Crystal Mosley
MSDS 454 Sec 55

Individual Assignment #2

The Modeling Problem

Due to the growth of LANs and WANs inside of every type of building imaginable, statistical interest has spawned to try and find a reliable way of tracking people and things inside offices, school, stores, etc. Global Positioning Systems (GPS) have a hard time working within buildings; therefore, Indoor Positioning Systems (IPS) are often used to help make the tracking of people and things come alive via Wi-Fi signals and network access points. We will go through the process of cleaning the raw data, exploring the MAC (media access control) addresses provided within the data, explore the positioning of the hand-held devices, signal strength analysis and relationships, until finally we get to the modeling of K-Nearest Neighbors (KNN). KNN is a non-parametric, lazy learning algorithm. Its purpose is to use a database in which the data points are separated into several classes to predict the classification of a new sample point. For our KNN models, we will explore 3 types: Euclidean distance, absolute value distance, and the weighted distance.

The Data

The data utilized for this analysis are the online and offline traces provided by Case Studies in Data Science with R. The offline data set contained over 1 million observations and 10 character-valued variables. We converted the three position variables, signal, and time variables to numeric, and changed the rest to factors. We also removed the adhoc variables for measurements since this analysis will not utilize them. Being that the variables were not reader friendly, we also gave them better names. Table 1 displays the newly named variables, type, and new name descriptions.

Table 1: Variables from offline trace

Variable	Type	Description
time	Numeric	Timestamp in milliseconds since midnight, January 1, 1970
scanMac	Factor	MAC address from hand-held devices for measurements
posX	Numeric	Hand-held device location
posY	Numeric	Hand-held device location
posZ	Numeric	Hand-held device location
orientation	Numeric	Position of each hand-held device in 45-degree increments
mac	Factor	MAC address
signal	Numeric	Measurements taken from each access point from the orientation
channel	Factor	MAC address channel used

When exploring the *orientation* variable, there were only supposed to be 8 values; however, 203 were found. So, we plotted an empirical distribution function of orientation. As **Figure 1** shows, there are 8 orientations that are 45 degrees apart and the 0 orientation is split into two groups – possibly throwing off the length. To alleviate this, we created a *roundOrientation()* and created a boxplot to check the results, **Figure 2**.

Figure 1: ECDF of Orientation for devices

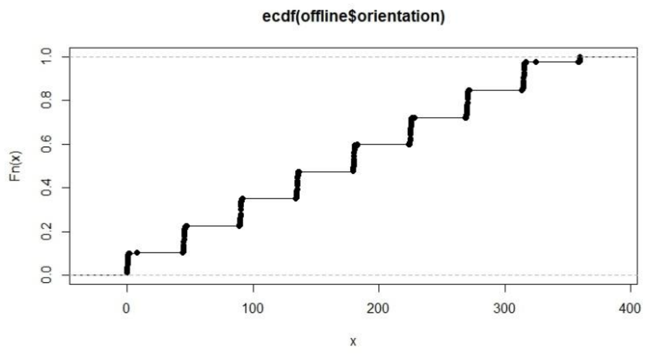
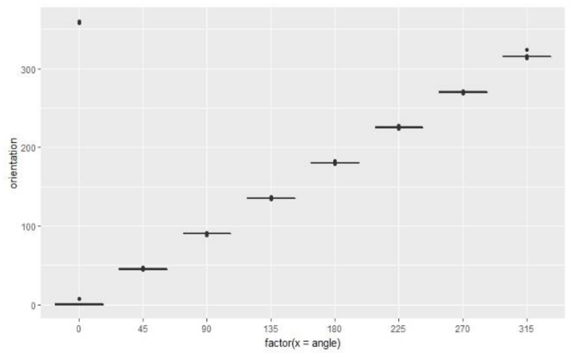


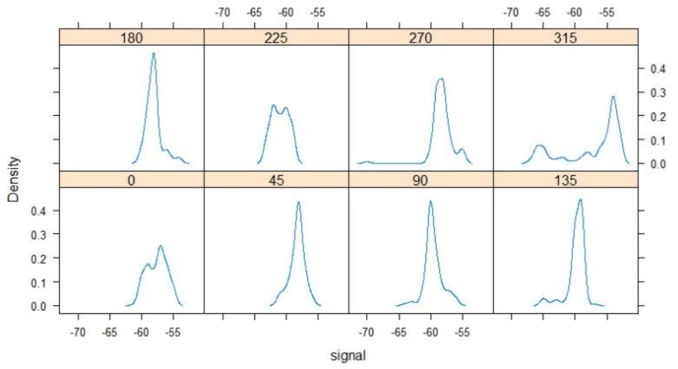
Figure 2: Boxplot of Orientation for devices, fixed



Exploratory Data Analysis

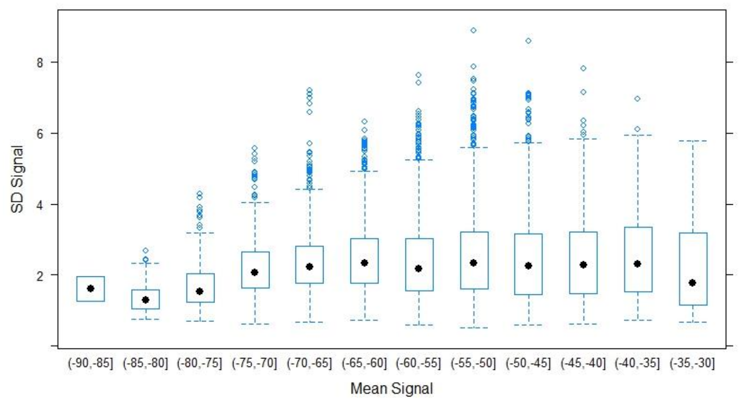
Non-normal, multi-modal signal distributions are found when considering signal strength and the angle of the mobile at each (x,y) location reading. This indicates angle influences signal strength. **Figure 3** displays this analysis finding for one access point at a single (x,y) position.

Figure 3: Angle-based density plot



Next, we created a box-and-whisker plot for displaying the standard deviation for each average signal, calculated in a summary. **Figure 4** shows how variable signal ranges can be – the more variables there are, the stronger the signals will be.

Figure 4: Standard deviation for each avg signal – b&w plot



Next, we moved onto finding the nearest neighbors – KNN's. This method takes training data where the signal is measured to several access points from known positions in a building. When we get a new observation, a new set of signal strengths for a location that is not known, we find the observation in our training data that is closest to this new observation. For KNN's where k > 1, we find the k closest training points and estimate the new observations position by a combination of the positions of the k training points. Our goals are to find records in our offline data (training set) that are similar in nature to our new observations. **Table 2** displays the variables created for each KNN type on the training data. **Table 3** displays the test KNN variables for both k=1 and k=3.

Table 2: All KNN variables created

KNN Type	KNN Variable	KNN Predicted Variable
Euclidian	findKNN1	predKNN1
Absolute	findKNN2	predKNN2
Weighted	findWeightedKNN	predWeightedKNN

Table 3: All KNN variables on test data for k=1, k=3

KNN Type	K=1	K=3
Euclidian (L2-norm)	estXYk1_knn1	estXYk3_knn1
Absolute (L1-norm)	estXYk1_knn2	estXYk3_knn2
Weighted	estXYk1_wtd	estXYk3_wtd

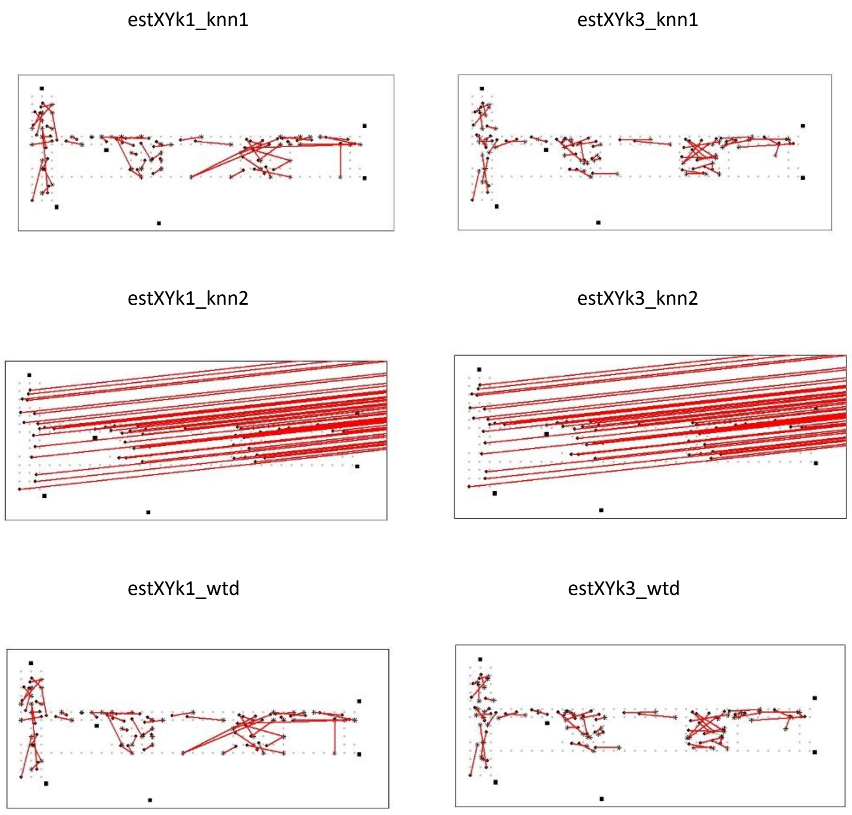
Using the test KNN variables, we explored the calculated errors on the floor via multiple plots showing the test locations (black dots) to their predicted locations (asterisks). The grey dots indicate offline locations and the black squares indicate the six access points for the three closest angles where the test data was measured – see **Figure 5**. Between the three KNN types, the calculated errors came out as follows:

sapply(list(estXYk1_knn1, estXYk3_knn1), calcError, actualXY) – 659 307

sapply(list(estXYk1_knn2, estXYk3_knn2), calcError, actualXY) – 1.7e+11 1.7e+11 sapply(list(estXYk1_wtd, estXYk3_wtd), calcError, actualXY) – 659 302

As can be seen, the L1-norm KNN calculated errors very poorly, and I'm not sure what went wrong within the calculation. We can see that the three nearest neighbors gave tighter looking clusters than the single nearest neighbors.

Figure 5: Calculated error plots for Table 3 data



Model Comparison

To identify which model performed best after fitting all models, I used the RSME (root mean square error) and MAE (mean absolute error) of the test set as shown in **Table 4**. Out of each KNN model, the Weighted model performed best; however, with the L1-norm showing an oddity, and we cannot be certain that the performance of this model did poorly against the Weighted model.

Table 4: Model Performance

KNN Model	RMSE	MAE
L2-norm	1314	1307
L1-norm	3.7e+11	3.7e+11
Weighted	1261	1255

Visualizing the SSE (sum of squared errors) for each of the three error KNN's (err1, err2, errWTD), we can see err1 (**Figure 6**) errors decreasing to about 6 neighbors and possibly level out until about 9 neighbors, and begin to increase at 10 neighbors because things start to expand geographically; err2 (**Figure 7**) shows significant error in that the calculation might be wrong for L1-norm since the error is a straight vertical line; errWTD (**Figure 8**) shows about the same as err1, however, it stays leveled out until around 11 or 12 neighbors and slowly starts to incline just as err1.

Figure 6: SSE x num of neighbors (err1)

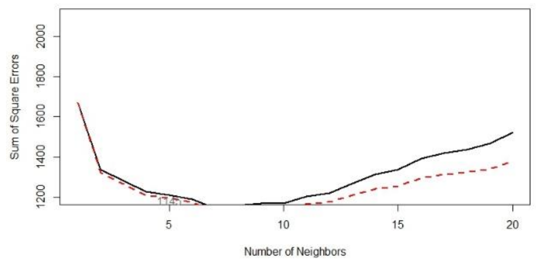


Figure 7: SSE x num of neighbors (err2)

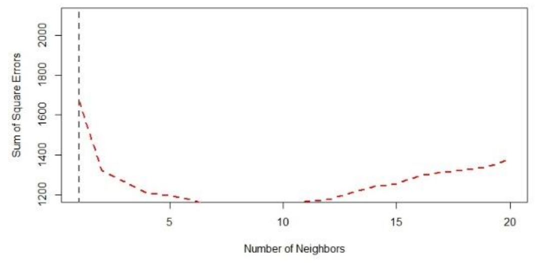
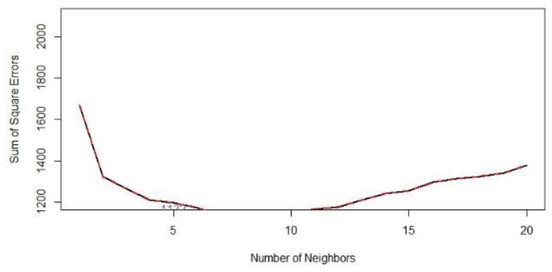


Figure 8: SSE x num of neighbors (errWTD)



Conclusion

We formatted the offline and online data into clean data frames. Different variables were explored to see how each would affect signal strength. We used the KNN method of three different types to build prediction models based on the offline data and tested each model with the online data. We found that we can minimize calculation error by employing the k=3 and angles = 3 into the training models.

Appendix A: R Code (trimmed)

```
#---------------------------------
#Model 1 - L2 knn - sqrt(sum(x^2))
#---------------------------------

findKNN1 = 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 ])
}

predKNN1 = 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]] =
      findKNN1(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)
}

#--------------------------------
#Model 2 - L1 knn - sum(abs(x^2))
#--------------------------------

findKNN2 = 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 ])
}

predKNN2 = 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]] =
      findKNN2(newSignal = as.numeric(newSignals[i, ]), trainSS)
  }

  estXY = lapply(closeXY,
               function(x) sapply(x[ , 2:3],
                                  function(x) sum(abs(x^2))))
  estXY = do.call("rbind", estXY)
  return(estXY)
}

#-------------------------------------------------------------
#Model 3 - Weighted knn - weight = as.numeric(1/dists[closest])
#-------------------------------------------------------------

findWeightedKNN = 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)
  closeXY = trainSubset[closest, 1:3 ]
  weight = as.numeric(1/dists[closest])
  return(cbind(closeXY, weight))
}

predWeightedKNN = 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)
    base = findWeightedKNN(newSignal = as.numeric(newSignals[i, ]), trainSS)
    wts = append(base[1:k, 4]/sum(base[1:k, 4]), rep(0, nrow(base)-k))
    base[, 2:3] = base[, 2:3]*wts
    closeXY[[i]] = base[,1:3]
  }

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

```
#Testing on the test set

estXYk1_knn1 = predKNN1(newSignals = onlineSummary[ , 6:11],
               newAngles = onlineSummary[ , 4],
               offlineSummary, numAngles = 3, k = 1)

estXYk3_knn1 = predKNN1(newSignals = onlineSummary[ , 6:11],
               newAngles = onlineSummary[ , 4],
               offlineSummary, numAngles = 3, k = 3)

estXYk1_knn2 = predKNN2(newSignals = onlineSummary[ , 6:11],
                    newAngles = onlineSummary[ , 4],
                    offlineSummary, numAngles = 3, k = 1)

estXYk3_knn2 = predKNN2(newSignals = onlineSummary[ , 6:11],
                    newAngles = onlineSummary[ , 4],
                    offlineSummary, numAngles = 3, k = 3)

estXYk1_wtd = predWeightedKNN(newSignals = onlineSummary[ , 6:11],
                    newAngles = onlineSummary[ , 4],
                    offlineSummary, numAngles = 3, k = 1)

estXYk3_wtd = predWeightedKNN(newSignals = onlineSummary[ , 6:11],
                   newAngles = onlineSummary[ , 4],
                   offlineSummary, numAngles = 3, k = 3)

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

#-----------------
#Error calculation
#-----------------

rmse <- function(error){
  sqrt(mean(error^2))
}

mae <- function(error){
  mean(abs(error))
}

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

sapply(list(estXYk1_knn1, estXYk3_knn1), calcError, actualXY)
sapply(list(estXYk1_knn2, estXYk3_knn2), calcError, actualXY)
sapply(list(estXYk1_wtd, estXYk3_wtd), calcError, actualXY)
```

```
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)
  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")
}

trainPoints = offlineSummary[ offlineSummary$angle == 0 &
                               offlineSummary$mac == "00:0f:a3:39:e1:c0" ,
                              c("posX", "posY")]

pdf(file="GEO_FloorPlanK1Errors_estXYk1_knn1.pdf", width = 10, height = 7)
oldPar = par(mar = c(1, 1, 1, 1))
floorErrorMap(estXYk1_knn1, onlineSummary[ , c("posX","posY")],
             trainPoints = trainPoints, AP = AP)
par(oldPar)
dev.off()

pdf(file="GEO_FloorPlanK1Errors_estXYk3_knn1.pdf", width = 10, height = 7)
oldPar = par(mar = c(1, 1, 1, 1))
floorErrorMap(estXYk3_knn1, onlineSummary[ , c("posX","posY")],
             trainPoints = trainPoints, AP = AP)
par(oldPar)
dev.off()

pdf(file="GEO_FloorPlanK1Errors_estXYk1_knn2.pdf", width = 10, height = 7)
oldPar = par(mar = c(1, 1, 1, 1))
floorErrorMap(estXYk1_knn2, onlineSummary[ , c("posX","posY")],
             trainPoints = trainPoints, AP = AP)
par(oldPar)
dev.off()

pdf(file="GEO_FloorPlanK1Errors_estXYk3_knn2.pdf", width = 10, height = 7)
oldPar = par(mar = c(1, 1, 1, 1))
floorErrorMap(estXYk3_knn2, onlineSummary[ , c("posX","posY")],
             trainPoints = trainPoints, AP = AP)
par(oldPar)
```

```
dev.off()

pdf(file="GEO_FloorPlanK1Errors_estXYk1_wtd.pdf", width = 10, height = 7)
oldPar = par(mar = c(1, 1, 1, 1))
floorErrorMap(estXYk1_wtd, onlineSummary[ , c("posX","posY")],
            trainPoints = trainPoints, AP = AP)
par(oldPar)
dev.off()

pdf(file="GEO_FloorPlanK1Errors_estXYk3_wtd.pdf", width = 10, height = 7)
oldPar = par(mar = c(1, 1, 1, 1))
floorErrorMap(estXYk3_wtd, onlineSummary[ , c("posX","posY")],
            trainPoints = trainPoints, AP = AP)
par(oldPar)
dev.off()



#Cross-validation

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)
}
```

```
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])

#Folds using k=3

estFoldKNN1 = predKNN1(newSignals = onlineFold[ , 6:11],
                newAngles = onlineFold[ , 4],
                offlineFold, numAngles = 3, k = 3)

estFoldKNN2 = predKNN2(newSignals = onlineFold[ , 6:11],
                    newAngles = onlineFold[ , 4],
                    offlineFold, numAngles = 3, k = 3)

estFoldWTD = predWeightedKNN(newSignals = onlineFold[ , 6:11],
                    newAngles = onlineFold[ , 4],
                    offlineFold, numAngles = 3, k = 3)

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

calcError(estFoldKNN1, actualFold)
calcError(estFoldKNN2, actualFold)
calcError(estFoldWTD, actualFold)

#Test k out to 20 neighbors

K = 20
err1 = rep(0, K)
err2 = rep(0, K)
errWTD = 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) {
    estFoldKNN1 = predKNN1(newSignals = onlineFold[ , 6:11],
                  newAngles = onlineFold[ , 4],
                  offlineFold, numAngles = 3, k = k)
    err1[k] = err1[k] + calcError(estFoldKNN1, actualFold)

    estFoldKNN2 = predKNN2(newSignals = onlineFold[ , 6:11],
                    newAngles = onlineFold[ , 4],
                    offlineFold, numAngles = 3, k = k)
    err2[k] = err2[k] + calcError(estFoldKNN2, actualFold)

    estFoldWTD = predWeightedKNN(newSignals = onlineFold[ , 6:11],
                    newAngles = onlineFold[ , 4],
                    offlineFold, numAngles = 3, k = k)
    errWTD[k] = errWTD[k] + calcError(estFoldWTD, actualFold)
  }
}

pdf(file = "Geo_CVChoiceOfK_err1.pdf", width = 10, height = 6)
oldPar = par(mar = c(4, 3, 1, 1))
plot(y = err1, x = (1:K),  type = "l", lwd= 2,
     ylim = c(1200, 2100),
     xlab = "Number of Neighbors",
     ylab = "Sum of Square Errors")
lines(y=errWTD, x=1:K, lty = 2, lwd=2, col='red')

rmseMin1 = min(err1)
kMin = which(err1 == rmseMin1)[1]
segments(x0 = 0, x1 = kMin, y0 = rmseMin1, col = gray(0.4),
         lty = 2, lwd = 2)
segments(x0 = kMin, x1 = kMin, y0 = 1100,  y1 = rmseMin1,
         col = grey(0.4), lty = 2, lwd = 2)

text(x = kMin - 2, y = rmseMin1 + 40,
     label = as.character(round(rmseMin1)), col = grey(0.4))
par(oldPar)
dev.off()

pdf(file = "Geo_CVChoiceOfK_err2.pdf", width = 10, height = 6)
oldPar = par(mar = c(4, 3, 1, 1))
plot(y = err2, x = (1:K),  type = "l", lwd= 2,
     ylim = c(1200, 2100),
     xlab = "Number of Neighbors",
     ylab = "Sum of Square Errors")
lines(y=errWTD, x=1:K, lty = 2, lwd=2, col='red')

rmseMin2 = min(err2)
kMin = which(err2 == rmseMin2)[1]
segments(x0 = 0, x1 = kMin, y0 = rmseMin2, col = gray(0.4),
        lty = 2, lwd = 2)
segments(x0 = kMin, x1 = kMin, y0 = 1100,  y1 = rmseMin2,
        col = grey(0.4), lty = 2, lwd = 2)

text(x = kMin - 2, y = rmseMin2 + 40,
    label = as.character(round(rmseMin2)), col = grey(0.4))
par(oldPar)
dev.off()

pdf(file = "Geo_CVChoiceOfK_errWTD.pdf", width = 10, height = 6)
oldPar = par(mar = c(4, 3, 1, 1))
plot(y = errWTD, x = (1:K),  type = "l", lwd= 2,
    ylim = c(1200, 2100),
    xlab = "Number of Neighbors",
    ylab = "Sum of Square Errors")
lines(y=errWTD, x=1:K, lty = 2, lwd=2, col='red')

rmseMin3 = min(errWTD)
kMin = which(errWTD == rmseMin3)[1]
segments(x0 = 0, x1 = kMin, y0 = rmseMin3, col = gray(0.4),
        lty = 2, lwd = 2)
segments(x0 = kMin, x1 = kMin, y0 = 1100,  y1 = rmseMin3,
        col = grey(0.4), lty = 2, lwd = 2)

text(x = kMin - 2, y = rmseMin3 + 40,
    label = as.character(round(rmseMin3)), col = grey(0.4))
par(oldPar)
dev.off()

rmseMin1
rmseMin2
rmseMin3

rmse(err1)
rmse(err2)
rmse(errWTD)

mae(err1)
mae(err2)
mae(errWTD)

#-----------------------
#KNN's with k=5, angles=3
#-----------------------

estXYknn1_5 = predKNN1(newSignals = onlineSummary[ , 6:11],
              newAngles = onlineSummary[ , 4],
              offlineSummary, numAngles = 3, k = 5)
```

```
estXYknn2_5 = predKNN2(newSignals = onlineSummary[ , 6:11],
                    newAngles = onlineSummary[ , 4],
                    offlineSummary, numAngles = 3, k = 5)
estXYknnWTD_5 = predWeightedKNN(newSignals = onlineSummary[ , 6:11],
                    newAngles = onlineSummary[ , 4],
                    offlineSummary, numAngles = 3, k = 5)

calcError(estXYknn1_5, actualXY)
calcError(estXYknn2_5, actualXY)
calcError(estXYknnWTD_5, actualXY)

# sapply(list(estXYk1_wtd, estXYknn1_5), calcError, actualXY)
# sapply(list(estXYk1_wtd, estXYknn2_5), calcError, actualXY)
# sapply(list(estXYk1_wtd, estXYknnWTD_5), calcError, actualXY)
```