469289 510.356 4.890830
## 2 Alternate Six Mac Addresses 5 0.7397714 637.840 4.435598
## 3     All Seven Mac Addresses 3 0.7858465 586.287 4.354741
## 4          Weighted KNN Model 3 0.8042825      NA 4.231039
```

7. Conclusions

The research depicts that the option to use all 8 MAC addresses available yields a better prediction for our unknown signal locations. This logically makes sense when thought out. As known AP can be leveraged to locate the devices of unknown signals, leveraging those additional signals that are in definite close proximity to the APs seems to allow what appears to be a boost in signal and therefore higher accuracy of location prediction for the unknown device. This translated into practice looks like companies leveraging, not only known access points, but those errant signals that are also present as devices used by individuals in the

buildings. Effectively allowing them to more easily locate the things they are looking for within stranding structures.

This being said, it does not appear that the overall accuracy of using WiFi signals with known access points for RTLS is the best way to determine a signal's unknown location. We can note that the errors depicted in Table 3.1 are 4 times as large as the single meter distance of measurements taken. Additionally, in Figure 6.1 we see that the predicted signal locations versus the known locations are highly variable. However, we do see that in hallways there is more accuracy since there are less barriers for the signal to pass through in order to receive a signal.

Lastly, the k-NN algorithm can be costly in processing time to leverage. This is due to the fact that the k-NN model computes the distance across the entire data set each time the model is executed. The calculation time can become costly for researchers and therefore could be too great of an overhead cost. Nguyen (2017) argues that the "location fingerprinting" with confidence measurements and a weighted k-NN method for indoor positioning systems produces a 20% higher accuracy output for predicting unknown signals making it a method that would be a good alternative to k-NN. WIth such a higher accuracy it is possible the computationally intensive k-NN model is worth the extra time and overhead.

8. References

Nolan, D. and Lang, D.T. (2015). Data Science in R A Case Studies Approach to Computational Reasoning and Problem Solving. CRC Press.

Nguyen, Khuong. "A Performance Guaranteed Indoor Positioning System Using Conformal Prediction and the WiFi Signal Strength." Journal of information and telecommunication (Print) 1.1 (2017): 41–65. Web.