Branch: **master**

Find file | Copy path

Indoor-Positioning-Via-Wifi-Fingerprinting / 03 -WifiLocationing.R

armin-talic R code

095cb1c on Feb 10, 2018

1 contributor

Raw | Blame | History

1607 lines (1240 sloc) 57.1 KB

```
# Indoor locationing via wifi fingerprinting
# Armin Talic


# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ LIBRARIES ~~~~~~~~~~~~~~~~~~~~~~~~~~~
library(readr)
library(dplyr)
library(scatterplot3d)
library(ggplot2)
library(corrplot)
library(caret)
library(som)
library(data.table)


# Set working directory

# Import training dataset
```

```
trainingData <- read_csv("trainingData.csv")

# Import validation dataset
testData <- read_csv("validationData.csv")

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ DATA PREPROCESSING ~~~~~~~~~~~~~~~~~~~

# Remove columns (WAP) where all the values = 100 (WAP was not detected)
# Training dataset
uniquelength <- sapply(trainingData,function(x) length(unique(x)))
trainingData <- subset(trainingData, select=uniquelength>1)
# Test dataset
uniquelength <- sapply(testData,function(x) length(unique(x)))
testData <- subset(testData, select=uniquelength>1)

# Remove rows (WAP) where all the values = 100 (WAP was not detected)
# Training dataset
keep <- apply(trainingData[,1:465], 1, function(x) length(unique(x[!is.na(x)]))
trainingData[keep, ]

# # Test dataset
keep <- apply(testData[,1:465], 1, function(x) length(unique(x[!is.na(x)])) != 1
testData[keep, ]

# Converting data types
# Training dataset
trainingData$FLOOR <- as.factor(trainingData$FLOOR)
trainingData$BUILDINGID <- as.factor(trainingData$BUILDINGID)
trainingData$RELATIVEPOSITION <- as.factor(trainingData$RELATIVEPOSITION)
trainingData$USERID <- as.factor(trainingData$USERID)
trainingData$PHONEID <- as.factor(trainingData$PHONEID)

# Test dataset
testData$FLOOR <- as.factor(testData$FLOOR)
testData$BUILDINGID <- as.factor(testData$BUILDINGID)
testData$PHONEID <- as.factor(testData$PHONEID)


# Change WAP values so that no signal is 0 and highest signal is 104
# Training Data
trainingData[trainingData == 100] <- -105
trainingData[,1:465] <- trainingData[,1:465] + 105

# Test data
testData[testData == 100] <- -105
```

```r
testData[,1:367] <- testData[,1:367] + 105

# Check distribution of signal strength
# traning data
x <- trainingData[,1:465]
x <- stack(x)

x <- x[-grep(0, x$values),]
hist(x$values, xlab = "WAP strength", main = "Distribution of WAPs signal stengt

# test data
y <- testData[,1:367]
y <- stack(y)

y <- y[-grep(0, y$values),]
hist(y$values, xlab = "WAP strength", main = "Distribution of WAPs signal stengt

ggplot() +
  geom_histogram(data = x, aes(values), fill = "red", alpha = 1, binwidth = 5) +
  geom_histogram(data = y, aes(values), fill = "blue", alpha = 1, binwidth = 5)
  ggtitle("Distribution of WAPs signal strength (Training and Test sets)") +
  xlab("WAP strength")


# Check distribution of how many WAPs have signal
# TRAINING SET
trainingData$count <- rowSums(trainingData[, 1:465] != 0)
ggplot(trainingData, aes(count, fill = as.factor(trainingData$BUILDINGID))) +
  geom_histogram(binwidth = 2)+
  ggtitle("Number of WAPs detected per building (Training set)") +
  scale_fill_manual(name="Buildings", values = c("0" = "royalblue2",
                             "1" = "firebrick2",
                             "2" = "springgreen1"),
                    labels=c("Building 1","Building 2", "Building 3"))

# TEST SET
testData$count <- rowSums(testData[, 1:367] != 0)
ggplot(testData, aes(count, fill = as.factor(testData$BUILDINGID))) +
  geom_histogram(binwidth = 2)+
  ggtitle("Number of WAPs detected per building (Test set)") +
  scale_fill_manual(name="Buildings", values = c("0" = "royalblue2",
                                                  "1" = "firebrick2",
                                                  "2" = "springgreen1"),
                    labels=c("Building 1","Building 2", "Building 3"))

```

```
# Convert Longitude and Latitude values to absolute values
# Latitude
trainingData$LATITUDE <- trainingData$LATITUDE - min(trainingData$LATITUDE)
testData$LATITUDE <- testData$LATITUDE - min(testData$LATITUDE)


# Longitude
trainingData$LONGITUDE <- trainingData$LONGITUDE - min(trainingData$LONGITUDE)
testData$LONGITUDE <- testData$LONGITUDE - min(testData$LONGITUDE)

# check maximum values for longitude and latitude in training and test sets
max(trainingData$LONGITUDE)
max(testData$LONGITUDE)
#remove all rows where LONGITUDE values from test set that are higher than in tr
testData<-testData[!(testData$LONGITUDE > 390.5194), ]

#max(trainingData$LATITUDE)
#max(testData$LATITUDE)

# Locations at which users logged in
# Red colour is outside the room, black inside
p <- ggplot(trainingData, aes(trainingData$LONGITUDE, trainingData$LATITUDE))
p + geom_point(colour = as.factor(trainingData$RELATIVEPOSITION)) +
  xlim(0, 400) +
  ylim(0, 300) +
  xlab("Longitude") +
  ylab("Latitude") +
  ggtitle ("Locations at which users loged in (Training dataset)")

# Training and Validation log in locations
ggplot() +
  geom_point(data = trainingData, aes(x = LONGITUDE, y = LATITUDE, colour = "Tra
  geom_point(data = testData, aes(x = LONGITUDE, y = LATITUDE, colour = "Test da
  ggtitle("Log In Locations (Training and Test sets)")

# MODEL TO PREDICT BUILDING
# Split Training and Test sets

training <- trainingData[ ,1:469]

# Import validation dataset
validation <- testData

# Drop columns from validation that do not match with traning set
```

```
cols_to_keep <- intersect(colnames(training),colnames(validation))
training <- training[,cols_to_keep, drop=FALSE]
validation <- validation[,cols_to_keep, drop=FALSE]

set.seed(123)
trainIndex <- createDataPartition(y = training$BUILDINGID, p = 0.75,
                                  list = FALSE)


# Training and Test sets

trainSet <- training [trainIndex,]
testSet <- training [-trainIndex,]

# K-NN

df <- trainSet


set.seed(123)
ctrl <- trainControl(method="cv",number = 5)
knnFit <- train((BUILDINGID ~ .), data = df, method = "knn", trControl = ctrl, t

#Output of kNN fit
knnFit

# test the k-NN model
knnPredict <- predict(knnFit,newdata = testSet)

# confusion matrix to see accuracy value and other parameter values
knnCM <-confusionMatrix(knnPredict, testSet$BUILDINGID)

# Check results on validation dataset
# Convert data types in validation dataset
validation$BUILDINGID <- as.factor(validation$BUILDINGID)

# Apply k-NN model to the validation data
knnPredicttest <- predict(knnFit,newdata = validation)
knnCM <-confusionMatrix(knnPredicttest, validation$BUILDINGID)
knnCM
# Performance:
#Confusion Matrix     0    1    2
#                0  536    0    0
#                1    0  307    0
#                2    0    0  268
```

```r
# Accuracy 1
# Kappa 1


# Create dataset for each building and floor
# Create a dataset for each building
Building1 <- subset(trainingData, BUILDINGID == 0)
Building2 <- subset(trainingData, BUILDINGID == 1)
Building3 <- subset(trainingData, BUILDINGID == 2)

# Remove columns (WAP) where all the values = 100 (WAP was not detected)
# Building 1
uniquelength <- sapply(Building1,function(x) length(unique(x)))
Building1 <- subset(Building1, select=uniquelength>1)

# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(Building1[,1:200], 1, function(x) length(unique(x[!is.na(x)])) !=
Building1[keep, ]

# Building 2
uniquelength <- sapply(Building2,function(x) length(unique(x)))
Building2 <- subset(Building2, select=uniquelength>1)

# Building 3
uniquelength <- sapply(Building3,function(x) length(unique(x)))
Building3 <- subset(Building3, select=uniquelength>1)

# 3D scatterplot of floors and log in locations
# # # Building 1
#  Building1Floor1 <- subset(Building1, FLOOR == 0)
#  Building1Floor2 <- subset(Building1, FLOOR == 1)
#  Building1Floor3 <- subset(Building1, FLOOR == 2)
#  Building1Floor4 <- subset(Building1, FLOOR == 3)

# # Building 2
# Building2Floor1 <- subset(Building2, FLOOR == 0)
# Building2Floor2 <- subset(Building2, FLOOR == 1)
# Building2Floor3 <- subset(Building2, FLOOR == 2)
# Building2Floor4 <- subset(Building2, FLOOR == 3)
#
# # Building 3
# Building3Floor1 <- subset(Building3, FLOOR == 0)
# Building3Floor2 <- subset(Building3, FLOOR == 1)
# Building3Floor3 <- subset(Building3, FLOOR == 2)
# Building3Floor4 <- subset(Building3, FLOOR == 3)
```

```
# Building3Floor5 <- subset(Building3, FLOOR == 4)

# # Remove columns (WAP) where all the values = 100 (WAP was not detected)
# uniquelength <- sapply(Building1Floor1,function(x) length(unique(x)))
# Building1Floor1 <- subset(Building1Floor1, select=uniquelength>1)
#
# # Remove rows (WAP) where all the values = 100 (WAP was not detected)
# keep <- apply(Building1Floor1[,1:200], 1, function(x) length(unique(x[!is.na(x
# Building1Floor1[keep, ]

# Building 1 inspection
unique(Building1$USERID) # 2 different user IDs
unique(Building1$PHONEID) # 2 different phone IDs

# Building 2 inspection
# length(unique(Building2$USERID)) # 12 different user IDs
# length(unique(Building2$PHONEID)) # 11 different phone IDs

# Building 3 inspection
# length(unique(Building3$USERID)) # 16 different user IDs
# length(unique(Building3$PHONEID)) # 15 different phone IDs

# # Plots
# # Building 1
# Building1Floor1$z <- 1
# Building1Floor2$z <- 2
# Building1Floor3$z <- 3
# Building1Floor4$z <- 4
#
# buildplot1 <- rbind(Building1Floor1,Building1Floor2)
# buildplot1 <- rbind(buildplot1, Building1Floor3)
# buildplot1 <- rbind(buildplot1, Building1Floor4)
# buildplot1 <- buildplot1[,521:530]
# z <- buildplot1$z
# x <- buildplot1$LONGITUDE
# y <- buildplot1$LATITUDE
# scatterplot3d(x, y, z, pch = 20, angle = 45, color = buildplot1$RELATIVEPOSITI

# # Building 2
# Building2Floor1$z <- 1
# Building2Floor2$z <- 2
# Building2Floor3$z <- 3
# Building2Floor4$z <- 4
#
# buildplot2 <- rbind(Building2Floor1,Building2Floor2)
```

```
# buildplot2 <- rbind(buildplot2, Building2Floor3)
# buildplot2 <- rbind(buildplot2, Building2Floor4)
# buildplot2 <- buildplot2[,521:530]
# c <- buildplot2$z
# a <- buildplot2$LONGITUDE
# b <- buildplot2$LATITUDE
# scatterplot3d(a, b, c, angle = 60, pch = buildplot2$z, color = buildplot2$RELA
#
# # Building 3
# Building3Floor1$z <- 1
# Building3Floor2$z <- 2
# Building3Floor3$z <- 3
# Building3Floor4$z <- 4
# Building3Floor5$z <- 5
#
# buildplot3 <- rbind(Building3Floor1,Building3Floor2)
# buildplot3 <- rbind(buildplot3, Building3Floor3)
# buildplot3 <- rbind(buildplot3, Building3Floor4)
# buildplot3 <- rbind(buildplot3, Building3Floor5)
# buildplot3 <- buildplot3[,521:530]
# c <- buildplot3$z
# a <- buildplot3$LONGITUDE
# b <- buildplot3$LATITUDE
# scatterplot3d(a, b, c, angle = 20, pch = buildplot3$z, color = buildplot3$RELA


# Check where is the highest signal strength
which(Building3[,1:119] == 105)
which(Building3[,1:119] == 105, arr.ind=TRUE)

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Signal greater than 60 ~~~~~~~~~~~~~~~~~~
abc <- which(apply(trainingData[, 1:465], 1, function(x) length(which(x > 60)))
GoodSignal <- trainingData[abc, ]

gs <- ggplot(GoodSignal, aes(GoodSignal$LONGITUDE, GoodSignal$LATITUDE))
gs + geom_point(colour = as.factor(GoodSignal$RELATIVEPOSITION)) +
  ggtitle("Log in locations where WAP signal was high") +
  xlab("Longitude") +
  ylab("Latitude")


# Remove columns (WAP) where all the values = 0 (WAP was not detected)
uniquelength <- sapply(GoodSignal,function(x) length(unique(x)))
GoodSignal <- subset(GoodSignal, select=uniquelength>1)

```

```r
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(GoodSignal[,1:183], 1, function(x) length(unique(x[!is.na(x)])) !=
GoodSignal[keep, ]


# Signal between 20 and 60
def <- which(apply(trainingData[, 1:465], 1, function(x) length(which(x > 20 & x
Mediumsignal <- trainingData[def, ]

ms <- ggplot(Mediumsignal, aes(Mediumsignal$LONGITUDE, Mediumsignal$LATITUDE))
ms + geom_point(colour = as.factor(Mediumsignal$RELATIVEPOSITION))

# Signal between 0 and 20
ghi <- which(apply(trainingData[, 1:465], 1, function(x) length(which(x > 0 & x
BadSignal <- trainingData[ghi, ]

bs <- ggplot(BadSignal, aes(BadSignal$LONGITUDE, BadSignal$LATITUDE))
bs + geom_point(colour = as.factor(BadSignal$RELATIVEPOSITION))


# THE FOLLOWING MODEL HAS LOW PERFORMACE SINCE THE DATA IS NOT NORMALIZED

# Build model to predict Floor for Building 1
# Split Training and Test sets
#
# Building1 <- subset(trainingData, BUILDINGID == 0)
# # Remove columns (WAP) where all the values = 0 (WAP was not detected)
# # Building 1
# uniquelength <- sapply(Building1,function(x) length(unique(x)))
# Building1 <- subset(Building1, select=uniquelength>1)
#
# # Remove rows (WAP) where all the values = 0 (WAP was not detected)
# keep <- apply(Building1[,1:200], 1, function(x) length(unique(x[!is.na(x)])) !
# Building1 <-  Building1[keep, ]
#
# Building1$FLOOR <- factor(Building1$FLOOR)
# cols <- c(1:200, 203)
# training <- Building1[ , cols]
# training$FLOOR <- factor(training$FLOOR)
#
# set.seed(123)
# trainIndex <- createDataPartition(y = training$FLOOR, p = 0.75,
#                                   list = FALSE)
#
#
```

```
# #                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~    Training and Test sets  ~~~~~
#
# trainSet <- training [trainIndex,]
# testSet <- training [-trainIndex,]
#
# #                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~    K-NN    ~~~~~~~~~~~~~~~~~~~~~
#
# df <- trainSet
#
#
# set.seed(123)
# ctrl <- trainControl(method="cv",number = 10)
# knnFit <- train((FLOOR ~ .), data = df, method = "knn", trControl = ctrl, tune
#
# #Output of kNN fit
# knnFit
# #write.csv(knnFit, file = 'knnperformance.csv')
#
# # test the k-NN model
# knnPredict <- predict(knnFit,newdata = testSet)
# postResample(knnPredict, testSet$FLOOR)
#
# # Check results on validation dataset
# # Import validation dataset
# validation <- testData
#
# # Drop columns from validation that do not match with traning set
# cols_to_keep <- intersect(colnames(training),colnames(validation))
# validation <- validation[,cols_to_keep, drop=FALSE]
#
# # Apply k-NN model to the validation data
# knnPredicttest <- predict(knnFit,newdata = validation)
# postResample(knnPredicttest, validation$FLOOR)
#
# # Result Accuracy     Kappa
# # ~~~~~~ 0.5094509 0.3861611
#
# #             ~~~~~~~~~~~~~~~~~~~~~~~~~~~~   Random Forest   ~~~~~~~~~~~~~~~~~
#
# set.seed(123)
# ctrl <- trainControl(method="cv", number = 10)
#
# # Random forest
# rfFit <- train(FLOOR ~ ., data = trainSet, method = "rf", trControl = ctrl, pr
# rfPredict <- predict(rfFit, newdata = testSet)
```

```
# rfCM <- confusionMatrix(rfPredict, testSet$FLOOR)
#
# rfPredicttest <- predict(rfFit, newdata = validation)
# postResample(knnPredicttest, validation$FLOOR)
# # Performance Accuracy     Kappa
# #             0.5112511 0.3883641


# THIS MODEL ALSO HAS LOW PERFORMACE
# # ~~~~~~~~~~~~~~~~~~~~~~~~~~ Model with Normalized rows (mean = 0, variance =
# som_row <- normalize(Building1[,1:200], byrow=TRUE)
# try <- as.data.frame(som_row)
# try$FLOOR <- Building1$FLOOR
# Building1[1:200] <- try[1:200]
#
# Building1$FLOOR <- factor(Building1$FLOOR)
# cols <- c(1:200, 203)
# training <- Building1[ , cols]
# training$FLOOR <- factor(training$FLOOR)
#
# set.seed(123)
# trainIndex <- createDataPartition(y = training$FLOOR, p = 0.75,
#                                    list = FALSE)
#
#
# #                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~    Training and Test sets  ~~~~~
#
# trainSet <- training [trainIndex,]
# testSet <- training [-trainIndex,]
#
# #                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~    K-NN    ~~~~~~~~~~~~~~~~~~~~~
#
# df <- trainSet
#
#
# set.seed(123)
# ctrl <- trainControl(method="cv",number = 10)
# knnFit <- train((FLOOR ~ .), data = df, method = "knn", trControl = ctrl, tune
#
# #Output of kNN fit
# knnFit
# #write.csv(knnFit, file = 'knnperformance.csv')
#
# # test the k-NN model
# knnPredict <- predict(knnFit,newdata = testSet)
```

```
# postResample(knnPredict, testSet$FLOOR)
#
# # Check results on validation dataset
# # Import validation dataset
# validation <- testData
#
# # Drop columns from validation that do not match with traning set
# cols_to_keep <- intersect(colnames(training),colnames(validation))
# validation <- validation[,cols_to_keep, drop=FALSE]
#
# # Apply k-NN model to the validation data
# knnPredicttest <- predict(knnFit,newdata = validation)
# postResample(knnPredicttest, validation$FLOOR)
#
# # Performance Accuracy     Kappa
# #             0.5724572 0.4031253


# ONCE THE KNN MODEL IS BUILT WITH NORMALIZED DATA, RANDOM FOREST AND GRADIENT BO
# ARE USED WITH THE NORMALIZED DATA
# #             ~~~~~~~~~~~~~~~~~~~~~~~~~~~~   Random Forest   ~~~~~~~~~~~~~~~~~
#
# set.seed(123)
# ctrl <- trainControl(method="cv", number = 10)
#
# # Random forest
# rfFit <- train(FLOOR ~ ., data = trainSet, method = "rf", trControl = ctrl, pr
# rfPredict <- predict(rfFit, newdata = testSet)
# rfCM <- confusionMatrix(rfPredict, testSet$FLOOR)
#
# rfPredicttest <- predict(rfFit, newdata = validation)
# postResample(rfPredicttest, validation$FLOOR)
#
# rfCM <-confusionMatrix(rfPredicttest, validation$FLOOR)
# rfCM
#
# # Performance Accuracy     Kappa
# #             0.4050405 0.2021011
#
# #             ~~~~~~~~~~~~~~~~~~~~~~~~~~~~  eXtreme Gradient Boosting  ~~~~~~~
#
# set.seed(123)
# ctrl <- trainControl(method="cv", number = 10)
#
# # Random forest
```

```r
# GBFit <- train(FLOOR ~ ., data = trainSet, method = "xgbTree", trControl = ctr
# GBPredict <- predict(GBFit, newdata = testSet)
# rfCM <- confusionMatrix(GBPredict, testSet$FLOOR)
#
# GBPredicttest <- predict(GBFit, newdata = validation)
# postResample(GBPredicttest, validation$FLOOR)
# # Performance Accuracy     Kappa
# #            0.5499550 0.3281139


# Model with scaled data (0 to 1)
# BUILDING 1
try <- Building1[,1:200]
# Remove columns (WAP) where all the values = 0 (WAP was not detected)
uniquelength <- sapply(try,function(x) length(unique(x)))
try <- subset(try, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(try[,1:200], 1, function(x) length(unique(x[!is.na(x)])) != 1)
try <-  try[keep, ]

# Normalize the data
data_norm <- as.data.frame(t(apply(try, 1, function(x) (x - min(x))/(max(x)-min(


# Build knn model
uniquelength <- sapply(Building1,function(x) length(unique(x)))
Building1 <- subset(Building1, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(Building1[,1:200], 1, function(x) length(unique(x[!is.na(x)])) != 
Building1 <-  Building1[keep, ]

Building1[,1:200] <- data_norm

Building1$FLOOR <- factor(Building1$FLOOR)
cols <- c(1:200, 203)
training <- Building1[ , cols]
training$FLOOR <- factor(training$FLOOR)

# Import validation dataset
validation <- subset(testData, BUILDINGID == 0)
testData1 <- subset(testData, BUILDINGID == 0)

# Remove columns (WAP) where all the values = 0 (WAP was not detected)
uniquelength <- sapply(validation,function(x) length(unique(x)))
validation <- subset(validation, select=uniquelength>1)
```

```
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(validation[,1:183], 1, function(x) length(unique(x[!is.na(x)])) !=
validation <-  validation[keep, ]

# Drop columns from validation that do not match with traning set
cols_to_keep <- intersect(colnames(training),colnames(validation))
training <- training[,cols_to_keep, drop=FALSE]
validation <- validation[,cols_to_keep, drop=FALSE]
validation <- as.data.frame(t(apply(validation[,1:139], 1, function(x) (x - min(

validation$FLOOR <- testData1$FLOOR
validation$FLOOR <- factor(validation$FLOOR)

set.seed(123)
trainIndex <- createDataPartition(y = training$FLOOR, p = 0.75,
                                  list = FALSE)


#                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~    Training and Test sets  ~~~~~~~

trainSet <- training [trainIndex,]
testSet <- training [-trainIndex,]

#                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~    K-NN    ~~~~~~~~~~~~~~~~~~~~~~~

df <- trainSet

checkMeans <- df
checkMeans[,1:139][checkMeans[,1:139] == 0] <- NA
rowMeans(checkMeans[,1:139], na.rm = T)
# Keep only rows where mean value of detected WAPs is more than 0.5
checkMeans <- subset(checkMeans, rowMeans(checkMeans[,1:139], na.rm = T) > 0.6)
checkMeans[is.na(checkMeans)] <- 0

df <- checkMeans


# set.seed(123)
# ctrl <- trainControl(method="cv",number = 10)
# knnFit <- train((FLOOR ~ .), data = df, method = "knn", trControl = ctrl, tune
#
# #Output of kNN fit
# knnFit
# #write.csv(knnFit, file = 'knnperformance.csv')
#
```

```
# # test the k-NN model
# knnPredict <- predict(knnFit,newdata = testSet)
# postResample(knnPredict, testSet$FLOOR)
#
# # Check results on validation dataset
# # Apply k-NN model to the validation data
# knnPredicttest <- predict(knnFit,newdata = validation)
# postResample(knnPredicttest, validation$FLOOR)
#
# knnCM <-confusionMatrix(knnPredicttest, validation$FLOOR)
# knnCM

#              ~~~~~~~~~~~~~~~~~~~~~~~~~~~~   Random Forest   ~~~~~~~~~~~~~~~~~~~~

set.seed(123)
ctrl <- trainControl(method="cv", number = 5)

# Random forest
rfFit <- train(FLOOR ~ ., data = trainSet, method = "rf", trControl = ctrl, preP
rfPredict <- predict(rfFit, newdata = testSet)
rfCM <- confusionMatrix(rfPredict, testSet$FLOOR)
plot(rfFit$finalModel)

rfPredicttest <- predict(rfFit, newdata = validation)
postResample(rfPredicttest, validation$FLOOR)

rfCM <-confusionMatrix(rfPredicttest, validation$FLOOR)
rfCM


# #             ~~~~~~~~~~~~~~~~~~~~~~~~~~~~  eXtreme Gradient Boosting  ~~~~~~~
#
# set.seed(123)
# ctrl <- trainControl(method="cv", number = 10)
#
# # Random forest
# GBFit <- train(FLOOR ~ ., data = trainSet, method = "xgbTree", trControl = ctr
# GBPredict <- predict(GBFit, newdata = testSet)
# rfCM <- confusionMatrix(GBPredict, testSet$FLOOR)
#
# GBPredicttest <- predict(GBFit, newdata = validation)
# postResample(GBPredicttest, validation$FLOOR)


# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ PERFORMANCE FOR FLOORS OF BUILDING 1 ~~~~~~~~~~
```

```r

# ~~~~~~~~~~~~ KNN ~~~~~~~~~~~~~~
# Performance Accuracy     Kappa
#             0.9402985 0.9159940

# ~~~~~~~~~~~~ Gradient boosting machine ~~~~~~~~~~~~~~
# Performance Accuracy     Kappa
#             0.9514925 0.9314139

# ~~~~~~~~~~~~ Random Forest ~~~~~~~~~~~~~~
# Performance Accuracy     Kappa
#             0.9757      0.9657


# BUILDING 2
try <- Building2[,1:207]
# Remove columns (WAP) where all the values = 0 (WAP was not detected)
uniquelength <- sapply(try,function(x) length(unique(x)))
try <- subset(try, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(try[,1:207], 1, function(x) length(unique(x[!is.na(x)])) != 1)
try <-  try[keep, ]

# Normalize the data
data_norm <- as.data.frame(t(apply(try, 1, function(x) (x - min(x))/(max(x)-min(


# Build knn model
uniquelength <- sapply(Building2,function(x) length(unique(x)))
Building2 <- subset(Building2, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(Building2[,1:207], 1, function(x) length(unique(x[!is.na(x)])) !=
Building2 <-  Building2[keep, ]

Building2[,1:207] <- data_norm

Building2$FLOOR <- factor(Building2$FLOOR)
cols <- c(1:207, 210)
training <- Building2[ , cols]
training$FLOOR <- factor(training$FLOOR)

# Import validation dataset
validation <- subset(testData, BUILDINGID == 1)
testData2 <- subset(testData, BUILDINGID == 1)

```

```r
# Remove columns (WAP) where all the values = 0 (WAP was not detected)
uniquelength <- sapply(validation,function(x) length(unique(x)))
validation <- subset(validation, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(validation[,1:170], 1, function(x) length(unique(x[!is.na(x)])) !=
validation <-  validation[keep, ]

# Drop columns from validation that do not match with traning set
cols_to_keep <- intersect(colnames(training),colnames(validation))
training <- training[,cols_to_keep, drop=FALSE]
validation <- validation[,cols_to_keep, drop=FALSE]
validation <- as.data.frame(t(apply(validation[,1:146], 1, function(x) (x - min(

validation$FLOOR <- testData2$FLOOR
validation$FLOOR <- factor(validation$FLOOR)

set.seed(123)
trainIndex <- createDataPartition(y = training$FLOOR, p = 0.75,
                                  list = FALSE)


#                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~    Training and Test sets  ~~~~~~~

trainSet <- training [trainIndex,]
testSet <- training [-trainIndex,]

#                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~    K-NN    ~~~~~~~~~~~~~~~~~~~~~~~

df <- trainSet

checkMeans <- df
checkMeans[,1:146][checkMeans[,1:146] == 0] <- NA
rowMeans(checkMeans[,1:146], na.rm = T)
# Keep only rows where mean value of detected WAPs is more than 0.5
checkMeans <- subset(checkMeans, rowMeans(checkMeans[,1:146], na.rm = T) > 0.6)
checkMeans[is.na(checkMeans)] <- 0

df <- checkMeans


# set.seed(123)
# ctrl <- trainControl(method="cv",number = 10)
# knnFit <- train((FLOOR ~ .), data = df, method = "knn", trControl = ctrl, tune
#
# #Output of kNN fit
```

```
# knnFit
# #write.csv(knnFit, file = 'knnperformance.csv')
#
# # test the k-NN model
# knnPredict <- predict(knnFit,newdata = testSet)
# postResample(knnPredict, testSet$FLOOR)
#
# # Check results on validation dataset
# # Apply k-NN model to the validation data
# knnPredicttest <- predict(knnFit,newdata = validation)
# postResample(knnPredicttest, validation$FLOOR)

#             ~~~~~~~~~~~~~~~~~~~~~~~~~~~~   Random Forest   ~~~~~~~~~~~~~~~~~~~~

set.seed(123)
ctrl <- trainControl(method="cv", number = 10)

# Random forest
rfFit <- train(FLOOR ~ ., data = trainSet, method = "rf", trControl = ctrl, preP
plot(rfFit$finalModel)
rfPredict <- predict(rfFit, newdata = testSet)
rfCM <- confusionMatrix(rfPredict, testSet$FLOOR)

rfPredicttest <- predict(rfFit, newdata = validation)
postResample(rfPredicttest, validation$FLOOR)

rfCM <-confusionMatrix(rfPredicttest, validation$FLOOR)
rfCM


# #             ~~~~~~~~~~~~~~~~~~~~~~~~~~~~  eXtreme Gradient Boosting  ~~~~~~~
#
# set.seed(123)
# ctrl <- trainControl(method="cv", number = 10)
#
# # Random forest
# GBFit <- train(FLOOR ~ ., data = trainSet, method = "xgbTree", trControl = ctr
# GBPredict <- predict(GBFit, newdata = testSet)
# rfCM <- confusionMatrix(GBPredict, testSet$FLOOR)
#
# GBPredicttest <- predict(GBFit, newdata = validation)
# postResample(GBPredicttest, validation$FLOOR)

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ PERFORMANCE FOR FLOORS OF BUILDING 2 ~~~~~~~~~~

```

```
# ~~~~~~~~~~~~ KNN ~~~~~~~~~~~~~~
# Performance Accuracy     Kappa
#             0.7719870 0.6742064

# ~~~~~~~~~~~~ Gradient boosting machine ~~~~~~~~~~~~~~
# Performance Accuracy     Kappa
#             0.7524430 0.6609854

# ~~~~~~~~~~~~ Random Forest ~~~~~~~~~~~~~~
# Performance Accuracy     Kappa
#             0.8990228 0.8520298



# BUILDING 3
try <- Building3[,1:203]
# Remove columns (WAP) where all the values = 0 (WAP was not detected)
uniquelength <- sapply(try,function(x) length(unique(x)))
try <- subset(try, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(try[,1:203], 1, function(x) length(unique(x[!is.na(x)])) != 1)
try <-  try[keep, ]

# Normalize the data
data_norm <- as.data.frame(t(apply(try, 1, function(x) (x - min(x))/(max(x)-min(


# Build knn model
uniquelength <- sapply(Building3,function(x) length(unique(x)))
Building3 <- subset(Building3, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(Building3[,1:203], 1, function(x) length(unique(x[!is.na(x)])) !=
Building3 <-  Building3[keep, ]

Building3[,1:203] <- data_norm

Building3$FLOOR <- factor(Building3$FLOOR)
cols <- c(1:203, 206)
training <- Building3[ , cols]
training$FLOOR <- factor(training$FLOOR)

# Import validation dataset
validation <- subset(testData, BUILDINGID == 2)
testData3 <- subset(testData, BUILDINGID == 2)

```

```r
# Remove columns (WAP) where all the values = 0 (WAP was not detected)
uniquelength <- sapply(validation,function(x) length(unique(x)))
validation <- subset(validation, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(validation[,1:125], 1, function(x) length(unique(x[!is.na(x)])) !=
validation <-  validation[keep, ]

# Drop columns from validation that do not match with traning set
cols_to_keep <- intersect(colnames(training),colnames(validation))
training <- training[,cols_to_keep, drop=FALSE]
validation <- validation[,cols_to_keep, drop=FALSE]
validation <- as.data.frame(t(apply(validation[,1:104], 1, function(x) (x - min(

validation$FLOOR <- testData3$FLOOR
validation$FLOOR <- factor(validation$FLOOR)

set.seed(123)
trainIndex <- createDataPartition(y = training$FLOOR, p = 0.75,
                                  list = FALSE)


#                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~    Training and Test sets  ~~~~~~~

trainSet <- training [trainIndex,]
testSet <- training [-trainIndex,]

#                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~    K-NN    ~~~~~~~~~~~~~~~~~~~~~~~

df <- trainSet

checkMeans <- df
checkMeans[,1:104][checkMeans[,1:104] == 0] <- NA
rowMeans(checkMeans[,1:104], na.rm = T)
# Keep only rows where mean value of detected WAPs is more than 0.5
checkMeans <- subset(checkMeans, rowMeans(checkMeans[,1:104], na.rm = T) > 0.6)
checkMeans[is.na(checkMeans)] <- 0

df <- checkMeans


# set.seed(123)
# ctrl <- trainControl(method="cv",number = 10)
# knnFit <- train((FLOOR ~ .), data = df, method = "knn", trControl = ctrl, tune
#
# #Output of kNN fit
```

```
# knnFit
# #write.csv(knnFit, file = 'knnperformance.csv')
#
# # test the k-NN model
# knnPredict <- predict(knnFit,newdata = testSet)
# postResample(knnPredict, testSet$FLOOR)
#
# # Check results on validation dataset
# # Apply k-NN model to the validation data
# knnPredicttest <- predict(knnFit,newdata = validation)
# postResample(knnPredicttest, validation$FLOOR)

#             ~~~~~~~~~~~~~~~~~~~~~~~~~~~~   Random Forest   ~~~~~~~~~~~~~~~~~~~~

set.seed(123)
ctrl <- trainControl(method="cv", number = 10)

# Random forest
rfFit <- train(FLOOR ~ ., data = trainSet, method = "rf", trControl = ctrl, preP
# variables importance
varImp(rfFit)

rfPredict <- predict(rfFit, newdata = testSet)
rfCM <- confusionMatrix(rfPredict, testSet$FLOOR)

rfPredicttest <- predict(rfFit, newdata = validation)
postResample(rfPredicttest, validation$FLOOR)

# Confusion matrix of validation test prediction
rfCM <-confusionMatrix(rfPredicttest, validation$FLOOR)
rfCM

# #            ~~~~~~~~~~~~~~~~~~~~~~~~~~~~  eXtreme Gradient Boosting  ~~~~~~~
#
# set.seed(123)
# ctrl <- trainControl(method="cv", number = 10)
#
# # Random forest
# GBFit <- train(FLOOR ~ ., data = trainSet, method = "xgbTree", trControl = ctr
# GBPredict <- predict(GBFit, newdata = testSet)
# rfCM <- confusionMatrix(GBPredict, testSet$FLOOR)
#
# GBPredicttest <- predict(GBFit, newdata = validation)
# postResample(GBPredicttest, validation$FLOOR)
```

```
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ PERFORMANCE FOR FLOORS OF BUILDING 3 ~~~~~~~~~~
# ~~~~~~~~~~~~ KNN ~~~~~~~~~~~~~~
# Performance Accuracy     Kappa
#             0.8395522 0.7829142

# ~~~~~~~~~~~~ Gradient boosting machine ~~~~~~~~~~~~~~
# Performance Accuracy     Kappa
#             0.9402985 0.9186075

# ~~~~~~~~~~~~ Random Forest ~~~~~~~~~~~~~~
# Performance Accuracy     Kappa
#             0.9514925 0.9339376


# OVERALL ACCURACY FOR FLOOR PREDICTION IN THREE BUILDINGS
# n - number of instances in validation dataset
# ACCURACY = (acc1*n1+acc2*n2+acc3*n3)/(n1+n2+n3)
# n1 = 536, n2 = 307, n3 = 268
# ACCURACY = 0.94869487524 = 94.86 %

# PREDICT COORDINATES OF BUILDING 1
#LONGITUDE
try <- Building1[,1:200]
# Remove columns (WAP) where all the values = 0 (WAP was not detected)
uniquelength <- sapply(try,function(x) length(unique(x)))
try <- subset(try, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(try[,1:200], 1, function(x) length(unique(x[!is.na(x)])) != 1)
try <-  try[keep, ]

# Normalize the data
data_norm <- as.data.frame(t(apply(try, 1, function(x) (x - min(x))/(max(x)-min(


# Build knn model
uniquelength <- sapply(Building1,function(x) length(unique(x)))
Building1 <- subset(Building1, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(Building1[,1:200], 1, function(x) length(unique(x[!is.na(x)])) !=
Building1 <-  Building1[keep, ]

Building1[,1:200] <- data_norm

Building1$LONGITUDE <- Building1$LONGITUDE
cols <- c(1:201)
```

```
training <- Building1[ , cols]
training$LONGITUDE <- training$LONGITUDE

# Import validation dataset
validation <- subset(testData, BUILDINGID == 0)
testData1 <- subset(testData, BUILDINGID == 0)

# Remove columns (WAP) where all the values = 0 (WAP was not detected)
uniquelength <- sapply(validation,function(x) length(unique(x)))
validation <- subset(validation, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(validation[,1:183], 1, function(x) length(unique(x[!is.na(x)])) !=
validation <-  validation[keep, ]

# Drop columns from validation that do not match with traning set
cols_to_keep <- intersect(colnames(training),colnames(validation))
training <- training[,cols_to_keep, drop=FALSE]
validation <- validation[,cols_to_keep, drop=FALSE]
validation <- as.data.frame(t(apply(validation[,1:139], 1, function(x) (x - min(

validation$LONGITUDE <- testData1$LONGITUDE
validation$LONGITUDE <- validation$LONGITUDE

validationLONG <- as.data.frame(testData1$LONGITUDE)

set.seed(123)
trainIndex <- createDataPartition(y = training$LONGITUDE, p = 0.75,
                                  list = FALSE)


#                 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~    Training and Test sets  ~~~~~~~

trainSet <- training [trainIndex,]
testSet <- training [-trainIndex,]

#                 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~    K-NN    ~~~~~~~~~~~~~~~~~~~~~~~~

df <- trainSet

checkMeans <- df
checkMeans[,1:139][checkMeans[,1:139] == 0] <- NA
rowMeans(checkMeans[,1:139], na.rm = T)
# Keep only rows where mean value of detected WAPs is more than 0.5
checkMeans <- subset(checkMeans, rowMeans(checkMeans[,1:139], na.rm = T) > 0.6)
checkMeans[is.na(checkMeans)] <- 0
```

```r

df <- checkMeans


set.seed(123)
ctrl <- trainControl(method="cv",number = 10)
knnFit <- train((LONGITUDE ~ .), data = df, method = "knn", trControl = ctrl, tu

#Output of kNN fit
knnFit

# test the k-NN model
knnPredict <- predict(knnFit,newdata = testSet)
postResample(knnPredict, testSet$LONGITUDE)

# Check results on validation dataset
# Apply k-NN model to the validation data
knnPredicttest <- predict(knnFit,newdata = validation)
postResample(knnPredicttest, validation$LONGITUDE)

#              ~~~~~~~~~~~~ KNN ~~~~~~~~~~~~~~
# Performance   RMSE      Rsquared       MAE
#              7.598773   0.939573     6.044276


# Save results in csv file
#write.csv(knnPredicttest, file = "knnPredict.csv")
LongPred1 <- as.data.frame(knnPredicttest)

# LATITUDE
try <- Building1[,1:200]
# Remove columns (WAP) where all the values = 0 (WAP was not detected)
uniquelength <- sapply(try,function(x) length(unique(x)))
try <- subset(try, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(try[,1:200], 1, function(x) length(unique(x[!is.na(x)])) != 1)
try <-  try[keep, ]

# Normalize the data
data_norm <- as.data.frame(t(apply(try, 1, function(x) (x - min(x))/(max(x)-min(


# Build knn model
uniquelength <- sapply(Building1,function(x) length(unique(x)))
Building1 <- subset(Building1, select=uniquelength>1)
```

```
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(Building1[,1:200], 1, function(x) length(unique(x[!is.na(x)])) !=
Building1 <-  Building1[keep, ]

Building1[,1:200] <- data_norm

Building1$LATITUDE <- Building1$LATITUDE
cols <- c(1:200, 202)
training <- Building1[ , cols]
training$LATITUDE <- training$LATITUDE

# Import validation dataset
validation <- subset(testData, BUILDINGID == 0)
testData1 <- subset(testData, BUILDINGID == 0)

# Remove columns (WAP) where all the values = 0 (WAP was not detected)
uniquelength <- sapply(validation,function(x) length(unique(x)))
validation <- subset(validation, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(validation[,1:183], 1, function(x) length(unique(x[!is.na(x)])) !=
validation <-  validation[keep, ]

# Drop columns from validation that do not match with traning set
cols_to_keep <- intersect(colnames(training),colnames(validation))
training <- training[,cols_to_keep, drop=FALSE]
validation <- validation[,cols_to_keep, drop=FALSE]
validation <- as.data.frame(t(apply(validation[,1:139], 1, function(x) (x - min(

validation$LATITUDE <- testData1$LATITUDE
validation$LATITUDE <- validation$LATITUDE

validationLAT <- as.data.frame(testData1$LATITUDE)
validationLONGLAT <- as.data.frame(c(validationLONG, validationLAT))
# Change the column names of validationLONGLAT
colnames(validationLONGLAT)[1] <- "LONGITUDE"
colnames(validationLONGLAT)[2] <- "LATITUDE"


set.seed(123)
trainIndex <- createDataPartition(y = training$LATITUDE, p = 0.75,
                                  list = FALSE)


#                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~    Training and Test sets  ~~~~~~~
```

```
trainSet <- training [trainIndex,]
testSet <- training [-trainIndex,]

#                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~    K-NN    ~~~~~~~~~~~~~~~~~~~~~~~~

df <- trainSet

checkMeans <- df
checkMeans[,1:139][checkMeans[,1:139] == 0] <- NA
rowMeans(checkMeans[,1:139], na.rm = T)
# Keep only rows where mean value of detected WAPs is more than 0.5
checkMeans <- subset(checkMeans, rowMeans(checkMeans[,1:139], na.rm = T) > 0.6)
checkMeans[is.na(checkMeans)] <- 0

df <- checkMeans


set.seed(123)
ctrl <- trainControl(method="cv",number = 10)
knnFit <- train((LATITUDE ~ .), data = df, method = "knn", trControl = ctrl, tun

#Output of kNN fit
knnFit
#write.csv(knnFit, file = 'knnperformance.csv')

# test the k-NN model
knnPredict <- predict(knnFit,newdata = testSet)
postResample(knnPredict, testSet$LATITUDE)

# Check results on validation dataset
# Apply k-NN model to the validation data
knnPredicttest <- predict(knnFit,newdata = validation)
postResample(knnPredicttest, validation$LATITUDE)

#              ~~~~~~~~~~~~ KNN ~~~~~~~~~~~~~~
# Performance    RMSE       Rsquared       MAE
#               7.1845765   0.9601064     5.0841969

# Save results in csv file
#write.csv(knnPredicttest, file = "knnPredictLAT.csv")
LatPred1 <- as.data.frame(knnPredicttest)
LONGLAT_PREDICTIONS <- as.data.frame(c(LongPred1, LatPred1))

# change column names
colnames(LONGLAT_PREDICTIONS)[1] <- 'LONGITUDE'
```

```
colnames(LONGLAT_PREDICTIONS)[2] <- 'LATITUDE'

# Plot real and predicted results
# Training and Validation log in locations
ggplot() +
  geom_point(data = LONGLAT_PREDICTIONS , aes(x = LONGITUDE, y = LATITUDE, colou
  geom_point(data = validationLONGLAT , aes(x = LONGITUDE, y = LATITUDE, colour
  ggtitle("Log In Locations")

# Distribution of distance error (in meters)
Error = sqrt((LONGLAT_PREDICTIONS$LONGITUDE - validationLONGLAT$LONGITUDE)^2 +(L
hist(Error, freq = T, xlab = " Absolute error (m)", col = "red", main = "Error d

# PREDICT COORDINATES OF BUILDING 2
#LONGITUDE
try <- Building2[,1:207]
# Remove columns (WAP) where all the values = 0 (WAP was not detected)
uniquelength <- sapply(try,function(x) length(unique(x)))
try <- subset(try, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(try[,1:207], 1, function(x) length(unique(x[!is.na(x)])) != 1)
try <-  try[keep, ]

# Normalize the data
data_norm <- as.data.frame(t(apply(try, 1, function(x) (x - min(x))/(max(x)-min(


# Build knn model
uniquelength <- sapply(Building2,function(x) length(unique(x)))
Building2 <- subset(Building2, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(Building2[,1:207], 1, function(x) length(unique(x[!is.na(x)])) !=
Building2 <-  Building2[keep, ]

Building2[,1:207] <- data_norm

Building2$LONGITUDE <- Building2$LONGITUDE
cols <- c(1:208)
training <- Building2[ , cols]
training$LONGITUDE <- training$LONGITUDE

# Import validation dataset
validation <- subset(testData, BUILDINGID == 1)
testData2 <- subset(testData, BUILDINGID == 1)
```

```
# Remove columns (WAP) where all the values = 0 (WAP was not detected)
uniquelength <- sapply(validation,function(x) length(unique(x)))
validation <- subset(validation, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(validation[,1:170], 1, function(x) length(unique(x[!is.na(x)])) !=
validation <-  validation[keep, ]

# Drop columns from validation that do not match with traning set
cols_to_keep <- intersect(colnames(training),colnames(validation))
training <- training[,cols_to_keep, drop=FALSE]
validation <- validation[,cols_to_keep, drop=FALSE]
validation <- as.data.frame(t(apply(validation[,1:146], 1, function(x) (x - min(

validation$LONGITUDE <- testData2$LONGITUDE
validation$LONGITUDE <- validation$LONGITUDE

validationLONG <- as.data.frame(testData2$LONGITUDE)

set.seed(123)
trainIndex <- createDataPartition(y = training$LONGITUDE, p = 0.75,
                                  list = FALSE)


#                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~    Training and Test sets  ~~~~~~~

trainSet <- training [trainIndex,]
testSet <- training [-trainIndex,]

#                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~    K-NN    ~~~~~~~~~~~~~~~~~~~~~~~

df <- trainSet

checkMeans <- df
checkMeans[,1:146][checkMeans[,1:146] == 0] <- NA
rowMeans(checkMeans[,1:146], na.rm = T)
# Keep only rows where mean value of detected WAPs is more than 0.5
checkMeans <- subset(checkMeans, rowMeans(checkMeans[,1:146], na.rm = T) > 0.6)
checkMeans[is.na(checkMeans)] <- 0

df <- checkMeans


set.seed(123)
ctrl <- trainControl(method="cv",number = 10)
knnFit <- train((LONGITUDE ~ .), data = df, method = "knn", trControl = ctrl, tu
```

```
#Output of kNN fit
knnFit

# test the k-NN model
knnPredict <- predict(knnFit,newdata = testSet)
postResample(knnPredict, testSet$LONGITUDE)

# Check results on validation dataset
# Apply k-NN model to the validation data
knnPredicttest <- predict(knnFit,newdata = validation)
postResample(knnPredicttest, validation$LONGITUDE)

#               ~~~~~~~~~~~~ KNN ~~~~~~~~~~~~~~
# Performance   RMSE      Rsquared       MAE
#               9.385072  0.965634      7.114257


# Save results in csv file
#write.csv(knnPredicttest, file = "knnPredict.csv")
LongPred2 <- as.data.frame(knnPredicttest)

# LATITUDE
try <- Building2[,1:207]
# Remove columns (WAP) where all the values = 0 (WAP was not detected)
uniquelength <- sapply(try,function(x) length(unique(x)))
try <- subset(try, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(try[,1:207], 1, function(x) length(unique(x[!is.na(x)])) != 1)
try <-  try[keep, ]

# Normalize the data
data_norm <- as.data.frame(t(apply(try, 1, function(x) (x - min(x))/(max(x)-min(


# Build knn model
uniquelength <- sapply(Building2,function(x) length(unique(x)))
Building2 <- subset(Building2, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(Building2[,1:207], 1, function(x) length(unique(x[!is.na(x)])) !=
Building2 <-  Building2[keep, ]

Building2[,1:207] <- data_norm

Building2$LATITUDE <- Building2$LATITUDE
```

```
cols <- c(1:207, 209)
training <- Building2[ , cols]
training$LATITUDE <- training$LATITUDE

# Import validation dataset
validation <- subset(testData, BUILDINGID == 1)
testData2 <- subset(testData, BUILDINGID == 1)

# Remove columns (WAP) where all the values = 0 (WAP was not detected)
uniquelength <- sapply(validation,function(x) length(unique(x)))
validation <- subset(validation, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(validation[,1:170], 1, function(x) length(unique(x[!is.na(x)])) !=
validation <-  validation[keep, ]

# Drop columns from validation that do not match with traning set
cols_to_keep <- intersect(colnames(training),colnames(validation))
training <- training[,cols_to_keep, drop=FALSE]
validation <- validation[,cols_to_keep, drop=FALSE]
validation <- as.data.frame(t(apply(validation[,1:146], 1, function(x) (x - min(

validation$LATITUDE <- testData2$LATITUDE
validation$LATITUDE <- validation$LATITUDE

validationLAT <- as.data.frame(testData2$LATITUDE)
validationLONG <- as.data.frame(testData2$LONGITUDE)
validationLONGLAT <- as.data.frame(c(validationLONG, validationLAT))
# Change the column names of validationLONGLAT
colnames(validationLONGLAT)[1] <- "LONGITUDE"
colnames(validationLONGLAT)[2] <- "LATITUDE"

set.seed(123)
trainIndex <- createDataPartition(y = training$LONGITUDE, p = 0.75,
                                  list = FALSE)


#                 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~    Training and Test sets  ~~~~~~~

trainSet <- training [trainIndex,]
testSet <- training [-trainIndex,]

#                 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~    K-NN    ~~~~~~~~~~~~~~~~~~~~~~~~

df <- trainSet

```

```
checkMeans <- df
checkMeans[,1:146][checkMeans[,1:146] == 0] <- NA
rowMeans(checkMeans[,1:146], na.rm = T)
# Keep only rows where mean value of detected WAPs is more than 0.5
checkMeans <- subset(checkMeans, rowMeans(checkMeans[,1:146], na.rm = T) > 0.6)
checkMeans[is.na(checkMeans)] <- 0

df <- checkMeans


set.seed(123)
ctrl <- trainControl(method="cv",number = 10)
knnFit <- train((LATITUDE ~ .), data = df, method = "knn", trControl = ctrl, tun

#Output of kNN fit
knnFit

# test the k-NN model
knnPredict <- predict(knnFit,newdata = testSet)
postResample(knnPredict, testSet$LATITUDE)

# Check results on validation dataset
# Apply k-NN model to the validation data
knnPredicttest <- predict(knnFit,newdata = validation)
postResample(knnPredicttest, validation$LATITUDE)

#              ~~~~~~~~~~~~ KNN ~~~~~~~~~~~~~~
# Performance    RMSE       Rsquared       MAE
#               11.4652267  0.9069251  8.1506292

# Save results in csv file
#write.csv(knnPredicttest, file = "knnPredictLAT.csv")
LatPred2 <- as.data.frame(knnPredicttest)
LONGLAT_PREDICTIONS <- as.data.frame(c(LongPred2, LatPred2))

# change column names
colnames(LONGLAT_PREDICTIONS)[1] <- 'LONGITUDE'
colnames(LONGLAT_PREDICTIONS)[2] <- 'LATITUDE'

# Plot real and predicted results
# Training and Validation log in locations
ggplot() +
  geom_point(data = LONGLAT_PREDICTIONS , aes(x = LONGITUDE, y = LATITUDE, colou
  geom_point(data = validationLONGLAT , aes(x = LONGITUDE, y = LATITUDE, colour
  ggtitle("Log In Locations")
```

```
# Distribution of distance error (in meters)
Error = sqrt((LONGLAT_PREDICTIONS$LONGITUDE - validationLONGLAT$LONGITUDE)^2 +(L
hist(Error, freq = T, xlab = " Absolute error (m)", col = "red", main = "Error d











# PREDICT COORDINATES OF BUILDING 3
#LONGITUDE
try <- Building3[,1:203]
# Remove columns (WAP) where all the values = 0 (WAP was not detected)
uniquelength <- sapply(try,function(x) length(unique(x)))
try <- subset(try, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(try[,1:203], 1, function(x) length(unique(x[!is.na(x)])) != 1)
try <-  try[keep, ]

# Normalize the data
data_norm <- as.data.frame(t(apply(try, 1, function(x) (x - min(x))/(max(x)-min(


# Build knn model
uniquelength <- sapply(Building3,function(x) length(unique(x)))
Building3 <- subset(Building3, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(Building3[,1:203], 1, function(x) length(unique(x[!is.na(x)])) !=
Building3 <-  Building3[keep, ]

Building3[,1:203] <- data_norm

Building3$LONGITUDE <- Building3$LONGITUDE
cols <- c(1:204)
training <- Building3[ , cols]
training$LONGITUDE <- training$LONGITUDE

# Import validation dataset
```

```r
validation <- subset(testData, BUILDINGID == 2)
testData3 <- subset(testData, BUILDINGID == 2)

# Remove columns (WAP) where all the values = 0 (WAP was not detected)
uniquelength <- sapply(validation,function(x) length(unique(x)))
validation <- subset(validation, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(validation[,1:123], 1, function(x) length(unique(x[!is.na(x)])) !=
validation <-  validation[keep, ]

# Drop columns from validation that do not match with traning set
cols_to_keep <- intersect(colnames(training),colnames(validation))
training <- training[,cols_to_keep, drop=FALSE]
validation <- validation[,cols_to_keep, drop=FALSE]
validation <- as.data.frame(t(apply(validation[,1:104], 1, function(x) (x - min(

validation$LONGITUDE <- testData3$LONGITUDE
validation$LONGITUDE <- validation$LONGITUDE

validationLONG <- as.data.frame(testData3$LONGITUDE)

set.seed(123)
trainIndex <- createDataPartition(y = training$LONGITUDE, p = 0.75,
                                  list = FALSE)


#                 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~    Training and Test sets  ~~~~~~~

trainSet <- training [trainIndex,]
testSet <- training [-trainIndex,]

#                 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~    K-NN    ~~~~~~~~~~~~~~~~~~~~~~~

df <- trainSet

checkMeans <- df
checkMeans[,1:104][checkMeans[,1:104] == 0] <- NA
rowMeans(checkMeans[,1:104], na.rm = T)
# Keep only rows where mean value of detected WAPs is more than 0.5
checkMeans <- subset(checkMeans, rowMeans(checkMeans[,1:104], na.rm = T) > 0.6)
checkMeans[is.na(checkMeans)] <- 0

df <- checkMeans
```

```r
set.seed(123)
ctrl <- trainControl(method="cv",number = 10)
knnFit <- train((LONGITUDE ~ .), data = df, method = "knn", trControl = ctrl, tu

#Output of kNN fit
knnFit

# test the k-NN model
knnPredict <- predict(knnFit,newdata = testSet)
postResample(knnPredict, testSet$LONGITUDE)

# Check results on validation dataset
# Apply k-NN model to the validation data
knnPredicttest <- predict(knnFit,newdata = validation)
postResample(knnPredicttest, validation$LONGITUDE)

#              ~~~~~~~~~~~~ KNN ~~~~~~~~~~~~~~
# Performance   RMSE       Rsquared       MAE
#              12.8063785  0.8358643  9.0960994


# Save results in csv file
#write.csv(knnPredicttest, file = "knnPredict.csv")
LongPred3 <- as.data.frame(knnPredicttest)

# LATITUDE
try <- Building3[,1:203]
# Remove columns (WAP) where all the values = 0 (WAP was not detected)
uniquelength <- sapply(try,function(x) length(unique(x)))
try <- subset(try, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(try[,1:203], 1, function(x) length(unique(x[!is.na(x)])) != 1)
try <-  try[keep, ]

# Normalize the data
data_norm <- as.data.frame(t(apply(try, 1, function(x) (x - min(x))/(max(x)-min(


# Build knn model
uniquelength <- sapply(Building3,function(x) length(unique(x)))
Building3 <- subset(Building3, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(Building3[,1:203], 1, function(x) length(unique(x[!is.na(x)])) !=
Building3 <-  Building3[keep, ]

```

```r
Building3[,1:203] <- data_norm

Building3$LONGITUDE <- Building3$LONGITUDE
cols <- c(1:203, 205)
training <- Building3[ , cols]
training$LATITUDE <- training$LATITUDE

# Import validation dataset
validation <- subset(testData, BUILDINGID == 2)
testData3 <- subset(testData, BUILDINGID == 2)

# Remove columns (WAP) where all the values = 0 (WAP was not detected)
uniquelength <- sapply(validation,function(x) length(unique(x)))
validation <- subset(validation, select=uniquelength>1)
# Remove rows (WAP) where all the values = 0 (WAP was not detected)
keep <- apply(validation[,1:123], 1, function(x) length(unique(x[!is.na(x)])) !=
validation <-  validation[keep, ]

# Drop columns from validation that do not match with traning set
cols_to_keep <- intersect(colnames(training),colnames(validation))
training <- training[,cols_to_keep, drop=FALSE]
validation <- validation[,cols_to_keep, drop=FALSE]
validation <- as.data.frame(t(apply(validation[,1:104], 1, function(x) (x - min(

validation$LATITUDE <- testData3$LATITUDE
validation$LATITUDE <- validation$LATITUDE

validationLAT <- as.data.frame(testData3$LATITUDE)

set.seed(123)
trainIndex <- createDataPartition(y = training$LATITUDE, p = 0.75,
                                  list = FALSE)


#                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~    Training and Test sets  ~~~~~~~

trainSet <- training [trainIndex,]
testSet <- training [-trainIndex,]

#                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~    K-NN    ~~~~~~~~~~~~~~~~~~~~~~~

df <- trainSet

checkMeans <- df
checkMeans[,1:104][checkMeans[,1:104] == 0] <- NA
```

```r
rowMeans(checkMeans[,1:104], na.rm = T)
# Keep only rows where mean value of detected WAPs is more than 0.5
checkMeans <- subset(checkMeans, rowMeans(checkMeans[,1:104], na.rm = T) > 0.6)
checkMeans[is.na(checkMeans)] <- 0

df <- checkMeans


set.seed(123)
ctrl <- trainControl(method="cv",number = 10)
knnFit <- train((LATITUDE ~ .), data = df, method = "knn", trControl = ctrl, tun

#Output of kNN fit
knnFit

# test the k-NN model
knnPredict <- predict(knnFit,newdata = testSet)
postResample(knnPredict, testSet$LATITUDE)

# Check results on validation dataset
# Apply k-NN model to the validation data
knnPredicttest <- predict(knnFit,newdata = validation)
postResample(knnPredicttest, validation$LATITUDE)

#              ~~~~~~~~~~~~ KNN ~~~~~~~~~~~~~~
# Performance    RMSE       Rsquared       MAE
#               9.9643518  0.8955963  7.4651590

# Save results in csv file
#write.csv(knnPredicttest, file = "knnPredictLAT.csv")
LatPred3 <- as.data.frame(knnPredicttest)
LONGLAT_PREDICTIONS <- as.data.frame(c(LongPred3, LatPred3))

validationLONG <- testData3$LONGITUDE
validationLAT <- testData3$LATITUDE

validationLONGLAT <- as.data.frame(c(validationLONG, validationLAT))
colnames(validationLONGLAT)[1] <- 'LONGITUDE'
colnames(validationLONGLAT)[2] <- 'LATITUDE'

# change column names
colnames(LONGLAT_PREDICTIONS)[1] <- 'LONGITUDE'
colnames(LONGLAT_PREDICTIONS)[2] <- 'LATITUDE'

# Plot real and predicted results
```

```r
# Training and Validation log in locations
ggplot() +
  geom_point(data = LONGLAT_PREDICTIONS , aes(x = LONGITUDE, y = LATITUDE, colou
  geom_point(data = testData3 , aes(x = LONGITUDE, y = LATITUDE, colour = "Real 
  ggtitle("Log In Locations")

# Distribution of distance error (in meters)
Error = sqrt((LONGLAT_PREDICTIONS$LONGITUDE - testData3$LONGITUDE)^2 +(LONGLAT_P
hist(Error, freq = T, xlab = " Absolute error (m)", col = "red", main = "Error d
```