Predicting Real-Time Indoor Position Using WIFI

Laura Lazarescou, John Rodgers, Maysam Mansor and Mel Schwan

January 18, 2021

1 Introduction

Predicting the real-time location of an object using a real-time location system (RTLS) is extremely valuable in many circumstances and commercial environments. Indoor positioning systems (IPS) track people and things inside stores, hospitals, warehouses, and factories using readily available technology. This enables data predictive modeling which can increase the safety, efficiency and profits of any organization. For this project we used the RTLS test data set provide with the Nolan, Deborah; Lang, Duncan Temple. Data Science in R: A Case Studies Approach to Computational Reasoning and Problem Solving (Chapman & Hall/CRC The R Series) (p. 3). CRC Press. The importance of accurately tracking human assets has become a valuable capability during the COVID-19 pandemic. Contact tracing would be greatly simplified if real-time tracking data could be retrieved when an employee tests positive.

This case study will include the normal data analysis steps of asking the right question, data wrangling, exploratory data analysis, model design and optimization, results and conclusion. The right question is "Can we accurately predict the location of a person or object through the WIFI access point (AP) signal strengths in an indoor environment?"

2 Methods

2.1 Data

2.1.1 Data Collection

Two relevant data sets for developing an IPS are available on the CRAWDAD site (A Community Resource for Archiving Wireless Data At Dartmouth) [2]. One is a reference set, termed "offline," that contains receive signal strengths (RSS) measured using a hand-held scanner on a grid of 166 points spaced 1 meter apart in the hallways of one floor which measures about 15 meters by 36 meters. The points of measurement are referred to as posX and posY. Additionally, the handheld scanning device was rotated to 8 angles (45°increments from 0) at each measurement location. These reference locations give us a calibration set of signal strengths for the building, and we use them to build our model to predict the locations of the hand-held device when its position is unknown. The data was collected in an indoor office environment on a single floor with 7 fixed WiFi APs. The WiFi APs are identifiable by their media-access-control (MAC) addresses.

Two of these APs with MAC address of 00:0f:a3:39:e1:c0 and 00:0f:a3:39:dd:cd are positioned in close proximity. In the Nolan and Lang study, they chose to disregard the measurements associated with MAC address 00:0f:a3:39:dd:cd.

2.1.2 Data Cleaning

Much of the Nolan and Lang study focused on the tokenization and grooming of the data. We extracted the measurement log data by reading each line consisting of the Scanner MAC address, measurement position, measurement orientation , AP MAC address, AP receive signal strength, and time stamp and tokenizing each unit of information. The data was initially organized in a long format data frame with time, position X, Y, and Z, orientation, AP MAC address, signal strength, and channel as the columns. We determined by looking at the column data that several columns were not necessary. Position of Z and the scan MAC address were fixed so these were eliminated. The channel column was also eliminated because only one channel was used for data collection. All MAC addresses coded as type 1 were eliminated since only type 3 were fixed APs.

In our study, we repeated the Nolan and Lang approach, then removed some of the data filters to allow for analysis of data from all seven access points.

2.2 Modeling Methods and Results

Since our objective for this case study is to answer the question: "Can we accurately predict the location of a person or object through the WIFI access point (AP) signal strengths in an indoor environment?" we will evaluate four scenarios to see which has the lowest amount of error, or specifically, which has the lowest Root Mean Squared Error (RMSE)?

2.2.1 Internal RMSE: K-Fold Cross-Validation

If we did not have the "online.txt" dataset as a test dataset, we could choose to break our train dataset into train and test. However, the k-fold cross-validation technique essentialy splits our "offline" dataset into "V" non-overlapping subsets of equal size. We assess the predictive ability of the model using the subset that was left out. The model fitting and assessment is aggregated across the folds. In our nearest neighbor scenario, we use all 8 orientations and 6 MAC addresses with each location. We used v = 11; then each fold has floor(166/v), or 15, locations.

In addition to determining an internal RMSE, cross-validation allows us to iterate on a sequence of k-nearest neighbors and decide which value of nearest neighbors might yield a model with the lowest RMSE. In Figure 1 below, it is clear that five (k=5) nearest neighbors is the lowest point in the graph, which represents the lowest value for RMSE.

The internal value of RMSE, using 11-fold cross-validation is 1105.

We plotted (fig.1) these values against the value of K tp find the optimum K value (5) for the lowest RMSE (1238)

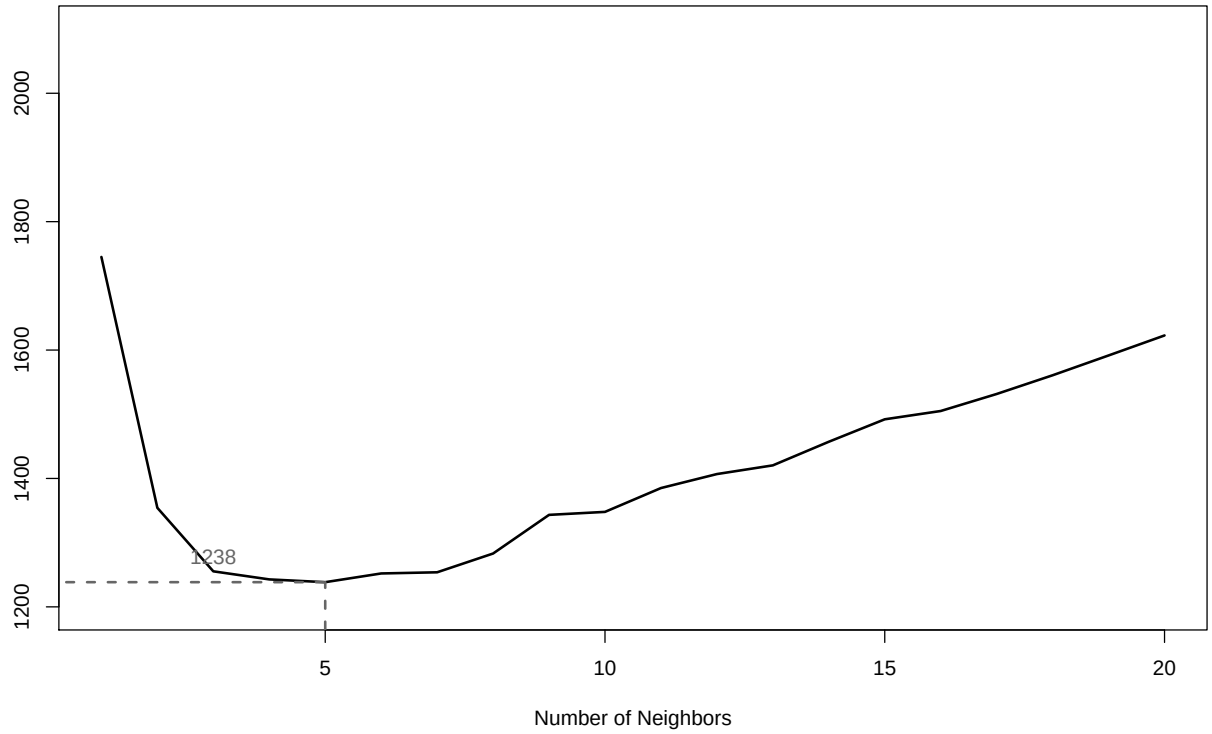

Figure 1: K-Fold Optimization of K value

2.2.2 Scenario 1: Evaluate Six Access Points Excluding 00:0f:a3:39:dd:cd

We provide this reference as a baseline for future scenarios. This scenario is the same scenario that was performed by Nolan and Lang.

In Scenario 1, the RMSE for the baseline six-AP configuration is 276.

2.2.3 Scenario 2: Evaluate Six Access Points Excluding 00:0f:a3:39:e1:c0

This scenario allows us to compare the performance of the access point that was ignored by Nolan and Lang.

The RMSE for scenario 2 is 250.

2.2.4 Scenario 3: Evaluate Seven Access Points

This scenario included consideration of both access points that appeared to be redundant: 00:0f:a3:39:dd:cd and 00:0f:a3:39:e1:c0.

Five additional access points were used in all studies: “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”

In Scenario 3, whe we consider all seven APs, the RMSE is 228.

2.2.5 Scenario 4: Evaluate Using Weighted Distance

In this scenario we calculate the Euclidean distance between the "offline" or train dataset and each line of our "online" or test dataset. We used five (5) nearest neighbors to calculate the weighted distance because our k-fold internal analysis indicated that k=5 was the optimal number of nearest neighbors.

In Scenario 4, our weighted average approach yielded an RMSE of 1770 Given that this scenario is a different context, we believe our RMSE values may not be comparable to those of Scenarios 1, 2 and 3.

2.3 Conclusion

In the original Nolan and Lange study, one of seven access points was rejected without studying its contribution to the accuracy of the overall system. As we see in comparing Scenarios 1, 2 and 3, Nolan and Lange chose the least accurate configuration.

Scenario 2 demonstrated that our test data was better modeled by the inclusion of access point — (MAC address)

Scenario 3 surprised us by demonstrating that the RMSE could be even lower with seven access points. This makes the case for more access points, even when they appear to be located very close to each other.

Our weighted distance calculation was inconclusive. However, there is a strong case for continuing this analysis in different contexts. In our original exploratory data analysis (EDA) we noticed that RSS values varied depending on the angle of measurement. If certain angles are deemed to be more accurate, they might influence the positioning of assets in a cart, or badges worn by people. All of these measurements are also subject to confounding factors such as the presence of other wireless devices in proximity. Concrete walls, ceiling height, line of sight are all factors that could affect the RSS values.

There are also disadvantages to having too many access points. The primary disadvantage is cost. Each unit has a cost to purchase, configure and maintain. They also create interference if they are too closely positioned, or if their power is too high. These effects may have been present in this study but we do not have the data to account for interference or multi-path.

As we revisit our original question: "Can we accurately predict the location of a person or object with RTLS technology?" the answer is "yes", and depending on the required degree of accuracy and the amount of money an organization can invest, they may benefit from installing more access points.

3 Appendix: All code for this report

```
knitr::opts_chunk$set(echo = FALSE)

#  load-data-functions

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)
}
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]
}

fname = 'Data/offline.final.trace.txt'
readData =
  function(filename = fname,
           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)
  }
reshapeSS = function(data, varSignal = "signal",
                     keepVars = c("posXY", "posX","posY"),columns) {
  byLocation =
    with(data, by(data, list(posXY),
                  function(x) {
                    ans = x[1, keepVars]
                    avgSS = tapply(x[ , varSignal ], x$mac, mean)
                    y = matrix(avgSS, nrow = 1, ncol = columns, ##6,
                               dimnames = list(ans$posXY,
                                               names(avgSS)))
                    cbind(ans, y)
                  }))

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

selectTrain = function(angleNewObs, signals = NULL, m = 1, columns){
  # 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", columns = columns)
}

# Find nearest neighbor
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 XY
predXY = function(newSignals, newAngles, trainData,
                  numAngles = 1, k = 3, columns){

  closeXY = list(length = nrow(newSignals))

  for (i in 1:nrow(newSignals)) {
    trainSS = selectTrain(newAngles[i], trainData, m = numAngles, columns=columns)
    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)
}

# calculate RMSE
calcError =
function(estXY, actualXY)
   sum( rowSums( (estXY - actualXY)^2) )

# take offline data set and optional excluded Mac and calculate return RMSE

errorFunc = function(offline, numberMac = 7) {
  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)

  macs = unique(offlineSummary$mac)

  online = readData("Data/online.final.trace.txt", subMacs = macs)
  online$posXY = paste(online$posX, online$posY, sep = "-")

  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 = numberMac,
                          dimnames = list(ans$posXY, names(avgSS)))
                    cbind(ans, y)
                  }))

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

  actualXY = onlineSummary[ , c("posX", "posY")]

  estXYk5 = predXY(newSignals = onlineSummary[ , 6:11],
                   newAngles = onlineSummary[ , 4],
                   offlineSummary, numAngles = 3, k = 5, columns = numberMac)

  returnVal = calcError(estXYk5, actualXY)

  return(returnVal)

}

# Load Data

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

# Exclude cd RMSE

offlineMinusCd = offlineAll[ offlineAll$mac != "00:0f:a3:39:dd:cd", ]
rmse_minus_cd = errorFunc(offline = offlineMinusCd, numberMac = 6)
print(rmse_minus_cd)
# Exclude c0 RMSE

offlineMinusC0 = offlineAll[ offlineAll$mac != "00:0f:a3:39:e1:c0", ]
rmse_minus_c0 = errorFunc(offline = offlineMinusC0, numberMac = 6)
print(rmse_minus_c0)
# All RMSE

rmse_all = errorFunc(offline = offlineAll)
print(rmse_all)
# Weighted Distance Method

# Find nearest neighbor
findNN_dist = 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(list(trainSubset[closest, 1:3 ], dists[order(dists)]))
}

byLocAngleAP = with(offlineAll,
                    by(offlineAll, 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)

  macs = unique(offlineSummary$mac)

  online = readData("Data/online.final.trace.txt", subMacs = macs)
  online$posXY = paste(online$posX, online$posY, sep = "-")

  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 = 7,
                          dimnames = list(ans$posXY, names(avgSS)))
                    cbind(ans, y)
                  }))

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

```
  actualXY = onlineSummary[ , c("posX", "posY")]

# predict XY
predXY_dist = function(newSignals, newAngles, trainData,
                  numAngles = 1, k = 3, columns){

  closeXY = list(length = nrow(newSignals))
  weightXY = list(length = nrow(newSignals))
  finalXY = list(length = nrow(newSignals))

  for (i in 1:nrow(newSignals)) {
    trainSS = selectTrain(newAngles[i], trainData, m = numAngles, columns=columns)
    result = findNN_dist(newSignal = as.numeric(newSignals[i, ]), trainSS)
    closeXY[[i]] = result[[1]]
    weightXY[[i]] = 1/result[[2]]

  }

  for(i in 1:length(closeXY)) {
    finalXY[[i]]=closeXY[[i]][1:k,2:3]*weightXY[[i]]
  }

  estXY = lapply(finalXY,
                  function(x) sapply(x,
                                     function(x) sum(x)))

  estXY_dist = do.call("rbind", estXY)
  return(estXY_dist)
}

estXYk5_dist = predXY_dist(newSignals = onlineSummary[ , 6:11],
                   newAngles = onlineSummary[ , 4],
                   offlineSummary, numAngles = 3, k = 5, columns = 7)

RMSEwt = calcError(estXYk5_dist,actualXY)

print(paste("The Weighted RMSE is ",toString(round(RMSEwt))))

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")
}
predXY = function(newSignals, newAngles, trainData,
                  numAngles = 1, k = 3){

  closeXY = list(length = nrow(newSignals))

  for (i in 1:nrow(newSignals)) {
    trainSS = selectTrain(newAngles[i], trainData, m = numAngles)
    closeXY[[i]] =
      findNN(newSignal = as.numeric(newSignals[i, ]), trainSS)
  }

  estXY = lapply(closeXY,
                 function(x) sapply(x[ , 2:3],
                                    function(x) mean(x[1:k])))
  estXY = do.call("rbind", estXY)
  return(estXY)
}
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)
}

# k-folds-prep

byLocAngleAP = with(offlineMinusCd,
                    by(offlineMinusCd, 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)
v = 11
permuteLocs = sample(unique(offlineSummary$posXY))
permuteLocs = matrix(permuteLocs, ncol = v,
                     nrow = floor(length(permuteLocs)/v))

onlineFold = subset(offlineSummary, posXY %in% permuteLocs[ , 1])



offline = readData()
```

```
offline$posXY = paste(offline$posX, offlineAll$posY, sep = "-")
str(offline)
offline = offline[ offline$mac != "00:0f:a3:39:dd:cd", ]
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 = 3, k = 3)

actualFold = onlineFold[ , c("posX", "posY")]
calcError(estFold, actualFold)
K = 20
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 = 3, k = k)
    err[k] = err[k] + calcError(estFold, actualFold)
  }
}
oldPar = par(mar = c(4, 3, 1, 1))
plot(y = err, x = (1:K),  type = "l", lwd= 2,
     ylim = c(1000, 2100),
     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))
par(oldPar)
```