

Join GitHub today

Dismiss

GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together.

Sign up

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

```
1 # Indoor locationing via wifi fingerprinting
2 # Armin Talic
3
4
5 # ~~~~~ LIBRARIES ~~~~~
6 library(readr)
7 library(dplyr)
8 library(scatterplot3d)
9 library(ggplot2)
10 library(corrplot)
11 library(caret)
12 library(som)
13 library(data.table)
14
15
16 # Set working directory
17
18 # Import training dataset
```

```

19 trainingData <- read_csv("trainingData.csv")
20
21 # Import validation dataset
22 testData <- read_csv("validationData.csv")
23
24 # ~~~~~ DATA PREPROCESSING ~~~~~
25
26 # Remove columns (WAP) where all the values = 100 (WAP was not detected)
27 # Training dataset
28 unqulength <- sapply(trainingData,function(x) length(unique(x)))
29 trainingData <- subset(trainingData, select=unqulength>1)
30 # Test dataset
31 unqulength <- sapply(testData,function(x) length(unique(x)))
32 testData <- subset(testData, select=unqulength>1)
33
34 # Remove rows (WAP) where all the values = 100 (WAP was not detected)
35 # Training dataset
36 keep <- apply(trainingData[,1:465], 1, function(x) length(unique(x[!is.na(x)])))
37 trainingData[keep, ]
38
39 # # Test dataset
40 keep <- apply(testData[,1:465], 1, function(x) length(unique(x[!is.na(x)]))) != 1
41 testData[keep, ]
42
43 # Converting data types
44 # Training dataset
45 trainingData$FLOOR <- as.factor(trainingData$FLOOR)
46 trainingData$BUILDINGID <- as.factor(trainingData$BUILDINGID)
47 trainingData$RELATIVEPOSITION <- as.factor(trainingData$RELATIVEPOSITION)
48 trainingData$USERID <- as.factor(trainingData$USERID)
49 trainingData$PHONEID <- as.factor(trainingData$PHONEID)
50
51 # Test dataset
52 testData$FLOOR <- as.factor(testData$FLOOR)
53 testData$BUILDINGID <- as.factor(testData$BUILDINGID)
54 testData$PHONEID <- as.factor(testData$PHONEID)
55
56
57 # Change WAP values so that no signal is 0 and highest signal is 104
58 # Training Data
59 trainingData[trainingData == 100] <- -105
60 trainingData[,1:465] <- trainingData[,1:465] + 105
61
62 # Test data
63 testData[testData == 100] <- -105

```

```

64 testData[,1:367] <- testData[,1:367] + 105
65
66 # Check distribution of signal strength
67 # traning data
68 x <- trainingData[,1:465]
69 x <- stack(x)
70
71 x <- x[-grep(0, x$values),]
72 hist(x$values, xlab = "WAP strength", main = "Distribution of WAPs signal stengt
73
74 # test data
75 y <- testData[,1:367]
76 y <- stack(y)
77
78 y <- y[-grep(0, y$values),]
79 hist(y$values, xlab = "WAP strength", main = "Distribution of WAPs signal stengt
80
81 ggplot() +
82   geom_histogram(data = x, aes(values), fill = "red", alpha = 1, binwidth = 5) +
83   geom_histogram(data = y, aes(values), fill = "blue", alpha = 1, binwidth = 5)
84   ggtitle("Distribution of WAPs signal strength (Training and Test sets)") +
85   xlab("WAP strength")
86
87
88 # Check distribution of how many WAPs have signal
89 # TRAINING SET
90 trainingData$count <- rowSums(trainingData[, 1:465] != 0)
91 ggplot(trainingData, aes(count, fill = as.factor(trainingData$BUILDINGID))) +
92   geom_histogram(binwidth = 2)+
93   ggtitle("Number of WAPs detected per building (Training set)") +
94   scale_fill_manual(name="Buildings", values = c("0" = "royalblue2",
95         "1" = "firebrick2",
96         "2" = "springgreen1"),
97         labels=c("Building 1","Building 2", "Building 3"))
98
99 # TEST SET
100 testData$count <- rowSums(testData[, 1:367] != 0)
101 ggplot(testData, aes(count, fill = as.factor(testData$BUILDINGID))) +
102   geom_histogram(binwidth = 2)+
103   ggtitle("Number of WAPs detected per building (Test set)") +
104   scale_fill_manual(name="Buildings", values = c("0" = "royalblue2",
105         "1" = "firebrick2",
106         "2" = "springgreen1"),
107         labels=c("Building 1","Building 2", "Building 3"))
108

```

```

109
110 # Convert Longitude and Latitude values to absolute values
111 # Latitude
112 trainingData$LATITUDE <- trainingData$LATITUDE - min(trainingData$LATITUDE)
113 testData$LATITUDE <- testData$LATITUDE - min(testData$LATITUDE)
114
115
116 # Longitude
117 trainingData$LONGITUDE <- trainingData$LONGITUDE - min(trainingData$LONGITUDE)
118 testData$LONGITUDE <- testData$LONGITUDE - min(testData$LONGITUDE)
119
120 # check maximum values for longitude and latitude in training and test sets
121 max(trainingData$LONGITUDE)
122 max(testData$LONGITUDE)
123 #remove all rows where LONGITUDE values from test set that are higher than in tr
124 testData<-testData[!(testData$LONGITUDE > 390.5194), ]
125
126 #max(trainingData$LATITUDE)
127 #max(testData$LATITUDE)
128
129 # Locations at which users logged in
130 # Red colour is outside the room, black inside
131 p <- ggplot(trainingData, aes(trainingData$LONGITUDE, trainingData$LATITUDE))
132 p + geom_point(colour = as.factor(trainingData$RELATIVEPOSITION)) +
133   xlim(0, 400) +
134   ylim(0, 300) +
135   xlab("Longitude") +
136   ylab("Latitude") +
137   ggtitle ("Locations at which users logged in (Training dataset)")
138
139 # Training and Validation log in locations
140 ggplot() +
141   geom_point(data = trainingData, aes(x = LONGITUDE, y = LATITUDE, colour = "Tra
142   geom_point(data = testData, aes(x = LONGITUDE, y = LATITUDE, colour = "Test da
143   ggtitle("Log In Locations (Training and Test sets)")
144
145 # MODEL TO PREDICT BUILDING
146 # Split Training and Test sets
147
148 training <- trainingData[ ,1:469]
149
150 # Import validation dataset
151 validation <- testData
152
153 # Drop columns from validation that do not match with traning set

```

```

154 cols_to_keep <- intersect(colnames(training),colnames(validation))
155 training <- training[,cols_to_keep, drop=FALSE]
156 validation <- validation[,cols_to_keep, drop=FALSE]
157
158 set.seed(123)
159 trainIndex <- createDataPartition(y = training$BUILDINGID, p = 0.75,
160                                   list = FALSE)
161
162
163 # Training and Test sets
164
165 trainSet <- training [trainIndex,]
166 testSet <- training [-trainIndex,]
167
168 # K-NN
169
170 df <- trainSet
171
172
173 set.seed(123)
174 ctrl <- trainControl(method="cv",number = 5)
175 knnFit <- train((BUILDINGID ~ .), data = df, method = "knn", trControl = ctrl, t
176
177 #Output of kNN fit
178 knnFit
179
180 # test the k-NN model
181 knnPredict <- predict(knnFit,newdata = testSet)
182
183 # confusion matrix to see accuracy value and other parameter values
184 knnCM <-confusionMatrix(knnPredict, testSet$BUILDINGID)
185
186 # Check results on validation dataset
187 # Convert data types in validation dataset
188 validation$BUILDINGID <- as.factor(validation$BUILDINGID)
189
190 # Apply k-NN model to the validation data
191 knnPredictttest <- predict(knnFit,newdata = validation)
192 knnCM <-confusionMatrix(knnPredictttest, validation$BUILDINGID)
193 knnCM
194 # Performance:
195 #Confusion Matrix      0      1      2
196 #              0 536      0      0
197 #              1   0    307      0
198 #              2   0      0    268

```

```

199 # Accuracy 1
200 # Kappa 1
201
202
203 # Create dataset for each building and floor
204 # Create a dataset for each building
205 Building1 <- subset(trainingData, BUILDINGID == 0)
206 Building2 <- subset(trainingData, BUILDINGID == 1)
207 Building3 <- subset(trainingData, BUILDINGID == 2)
208
209 # Remove columns (WAP) where all the values = 100 (WAP was not detected)
210 # Building 1
211 unqulength <- sapply(Building1,function(x) length(unique(x)))
212 Building1 <- subset(Building1, select=unqulength>1)
213
214 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
215 keep <- apply(Building1[,1:200], 1, function(x) length(unique(x[!is.na(x)])) !=
216 Building1[keep, ]
217
218 # Building 2
219 unqulength <- sapply(Building2,function(x) length(unique(x)))
220 Building2 <- subset(Building2, select=unqulength>1)
221
222 # Building 3
223 unqulength <- sapply(Building3,function(x) length(unique(x)))
224 Building3 <- subset(Building3, select=unqulength>1)
225
226 # 3D scatterplot of floors and log in locations
227 # # # Building 1
228 # Building1Floor1 <- subset(Building1, FLOOR == 0)
229 # Building1Floor2 <- subset(Building1, FLOOR == 1)
230 # Building1Floor3 <- subset(Building1, FLOOR == 2)
231 # Building1Floor4 <- subset(Building1, FLOOR == 3)
232
233 # # Building 2
234 # Building2Floor1 <- subset(Building2, FLOOR == 0)
235 # Building2Floor2 <- subset(Building2, FLOOR == 1)
236 # Building2Floor3 <- subset(Building2, FLOOR == 2)
237 # Building2Floor4 <- subset(Building2, FLOOR == 3)
238 #
239 # # Building 3
240 # Building3Floor1 <- subset(Building3, FLOOR == 0)
241 # Building3Floor2 <- subset(Building3, FLOOR == 1)
242 # Building3Floor3 <- subset(Building3, FLOOR == 2)
243 # Building3Floor4 <- subset(Building3, FLOOR == 3)

```

```

244 # Building3Floor5 <- subset(Building3, FLOOR == 4)
245
246 # # Remove columns (WAP) where all the values = 100 (WAP was not detected)
247 # uniqueLength <- sapply(Building1Floor1,function(x) length(unique(x)))
248 # Building1Floor1 <- subset(Building1Floor1, select=uniqueLength>1)
249 #
250 # # Remove rows (WAP) where all the values = 100 (WAP was not detected)
251 # keep <- apply(Building1Floor1[,1:200], 1, function(x) length(unique(x[!is.na(x
252 # Building1Floor1[keep, ]
253
254 # Building 1 inspection
255 unique(Building1$USERID) # 2 different user IDs
256 unique(Building1$PHONEID) # 2 different phone IDs
257
258 # Building 2 inspection
259 # length(unique(Building2$USERID)) # 12 different user IDs
260 # length(unique(Building2$PHONEID)) # 11 different phone IDs
261
262 # Building 3 inspection
263 # length(unique(Building3$USERID)) # 16 different user IDs
264 # length(unique(Building3$PHONEID)) # 15 different phone IDs
265
266 # # Plots
267 # # Building 1
268 # Building1Floor1$z <- 1
269 # Building1Floor2$z <- 2
270 # Building1Floor3$z <- 3
271 # Building1Floor4$z <- 4
272 #
273 # buildplot1 <- rbind(Building1Floor1,Building1Floor2)
274 # buildplot1 <- rbind(buildplot1, Building1Floor3)
275 # buildplot1 <- rbind(buildplot1, Building1Floor4)
276 # buildplot1 <- buildplot1[,521:530]
277 # z <- buildplot1$z
278 # x <- buildplot1$LONGITUDE
279 # y <- buildplot1$LATITUDE
280 # scatterplot3d(x, y, z, pch = 20, angle = 45, color = buildplot1$RELATIVEPOSITI
281
282 # # Building 2
283 # Building2Floor1$z <- 1
284 # Building2Floor2$z <- 2
285 # Building2Floor3$z <- 3
286 # Building2Floor4$z <- 4
287 #
288 # buildplot2 <- rbind(Building2Floor1,Building2Floor2)

```

```

289 # buildplot2 <- rbind(buildplot2, Building2Floor3)
290 # buildplot2 <- rbind(buildplot2, Building2Floor4)
291 # buildplot2 <- buildplot2[,521:530]
292 # c <- buildplot2$z
293 # a <- buildplot2$LONGITUDE
294 # b <- buildplot2$LATITUDE
295 # scatterplot3d(a, b, c, angle = 60, pch = buildplot2$z, color = buildplot2$RELATIVEPOSITION)
296 #
297 # # Building 3
298 # Building3Floor1$z <- 1
299 # Building3Floor2$z <- 2
300 # Building3Floor3$z <- 3
301 # Building3Floor4$z <- 4
302 # Building3Floor5$z <- 5
303 #
304 # buildplot3 <- rbind(Building3Floor1, Building3Floor2)
305 # buildplot3 <- rbind(buildplot3, Building3Floor3)
306 # buildplot3 <- rbind(buildplot3, Building3Floor4)
307 # buildplot3 <- rbind(buildplot3, Building3Floor5)
308 # buildplot3 <- buildplot3[,521:530]
309 # c <- buildplot3$z
310 # a <- buildplot3$LONGITUDE
311 # b <- buildplot3$LATITUDE
312 # scatterplot3d(a, b, c, angle = 20, pch = buildplot3$z, color = buildplot3$RELATIVEPOSITION)
313
314
315 # Check where is the highest signal strength
316 which(Building3[,1:119] == 105)
317 which(Building3[,1:119] == 105, arr.ind=TRUE)
318
319 # ~~~~~Signal greater than 60 ~~~~~
320 abc <- which(apply(trainingData[, 1:465], 1, function(x) length(which(x > 60))))
321 GoodSignal <- trainingData[abc, ]
322
323 gs <- ggplot(GoodSignal, aes(GoodSignal$LONGITUDE, GoodSignal$LATITUDE))
324 gs + geom_point(colour = as.factor(GoodSignal$RELATIVEPOSITION)) +
325   ggtitle("Log in locations where WAP signal was high") +
326   xlab("Longitude") +
327   ylab("Latitude")
328
329
330 # Remove columns (WAP) where all the values = 0 (WAP was not detected)
331 unqulength <- sapply(GoodSignal, function(x) length(unique(x)))
332 GoodSignal <- subset(GoodSignal, select=unqulength>1)
333

```


[illegible]

```

379 # # ~~~~~ Training and Test sets ~~~~~
380 #
381 # trainSet <- training [trainIndex,]
382 # testSet <- training [-trainIndex,]
383 #
384 # # ~~~~~ K-NN ~~~~~
385 #
386 # df <- trainSet
387 #
388 #
389 # set.seed(123)
390 # ctrl <- trainControl(method="cv",number = 10)
391 # knnFit <- train((FLOOR ~ .), data = df, method = "knn", trControl = ctrl, tune
392 #
393 # #Output of kNN fit
394 # knnFit
395 # #write.csv(knnFit, file = 'knnperformance.csv')
396 #
397 # # test the k-NN model
398 # knnPredict <- predict(knnFit,newdata = testSet)
399 # postResample(knnPredict, testSet$FLOOR)
400 #
401 # # Check results on validation dataset
402 # # Import validation dataset
403 # validation <- testData
404 #
405 # # Drop columns from validation that do not match with training set
406 # cols_to_keep <- intersect(colnames(training),colnames(validation))
407 # validation <- validation[,cols_to_keep, drop=FALSE]
408 #
409 # # Apply k-NN model to the validation data
410 # knnPredicttest <- predict(knnFit,newdata = validation)
411 # postResample(knnPredicttest, validation$FLOOR)
412 #
413 # # Result Accuracy Kappa
414 # # ~~~~~ 0.5094509 0.3861611
415 #
416 # # ~~~~~ Random Forest ~~~~~
417 #
418 # set.seed(123)
419 # ctrl <- trainControl(method="cv", number = 10)
420 #
421 # # Random forest
422 # rffit <- train(FLOOR ~ ., data = trainSet, method = "rf", trControl = ctrl, pr
423 # rfpredict <- predict(rffit, newdata = testSet)

```

```

424 # rfCM <- confusionMatrix(rfPredict, testSet$FLOOR)
425 #
426 # rfPredictttest <- predict(rfFit, newdata = validation)
427 # postResample(knnPredictttest, validation$FLOOR)
428 # # Performance Accuracy      Kappa
429 # #           0.5112511 0.3883641
430
431
432 # THIS MODEL ALSO HAS LOW PERFORMACE
433 # # ~~~~~ Model with Normalized rows (mean = 0, variance =
434 # som_row <- normalize(Building1[,1:200], byrow=TRUE)
435 # try <- as.data.frame(som_row)
436 # try$FLOOR <- Building1$FLOOR
437 # Building1[1:200] <- try[1:200]
438 #
439 # Building1$FLOOR <- factor(Building1$FLOOR)
440 # cols <- c(1:200, 203)
441 # training <- Building1[ , cols]
442 # training$FLOOR <- factor(training$FLOOR)
443 #
444 # set.seed(123)
445 # trainIndex <- createDataPartition(y = training$FLOOR, p = 0.75,
446 #                                   list = FALSE)
447 #
448 #
449 # #           ~~~~~ Training and Test sets ~~~~~
450 #
451 # trainSet <- training [trainIndex,]
452 # testSet <- training [-trainIndex,]
453 #
454 # #           ~~~~~ K-NN ~~~~~
455 #
456 # df <- trainSet
457 #
458 #
459 # set.seed(123)
460 # ctrl <- trainControl(method="cv",number = 10)
461 # knnFit <- train((FLOOR ~ .), data = df, method = "knn", trControl = ctrl, tune
462 #
463 # #Output of kNN fit
464 # knnFit
465 # #write.csv(knnFit, file = 'knnperformance.csv')
466 #
467 # # test the k-NN model
468 # knnPredict <- predict(knnFit,newdata = testSet)

```

```

469 # postResample(knnPredict, testSet$FLOOR)
470 #
471 # # Check results on validation dataset
472 # # Import validation dataset
473 # validation <- testData
474 #
475 # # Drop columns from validation that do not match with training set
476 # cols_to_keep <- intersect(colnames(training),colnames(validation))
477 # validation <- validation[,cols_to_keep, drop=FALSE]
478 #
479 # # Apply k-NN model to the validation data
480 # knnPredicttest <- predict(knnFit,newdata = validation)
481 # postResample(knnPredicttest, validation$FLOOR)
482 #
483 # # Performance Accuracy      Kappa
484 # #           0.5724572 0.4031253
485
486
487 # ONCE THE KNN MODEL IS BUILT WITH NORMALIZED DATA, RANDOM FOREST AND GRADIENT B
488 # ARE USED WITH THE NORMALIZED DATA
489 # #           ~~~~~ Random Forest ~~~~~
490 #
491 # set.seed(123)
492 # ctrl <- trainControl(method="cv", number = 10)
493 #
494 # # Random forest
495 # rfFit <- train(FLOOR ~ ., data = trainSet, method = "rf", trControl = ctrl, pr
496 # rfPredict <- predict(rfFit, newdata = testSet)
497 # rfCM <- confusionMatrix(rfPredict, testSet$FLOOR)
498 #
499 # rfPredicttest <- predict(rfFit, newdata = validation)
500 # postResample(rfPredicttest, validation$FLOOR)
501 #
502 # rfCM <- confusionMatrix(rfPredicttest, validation$FLOOR)
503 # rfCM
504 #
505 # # Performance Accuracy      Kappa
506 # #           0.4050405 0.2021011
507 #
508 # #           ~~~~~ eXtreme Gradient Boosting ~~~~~
509 #
510 # set.seed(123)
511 # ctrl <- trainControl(method="cv", number = 10)
512 #
513 # # Random forest

```

```

514 # GBFit <- train(FLOOR ~ ., data = trainSet, method = "xgbTree", trControl = ctr
515 # GBPredict <- predict(GBFit, newdata = testSet)
516 # rfCM <- confusionMatrix(GBPredict, testSet$FLOOR)
517 #
518 # GBPredicttest <- predict(GBFit, newdata = validation)
519 # postResample(GBPredicttest, validation$FLOOR)
520 # # Performance Accuracy      Kappa
521 # #           0.5499550 0.3281139
522
523
524 # Model with scaled data (0 to 1)
525 # BUILDING 1
526 try <- Building1[,1:200]
527 # Remove columns (WAP) where all the values = 0 (WAP was not detected)
528 unqulength <- sapply(try,function(x) length(unique(x)))
529 try <- subset(try, select=unqulength>1)
530 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
531 keep <- apply(try[,1:200], 1, function(x) length(unique(x[!is.na(x)])) != 1)
532 try <- try[keep, ]
533
534 # Normalize the data
535 data_norm <- as.data.frame(t(apply(try, 1, function(x) (x - min(x))/(max(x)-min(
536
537
538 # Build knn model
539 unqulength <- sapply(Building1,function(x) length(unique(x)))
540 Building1 <- subset(Building1, select=unqulength>1)
541 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
542 keep <- apply(Building1[,1:200], 1, function(x) length(unique(x[!is.na(x)])) !=
543 Building1 <- Building1[keep, ]
544
545 Building1[,1:200] <- data_norm
546
547 Building1$FLOOR <- factor(Building1$FLOOR)
548 cols <- c(1:200, 203)
549 training <- Building1[, cols]
550 training$FLOOR <- factor(training$FLOOR)
551
552 # Import validation dataset
553 validation <- subset(testData, BUILDINGID == 0)
554 testData1 <- subset(testData, BUILDINGID == 0)
555
556 # Remove columns (WAP) where all the values = 0 (WAP was not detected)
557 unqulength <- sapply(validation,function(x) length(unique(x)))
558 validation <- subset(validation, select=unqulength>1)

```

```

559 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
560 keep <- apply(validation[,1:183], 1, function(x) length(unique(x[!is.na(x)])) !=
561 validation <- validation[keep, ]
562
563 # Drop columns from validation that do not match with training set
564 cols_to_keep <- intersect(colnames(training), colnames(validation))
565 training <- training[,cols_to_keep, drop=FALSE]
566 validation <- validation[,cols_to_keep, drop=FALSE]
567 validation <- as.data.frame(t(apply(validation[,1:139], 1, function(x) (x - min(
568
569 validation$FLOOR <- testData1$FLOOR
570 validation$FLOOR <- factor(validation$FLOOR)
571
572 set.seed(123)
573 trainIndex <- createDataPartition(y = training$FLOOR, p = 0.75,
574                                   list = FALSE)
575
576
577 # ~~~~~ Training and Test sets ~~~~~
578
579 trainSet <- training [trainIndex,]
580 testSet <- training [-trainIndex,]
581
582 # ~~~~~ K-NN ~~~~~
583
584 df <- trainSet
585
586 checkMeans <- df
587 checkMeans[,1:139][checkMeans[,1:139] == 0] <- NA
588 rowMeans(checkMeans[,1:139], na.rm = T)
589 # Keep only rows where mean value of detected WAPs is more than 0.5
590 checkMeans <- subset(checkMeans, rowMeans(checkMeans[,1:139], na.rm = T) > 0.6)
591 checkMeans[is.na(checkMeans)] <- 0
592
593 df <- checkMeans
594
595
596 # set.seed(123)
597 # ctrl <- trainControl(method="cv", number = 10)
598 # knnFit <- train((FLOOR ~ .), data = df, method = "knn", trControl = ctrl, tune
599 #
600 # #Output of knn fit
601 # knnFit
602 # #write.csv(knnFit, file = 'knnperformance.csv')
603 #

```

```

604 # # test the k-NN model
605 # knnPredict <- predict(knnFit,newdata = testSet)
606 # postResample(knnPredict, testSet$FLOOR)
607 #
608 # # Check results on validation dataset
609 # # Apply k-NN model to the validation data
610 # knnPredicttest <- predict(knnFit,newdata = validation)
611 # postResample(knnPredicttest, validation$FLOOR)
612 #
613 # knnCM <- confusionMatrix(knnPredicttest, validation$FLOOR)
614 # knnCM
615
616 # ~~~~~ Random Forest ~~~~~
617
618 set.seed(123)
619 ctrl <- trainControl(method="cv", number = 5)
620
621 # Random forest
622 rfFit <- train(FLOOR ~ ., data = trainSet, method = "rf", trControl = ctrl, prep
623 rfPredict <- predict(rfFit, newdata = testSet)
624 rfCM <- confusionMatrix(rfPredict, testSet$FLOOR)
625 plot(rfFit$finalModel)
626
627 rfPredicttest <- predict(rfFit, newdata = validation)
628 postResample(rfPredicttest, validation$FLOOR)
629
630 rfCM <- confusionMatrix(rfPredicttest, validation$FLOOR)
631 rfCM
632
633
634 # # ~~~~~ eXtreme Gradient Boosting ~~~~~
635 #
636 # set.seed(123)
637 # ctrl <- trainControl(method="cv", number = 10)
638 #
639 # # Random forest
640 # GBFit <- train(FLOOR ~ ., data = trainSet, method = "xgbTree", trControl = ctr
641 # GBPredict <- predict(GBFit, newdata = testSet)
642 # rfCM <- confusionMatrix(GBPredict, testSet$FLOOR)
643 #
644 # GBPredicttest <- predict(GBFit, newdata = validation)
645 # postResample(GBPredicttest, validation$FLOOR)
646
647
648 # ~~~~~ PERFORMANCE FOR FLOORS OF BUILDING 1 ~~~~~

```

```

649
650 # ~~~~~ KNN ~~~~~
651 # Performance Accuracy      Kappa
652 #           0.9402985 0.9159940
653
654 # ~~~~~ Gradient boosting machine ~~~~~
655 # Performance Accuracy      Kappa
656 #           0.9514925 0.9314139
657
658 # ~~~~~ Random Forest ~~~~~
659 # Performance Accuracy      Kappa
660 #           0.9757      0.9657
661
662
663 # BUILDING 2
664 try <- Building2[,1:207]
665 # Remove columns (WAP) where all the values = 0 (WAP was not detected)
666 unqulength <- sapply(try,function(x) length(unique(x)))
667 try <- subset(try, select=unqulength>1)
668 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
669 keep <- apply(try[,1:207], 1, function(x) length(unique(x[!is.na(x)])) != 1)
670 try <- try[keep, ]
671
672 # Normalize the data
673 data_norm <- as.data.frame(t(apply(try, 1, function(x) (x - min(x))/(max(x)-min(
674
675
676 # Build knn model
677 unqulength <- sapply(Building2,function(x) length(unique(x)))
678 Building2 <- subset(Building2, select=unqulength>1)
679 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
680 keep <- apply(Building2[,1:207], 1, function(x) length(unique(x[!is.na(x)])) !=
681 Building2 <- Building2[keep, ]
682
683 Building2[,1:207] <- data_norm
684
685 Building2$FLOOR <- factor(Building2$FLOOR)
686 cols <- c(1:207, 210)
687 training <- Building2[, cols]
688 training$FLOOR <- factor(training$FLOOR)
689
690 # Import validation dataset
691 validation <- subset(testData, BUILDINGID == 1)
692 testData2 <- subset(testData, BUILDINGID == 1)
693

```



```

694 # Remove columns (WAP) where all the values = 0 (WAP was not detected)
695 unquellength <- sapply(validation,function(x) length(unique(x)))
696 validation <- subset(validation, select=unquellength>1)
697 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
698 keep <- apply(validation[,1:170], 1, function(x) length(unique(x[!is.na(x)])) !=
699 validation <- validation[keep, ]
700
701 # Drop columns from validation that do not match with training set
702 cols_to_keep <- intersect(colnames(training),colnames(validation))
703 training <- training[,cols_to_keep, drop=FALSE]
704 validation <- validation[,cols_to_keep, drop=FALSE]
705 validation <- as.data.frame(t(apply(validation[,1:146], 1, function(x) (x - min(
706
707 validation$FLOOR <- testData2$FLOOR
708 validation$FLOOR <- factor(validation$FLOOR)
709
710 set.seed(123)
711 trainIndex <- createDataPartition(y = training$FLOOR, p = 0.75,
712                                   list = FALSE)
713
714
715 # ~~~~~ Training and Test sets ~~~~~
716
717 trainSet <- training [trainIndex,]
718 testSet <- training [-trainIndex,]
719
720 # ~~~~~ K-NN ~~~~~
721
722 df <- trainSet
723
724 checkMeans <- df
725 checkMeans[,1:146][checkMeans[,1:146] == 0] <- NA
726 rowMeans(checkMeans[,1:146], na.rm = T)
727 # Keep only rows where mean value of detected WAPs is more than 0.5
728 checkMeans <- subset(checkMeans, rowMeans(checkMeans[,1:146], na.rm = T) > 0.6)
729 checkMeans[is.na(checkMeans)] <- 0
730
731 df <- checkMeans
732
733
734 # set.seed(123)
735 # ctrl <- trainControl(method="cv",number = 10)
736 # knnFit <- train((FLOOR ~ .), data = df, method = "knn", trControl = ctrl, tune
737 #
738 # #Output of knn fit

```

```

739 # knnFit
740 # #write.csv(knnFit, file = 'knnperformance.csv')
741 #
742 # # test the k-NN model
743 # knnPredict <- predict(knnFit,newdata = testSet)
744 # postResample(knnPredict, testSet$FLOOR)
745 #
746 # # Check results on validation dataset
747 # # Apply k-NN model to the validation data
748 # knnPredicttest <- predict(knnFit,newdata = validation)
749 # postResample(knnPredicttest, validation$FLOOR)
750
751 # ~~~~~ Random Forest ~~~~~
752
753 set.seed(123)
754 ctrl <- trainControl(method="cv", number = 10)
755
756 # Random forest
757 rfFit <- train(FLOOR ~ ., data = trainSet, method = "rf", trControl = ctrl, prep
758 plot(rfFit$finalModel)
759 rfPredict <- predict(rfFit, newdata = testSet)
760 rfCM <- confusionMatrix(rfPredict, testSet$FLOOR)
761
762 rfPredicttest <- predict(rfFit, newdata = validation)
763 postResample(rfPredicttest, validation$FLOOR)
764
765 rfCM <- confusionMatrix(rfPredicttest, validation$FLOOR)
766 rfCM
767
768
769 # # ~~~~~ eXtreme Gradient Boosting ~~~~~
770 #
771 # set.seed(123)
772 # ctrl <- trainControl(method="cv", number = 10)
773 #
774 # # Random forest
775 # GBFit <- train(FLOOR ~ ., data = trainSet, method = "xgbTree", trControl = ctr
776 # GBPredict <- predict(GBFit, newdata = testSet)
777 # rfCM <- confusionMatrix(GBPredict, testSet$FLOOR)
778 #
779 # GBPredicttest <- predict(GBFit, newdata = validation)
780 # postResample(GBPredicttest, validation$FLOOR)
781
782 # ~~~~~ PERFORMANCE FOR FLOORS OF BUILDING 2 ~~~~~
783

```

```

784 # ~~~~~ KNN ~~~~~
785 # Performance Accuracy      Kappa
786 #           0.7719870 0.6742064
787
788 # ~~~~~ Gradient boosting machine ~~~~~
789 # Performance Accuracy      Kappa
790 #           0.7524430 0.6609854
791
792 # ~~~~~ Random Forest ~~~~~
793 # Performance Accuracy      Kappa
794 #           0.8990228 0.8520298
795
796
797
798 # BUILDING 3
799 try <- Building3[,1:203]
800 # Remove columns (WAP) where all the values = 0 (WAP was not detected)
801 unqulength <- sapply(try,function(x) length(unique(x)))
802 try <- subset(try, select=unqulength>1)
803 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
804 keep <- apply(try[,1:203], 1, function(x) length(unique(x[!is.na(x)])) != 1)
805 try <- try[keep, ]
806
807 # Normalize the data
808 data_norm <- as.data.frame(t(apply(try, 1, function(x) (x - min(x))/(max(x)-min(
809
810
811 # Build knn model
812 unqulength <- sapply(Building3,function(x) length(unique(x)))
813 Building3 <- subset(Building3, select=unqulength>1)
814 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
815 keep <- apply(Building3[,1:203], 1, function(x) length(unique(x[!is.na(x)])) !=
816 Building3 <- Building3[keep, ]
817
818 Building3[,1:203] <- data_norm
819
820 Building3$FLOOR <- factor(Building3$FLOOR)
821 cols <- c(1:203, 206)
822 training <- Building3[ , cols]
823 training$FLOOR <- factor(training$FLOOR)
824
825 # Import validation dataset
826 validation <- subset(testData, BUILDINGID == 2)
827 testData3 <- subset(testData, BUILDINGID == 2)
828

```

```

829 # Remove columns (WAP) where all the values = 0 (WAP was not detected)
830 unqulength <- sapply(validation,function(x) length(unique(x)))
831 validation <- subset(validation, select=unqulength>1)
832 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
833 keep <- apply(validation[,1:125], 1, function(x) length(unique(x[!is.na(x)])) !=
834 validation <- validation[keep, ]
835
836 # Drop columns from validation that do not match with training set
837 cols_to_keep <- intersect(colnames(training),colnames(validation))
838 training <- training[,cols_to_keep, drop=FALSE]
839 validation <- validation[,cols_to_keep, drop=FALSE]
840 validation <- as.data.frame(t(apply(validation[,1:104], 1, function(x) (x - min(
841
842 validation$FLOOR <- testData3$FLOOR
843 validation$FLOOR <- factor(validation$FLOOR)
844
845 set.seed(123)
846 trainIndex <- createDataPartition(y = training$FLOOR, p = 0.75,
847 list = FALSE)
848
849
850 # ~~~~~ Training and Test sets ~~~~~
851
852 trainSet <- training [trainIndex,]
853 testSet <- training [-trainIndex,]
854
855 # ~~~~~ K-NN ~~~~~
856
857 df <- trainSet
858
859 checkMeans <- df
860 checkMeans[,1:104][checkMeans[,1:104] == 0] <- NA
861 rowMeans(checkMeans[,1:104], na.rm = T)
862 # Keep only rows where mean value of detected WAPs is more than 0.5
863 checkMeans <- subset(checkMeans, rowMeans(checkMeans[,1:104], na.rm = T) > 0.6)
864 checkMeans[is.na(checkMeans)] <- 0
865
866 df <- checkMeans
867
868
869 # set.seed(123)
870 # ctrl <- trainControl(method="cv",number = 10)
871 # knnFit <- train((FLOOR ~ .), data = df, method = "knn", trControl = ctrl, tune
872 #
873 # #Output of knn fit

```

```

874 # knnFit
875 # #write.csv(knnFit, file = 'knnperformance.csv')
876 #
877 # # test the k-NN model
878 # knnPredict <- predict(knnFit,newdata = testSet)
879 # postResample(knnPredict, testSet$FLOOR)
880 #
881 # # Check results on validation dataset
882 # # Apply k-NN model to the validation data
883 # knnPredicttest <- predict(knnFit,newdata = validation)
884 # postResample(knnPredicttest, validation$FLOOR)
885
886 # ~~~~~ Random Forest ~~~~~
887
888 set.seed(123)
889 ctrl <- trainControl(method="cv", number = 10)
890
891 # Random forest
892 rfFit <- train(FLOOR ~ ., data = trainSet, method = "rf", trControl = ctrl, preP
893 # variables importance
894 varImp(rfFit)
895
896 rfPredict <- predict(rfFit, newdata = testSet)
897 rfCM <- confusionMatrix(rfPredict, testSet$FLOOR)
898
899 rfPredicttest <- predict(rfFit, newdata = validation)
900 postResample(rfPredicttest, validation$FLOOR)
901
902 # Confusion matrix of validation test prediction
903 rfCM <- confusionMatrix(rfPredicttest, validation$FLOOR)
904 rfCM
905
906 # # ~~~~~ eXtreme Gradient Boosting ~~~~~
907 #
908 # set.seed(123)
909 # ctrl <- trainControl(method="cv", number = 10)
910 #
911 # # Random forest
912 # GBFit <- train(FLOOR ~ ., data = trainSet, method = "xgbTree", trControl = ctr
913 # GBPredict <- predict(GBFit, newdata = testSet)
914 # rfCM <- confusionMatrix(GBPredict, testSet$FLOOR)
915 #
916 # GBPredicttest <- predict(GBFit, newdata = validation)
917 # postResample(GBPredicttest, validation$FLOOR)
918

```

```

919 # ~~~~~ PERFORMANCE FOR FLOORS OF BUILDING 3 ~~~~~
920 # ~~~~~ KNN ~~~~~
921 # Performance Accuracy      Kappa
922 #           0.8395522 0.7829142
923
924 # ~~~~~ Gradient boosting machine ~~~~~
925 # Performance Accuracy      Kappa
926 #           0.9402985 0.9186075
927
928 # ~~~~~ Random Forest ~~~~~
929 # Performance Accuracy      Kappa
930 #           0.9514925 0.9339376
931
932
933 # OVERALL ACCURACY FOR FLOOR PREDICTION IN THREE BUILDINGS
934 # n - number of instances in validation dataset
935 # ACCURACY = (acc1*n1+acc2*n2+acc3*n3)/(n1+n2+n3)
936 # n1 = 536, n2 = 307, n3 = 268
937 # ACCURACY = 0.94869487524 = 94.86 %
938
939 # PREDICT COORDINATES OF BUILDING 1
940 #LONGITUDE
941 try <- Building1[,1:200]
942 # Remove columns (WAP) where all the values = 0 (WAP was not detected)
943 uniquelength <- sapply(try,function(x) length(unique(x)))
944 try <- subset(try, select=uniquelength>1)
945 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
946 keep <- apply(try[,1:200], 1, function(x) length(unique(x[!is.na(x)])) != 1)
947 try <- try[keep, ]
948
949 # Normalize the data
950 data_norm <- as.data.frame(t(apply(try, 1, function(x) (x - min(x))/(max(x)-min(
951
952
953 # Build knn model
954 uniquelength <- sapply(Building1,function(x) length(unique(x)))
955 Building1 <- subset(Building1, select=uniquelength>1)
956 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
957 keep <- apply(Building1[,1:200], 1, function(x) length(unique(x[!is.na(x)])) !=
958 Building1 <- Building1[keep, ]
959
960 Building1[,1:200] <- data_norm
961
962 Building1$LONGITUDE <- Building1$LONGITUDE
963 cols <- c(1:201)

```

```

964 training <- Building1[ , cols]
965 training$LONGITUDE <- training$LONGITUDE
966
967 # Import validation dataset
968 validation <- subset(testData, BUILDINGID == 0)
969 testData1 <- subset(testData, BUILDINGID == 0)
970
971 # Remove columns (WAP) where all the values = 0 (WAP was not detected)
972 uniquelength <- sapply(validation,function(x) length(unique(x)))
973 validation <- subset(validation, select=uniquelength>1)
974 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
975 keep <- apply(validation[,1:183], 1, function(x) length(unique(x[!is.na(x)])) !=
976 validation <- validation[keep, ]
977
978 # Drop columns from validation that do not match with training set
979 cols_to_keep <- intersect(colnames(training),colnames(validation))
980 training <- training[,cols_to_keep, drop=FALSE]
981 validation <- validation[,cols_to_keep, drop=FALSE]
982 validation <- as.data.frame(t(apply(validation[,1:139], 1, function(x) (x - min(
983
984 validation$LONGITUDE <- testData1$LONGITUDE
985 validation$LONGITUDE <- validation$LONGITUDE
986
987 validationLONG <- as.data.frame(testData1$LONGITUDE)
988
989 set.seed(123)
990 trainIndex <- createDataPartition(y = training$LONGITUDE, p = 0.75,
991                                   list = FALSE)
992
993
994 # ~~~~~ Training and Test sets ~~~~~
995
996 trainSet <- training [trainIndex,]
997 testSet <- training [-trainIndex,]
998
999 # ~~~~~ K-NN ~~~~~
1000
1001 df <- trainSet
1002
1003 checkMeans <- df
1004 checkMeans[,1:139][checkMeans[,1:139] == 0] <- NA
1005 rowMeans(checkMeans[,1:139], na.rm = T)
1006 # Keep only rows where mean value of detected WAPs is more than 0.5
1007 checkMeans <- subset(checkMeans, rowMeans(checkMeans[,1:139], na.rm = T) > 0.6)
1008 checkMeans[is.na(checkMeans)] <- 0

```

```

1009
1010 df <- checkMeans
1011
1012
1013 set.seed(123)
1014 ctrl <- trainControl(method="cv",number = 10)
1015 knnFit <- train((LONGITUDE ~ .), data = df, method = "knn", trControl = ctrl, tu
1016
1017 #Output of knn fit
1018 knnFit
1019
1020 # test the k-NN model
1021 knnPredict <- predict(knnFit,newdata = testSet)
1022 postResample(knnPredict, testSet$LONGITUDE)
1023
1024 # Check results on validation dataset
1025 # Apply k-NN model to the validation data
1026 knnPredictttest <- predict(knnFit,newdata = validation)
1027 postResample(knnPredictttest, validation$LONGITUDE)
1028
1029 # ~~~~~ KNN ~~~~~
1030 # Performance RMSE Rsquared MAE
1031 # 7.598773 0.939573 6.044276
1032
1033
1034 # Save results in csv file
1035 #write.csv(knnPredictttest, file = "knnPredict.csv")
1036 LongPred1 <- as.data.frame(knnPredictttest)
1037
1038 # LATITUDE
1039 try <- Building1[,1:200]
1040 # Remove columns (WAP) where all the values = 0 (WAP was not detected)
1041 uniquelength <- sapply(try,function(x) length(unique(x)))
1042 try <- subset(try, select=uniquelength>1)
1043 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
1044 keep <- apply(try[,1:200], 1, function(x) length(unique(x[!is.na(x)])) != 1)
1045 try <- try[keep, ]
1046
1047 # Normalize the data
1048 data_norm <- as.data.frame(t(apply(try, 1, function(x) (x - min(x))/(max(x)-min(
1049
1050
1051 # Build knn model
1052 uniquelength <- sapply(Building1,function(x) length(unique(x)))
1053 Building1 <- subset(Building1, select=uniquelength>1)

```



```

1054 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
1055 keep <- apply(Building1[,1:200], 1, function(x) length(unique(x[!is.na(x)])) !=
1056 Building1 <- Building1[keep, ]
1057
1058 Building1[,1:200] <- data_norm
1059
1060 Building1$LATITUDE <- Building1$LATITUDE
1061 cols <- c(1:200, 202)
1062 training <- Building1[, cols]
1063 training$LATITUDE <- training$LATITUDE
1064
1065 # Import validation dataset
1066 validation <- subset(testData, BUILDINGID == 0)
1067 testData1 <- subset(testData, BUILDINGID == 0)
1068
1069 # Remove columns (WAP) where all the values = 0 (WAP was not detected)
1070 uniqueLength <- sapply(validation,function(x) length(unique(x)))
1071 validation <- subset(validation, select=uniqueLength>1)
1072 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
1073 keep <- apply(validation[,1:183], 1, function(x) length(unique(x[!is.na(x)])) !=
1074 validation <- validation[keep, ]
1075
1076 # Drop columns from validation that do not match with training set
1077 cols_to_keep <- intersect(colnames(training),colnames(validation))
1078 training <- training[,cols_to_keep, drop=FALSE]
1079 validation <- validation[,cols_to_keep, drop=FALSE]
1080 validation <- as.data.frame(t(apply(validation[,1:139], 1, function(x) (x - min(
1081
1082 validation$LATITUDE <- testData1$LATITUDE
1083 validation$LATITUDE <- validation$LATITUDE
1084
1085 validationLAT <- as.data.frame(testData1$LATITUDE)
1086 validationLONGLAT <- as.data.frame(c(validationLONGL, validationLAT))
1087 # Change the column names of validationLONGLAT
1088 colnames(validationLONGLAT)[1] <- "LONGITUDE"
1089 colnames(validationLONGLAT)[2] <- "LATITUDE"
1090
1091
1092 set.seed(123)
1093 trainIndex <- createDataPartition(y = training$LATITUDE, p = 0.75,
1094                                   list = FALSE)
1095
1096
1097 # ~~~~~ Training and Test sets ~~~~~
1098

```

```

1099 trainSet <- training [trainIndex,]
1100 testSet <- training [-trainIndex,]
1101
1102 # ~~~~~ K-NN ~~~~~
1103
1104 df <- trainSet
1105
1106 checkMeans <- df
1107 checkMeans[,1:139][checkMeans[,1:139] == 0] <- NA
1108 rowMeans(checkMeans[,1:139], na.rm = T)
1109 # Keep only rows where mean value of detected WAPs is more than 0.5
1110 checkMeans <- subset(checkMeans, rowMeans(checkMeans[,1:139], na.rm = T) > 0.6)
1111 checkMeans[is.na(checkMeans)] <- 0
1112
1113 df <- checkMeans
1114
1115
1116 set.seed(123)
1117 ctrl <- trainControl(method="cv", number = 10)
1118 knnFit <- train((LATITUDE ~ .), data = df, method = "knn", trControl = ctrl, tun
1119
1120 #Output of kNN fit
1121 knnFit
1122 #write.csv(knnFit, file = 'knnperformance.csv')
1123
1124 # test the k-NN model
1125 knnPredict <- predict(knnFit, newdata = testSet)
1126 postResample(knnPredict, testSet$LATITUDE)
1127
1128 # Check results on validation dataset
1129 # Apply k-NN model to the validation data
1130 knnPredictttest <- predict(knnFit, newdata = validation)
1131 postResample(knnPredictttest, validation$LATITUDE)
1132
1133 # ~~~~~ KNN ~~~~~
1134 # Performance RMSE Rsquared MAE
1135 # 7.1845765 0.9601064 5.0841969
1136
1137 # Save results in csv file
1138 #write.csv(knnPredictttest, file = "knnPredictLAT.csv")
1139 LatPred1 <- as.data.frame(knnPredictttest)
1140 LONGLAT_PREDICTIONS <- as.data.frame(c(LongPred1, LatPred1))
1141
1142 # change column names
1143 colnames(LONGLAT_PREDICTIONS)[1] <- 'LONGITUDE'

```

```

1144 colnames(LONGLAT_PREDICTIONS)[2] <- 'LATITUDE'
1145
1146 # Plot real and predicted results
1147 # Training and Validation log in locations
1148 ggplot() +
1149   geom_point(data = LONGLAT_PREDICTIONS , aes(x = LONGITUDE, y = LATITUDE, colour = LONGITUDE)) +
1150   geom_point(data = validationLONGLAT , aes(x = LONGITUDE, y = LATITUDE, colour = LONGITUDE)) +
1151   ggtitle("Log In Locations")
1152
1153 # Distribution of distance error (in meters)
1154 Error = sqrt((LONGLAT_PREDICTIONS$LONGITUDE - validationLONGLAT$LONGITUDE)^2 + (LONGLAT_PREDICTIONS$LATITUDE - validationLONGLAT$LATITUDE)^2)
1155 hist(Error, freq = T, xlab = " Absolute error (m)", col = "red", main = "Error distribution")
1156
1157 # PREDICT COORDINATES OF BUILDING 2
1158 #LONGITUDE
1159 try <- Building2[,1:207]
1160 # Remove columns (WAP) where all the values = 0 (WAP was not detected)
1161 uniquelength <- sapply(try,function(x) length(unique(x)))
1162 try <- subset(try, select=uniquelength>1)
1163 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
1164 keep <- apply(try[,1:207], 1, function(x) length(unique(x[!is.na(x)])) != 1)
1165 try <- try[keep, ]
1166
1167 # Normalize the data
1168 data_norm <- as.data.frame(t(apply(try, 1, function(x) (x - min(x))/(max(x)-min(x)))))
1169
1170
1171 # Build knn model
1172 uniquelength <- sapply(Building2,function(x) length(unique(x)))
1173 Building2 <- subset(Building2, select=uniquelength>1)
1174 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
1175 keep <- apply(Building2[,1:207], 1, function(x) length(unique(x[!is.na(x)])) != 1)
1176 Building2 <- Building2[keep, ]
1177
1178 Building2[,1:207] <- data_norm
1179
1180 Building2$LONGITUDE <- Building2$LONGITUDE
1181 cols <- c(1:208)
1182 training <- Building2[ , cols]
1183 training$LONGITUDE <- training$LONGITUDE
1184
1185 # Import validation dataset
1186 validation <- subset(testData, BUILDINGID == 1)
1187 testData2 <- subset(testData, BUILDINGID == 1)
1188

```

```

1189 # Remove columns (WAP) where all the values = 0 (WAP was not detected)
1190 unqulength <- sapply(validation,function(x) length(unique(x)))
1191 validation <- subset(validation, select=unqulength>1)
1192 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
1193 keep <- apply(validation[,1:170], 1, function(x) length(unique(x[!is.na(x)])) !=
1194 validation <- validation[keep, ]
1195
1196 # Drop columns from validation that do not match with training set
1197 cols_to_keep <- intersect(colnames(training),colnames(validation))
1198 training <- training[,cols_to_keep, drop=FALSE]
1199 validation <- validation[,cols_to_keep, drop=FALSE]
1200 validation <- as.data.frame(t(apply(validation[,1:146], 1, function(x) (x - min(
1201
1202 validation$LONGITUDE <- testData2$LONGITUDE
1203 validation$LONGITUDE <- validation$LONGITUDE
1204
1205 validationLONG <- as.data.frame(testData2$LONGITUDE)
1206
1207 set.seed(123)
1208 trainIndex <- createDataPartition(y = training$LONGITUDE, p = 0.75,
1209 list = FALSE)
1210
1211
1212 # ~~~~~ Training and Test sets ~~~~~
1213
1214 trainSet <- training [trainIndex,]
1215 testSet <- training [-trainIndex,]
1216
1217 # ~~~~~ K-NN ~~~~~
1218
1219 df <- trainSet
1220
1221 checkMeans <- df
1222 checkMeans[,1:146][checkMeans[,1:146] == 0] <- NA
1223 rowMeans(checkMeans[,1:146], na.rm = T)
1224 # Keep only rows where mean value of detected WAPs is more than 0.5
1225 checkMeans <- subset(checkMeans, rowMeans(checkMeans[,1:146], na.rm = T) > 0.6)
1226 checkMeans[is.na(checkMeans)] <- 0
1227
1228 df <- checkMeans
1229
1230
1231 set.seed(123)
1232 ctrl <- trainControl(method="cv",number = 10)
1233 knnFit <- train((LONGITUDE ~ .), data = df, method = "knn", trControl = ctrl, tu

```

```

1234
1235 #Output of kNN fit
1236 knnFit
1237
1238 # test the k-NN model
1239 knnPredict <- predict(knnFit,newdata = testSet)
1240 postResample(knnPredict, testSet$LONGITUDE)
1241
1242 # Check results on validation dataset
1243 # Apply k-NN model to the validation data
1244 knnPredictttest <- predict(knnFit,newdata = validation)
1245 postResample(knnPredictttest, validation$LONGITUDE)
1246
1247 #           ~~~~~ KNN ~~~~~
1248 # Performance   RMSE       Rsquared       MAE
1249 #           9.385072  0.965634       7.114257
1250
1251
1252 # Save results in csv file
1253 #write.csv(knnPredictttest, file = "knnPredict.csv")
1254 LongPred2 <- as.data.frame(knnPredictttest)
1255
1256 # LATITUDE
1257 try <- Building2[,1:207]
1258 # Remove columns (WAP) where all the values = 0 (WAP was not detected)
1259 uniquelength <- sapply(try,function(x) length(unique(x)))
1260 try <- subset(try, select=uniquelength>1)
1261 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
1262 keep <- apply(try[,1:207], 1, function(x) length(unique(x[!is.na(x)])) != 1)
1263 try <- try[keep, ]
1264
1265 # Normalize the data
1266 data_norm <- as.data.frame(t(apply(try, 1, function(x) (x - min(x))/(max(x)-min(
1267
1268
1269 # Build knn model
1270 uniquelength <- sapply(Building2,function(x) length(unique(x)))
1271 Building2 <- subset(Building2, select=uniquelength>1)
1272 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
1273 keep <- apply(Building2[,1:207], 1, function(x) length(unique(x[!is.na(x)])) !=
1274 Building2 <- Building2[keep, ]
1275
1276 Building2[,1:207] <- data_norm
1277
1278 Building2$LATITUDE <- Building2$LATITUDE

```

```

1279 cols <- c(1:207, 209)
1280 training <- Building2[ , cols]
1281 training$LATITUDE <- training$LATITUDE
1282
1283 # Import validation dataset
1284 validation <- subset(testData, BUILDINGID == 1)
1285 testData2 <- subset(testData, BUILDINGID == 1)
1286
1287 # Remove columns (WAP) where all the values = 0 (WAP was not detected)
1288 uniquelength <- sapply(validation,function(x) length(unique(x)))
1289 validation <- subset(validation, select=uniquelength>1)
1290 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
1291 keep <- apply(validation[,1:170], 1, function(x) length(unique(x[!is.na(x)])) !=
1292 validation <- validation[keep, ]
1293
1294 # Drop columns from validation that do not match with training set
1295 cols_to_keep <- intersect(colnames(training),colnames(validation))
1296 training <- training[,cols_to_keep, drop=FALSE]
1297 validation <- validation[,cols_to_keep, drop=FALSE]
1298 validation <- as.data.frame(t(apply(validation[,1:146], 1, function(x) (x - min(
1299
1300 validation$LATITUDE <- testData2$LATITUDE
1301 validation$LATITUDE <- validation$LATITUDE
1302
1303 validationLAT <- as.data.frame(testData2$LATITUDE)
1304 validationLONG <- as.data.frame(testData2$LONGITUDE)
1305 validationONGLAT <- as.data.frame(c(validationLONG, validationLAT))
1306 # Change the column names of validationONGLAT
1307 colnames(validationONGLAT)[1] <- "LONGITUDE"
1308 colnames(validationONGLAT)[2] <- "LATITUDE"
1309
1310 set.seed(123)
1311 trainIndex <- createDataPartition(y = training$LONGITUDE, p = 0.75,
1312 list = FALSE)
1313
1314
1315 # ~~~~~ Training and Test sets ~~~~~
1316
1317 trainSet <- training [trainIndex,]
1318 testSet <- training [-trainIndex,]
1319
1320 # ~~~~~ K-NN ~~~~~
1321
1322 df <- trainSet
1323

```

```

1324 checkMeans <- df
1325 checkMeans[,1:146][checkMeans[,1:146] == 0] <- NA
1326 rowMeans(checkMeans[,1:146], na.rm = T)
1327 # Keep only rows where mean value of detected WAPs is more than 0.5
1328 checkMeans <- subset(checkMeans, rowMeans(checkMeans[,1:146], na.rm = T) > 0.6)
1329 checkMeans[is.na(checkMeans)] <- 0
1330
1331 df <- checkMeans
1332
1333
1334 set.seed(123)
1335 ctrl <- trainControl(method="cv", number = 10)
1336 knnFit <- train((LATITUDE ~ .), data = df, method = "knn", trControl = ctrl, tun
1337
1338 #Output of kNN fit
1339 knnFit
1340
1341 # test the k-NN model
1342 knnPredict <- predict(knnFit, newdata = testSet)
1343 postResample(knnPredict, testSet$LATITUDE)
1344
1345 # Check results on validation dataset
1346 # Apply k-NN model to the validation data
1347 knnPredicttest <- predict(knnFit, newdata = validation)
1348 postResample(knnPredicttest, validation$LATITUDE)
1349
1350 # ~~~~~ KNN ~~~~~
1351 # Performance RMSE Rsquared MAE
1352 # 11.4652267 0.9069251 8.1506292
1353
1354 # Save results in csv file
1355 #write.csv(knnPredicttest, file = "knnPredictLAT.csv")
1356 LatPred2 <- as.data.frame(knnPredicttest)
1357 LONGLAT_PREDICTIONS <- as.data.frame(c(LongPred2, LatPred2))
1358
1359 # change column names
1360 colnames(LONGLAT_PREDICTIONS)[1] <- 'LONGITUDE'
1361 colnames(LONGLAT_PREDICTIONS)[2] <- 'LATITUDE'
1362
1363 # Plot real and predicted results
1364 # Training and Validation log in locations
1365 ggplot() +
1366   geom_point(data = LONGLAT_PREDICTIONS, aes(x = LONGITUDE, y = LATITUDE, colour
1367   geom_point(data = validationLONGLAT, aes(x = LONGITUDE, y = LATITUDE, colour
1368   ggtitle("Log In Locations")

```

```

1369
1370 # Distribution of distance error (in meters)
1371 Error = sqrt((LOGLAT_PREDICTIONS$LONGITUDE - validationLOGLAT$LONGITUDE)^2 +(L
1372 hist(Error, freq = T, xlab = " Absolute error (m)", col = "red", main = "Error d
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385 # PREDICT COORDINATES OF BUILDING 3
1386 #LONGITUDE
1387 try <- Building3[,1:203]
1388 # Remove columns (WAP) where all the values = 0 (WAP was not detected)
1389 unqulength <- sapply(try,function(x) length(unique(x)))
1390 try <- subset(try, select=unqulength>1)
1391 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
1392 keep <- apply(try[,1:203], 1, function(x) length(unique(x[!is.na(x)])) != 1)
1393 try <- try[keep, ]
1394
1395 # Normalize the data
1396 data_norm <- as.data.frame(t(apply(try, 1, function(x) (x - min(x))/(max(x)-min(
1397
1398
1399 # Build knn model
1400 unqulength <- sapply(Building3,function(x) length(unique(x)))
1401 Building3 <- subset(Building3, select=unqulength>1)
1402 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
1403 keep <- apply(Building3[,1:203], 1, function(x) length(unique(x[!is.na(x)])) !=
1404 Building3 <- Building3[keep, ]
1405
1406 Building3[,1:203] <- data_norm
1407
1408 Building3$LONGITUDE <- Building3$LONGITUDE
1409 cols <- c(1:204)
1410 training <- Building3[ , cols]
1411 training$LONGITUDE <- training$LONGITUDE
1412
1413 # Import validation dataset

```



```

1414 validation <- subset(testData, BUILDINGID == 2)
1415 testData3 <- subset(testData, BUILDINGID == 2)
1416
1417 # Remove columns (WAP) where all the values = 0 (WAP was not detected)
1418 uniquelength <- sapply(validation,function(x) length(unique(x)))
1419 validation <- subset(validation, select=uniquelength>1)
1420 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
1421 keep <- apply(validation[,1:123], 1, function(x) length(unique(x[!is.na(x)])) !=
1422 validation <- validation[keep, ]
1423
1424 # Drop columns from validation that do not match with training set
1425 cols_to_keep <- intersect(colnames(training),colnames(validation))
1426 training <- training[,cols_to_keep, drop=FALSE]
1427 validation <- validation[,cols_to_keep, drop=FALSE]
1428 validation <- as.data.frame(t(apply(validation[,1:104], 1, function(x) (x - min(
1429
1430 validation$LONGITUDE <- testData3$LONGITUDE
1431 validation$LONGITUDE <- validation$LONGITUDE
1432
1433 validationLONG <- as.data.frame(testData3$LONGITUDE)
1434
1435 set.seed(123)
1436 trainIndex <- createDataPartition(y = training$LONGITUDE, p = 0.75,
1437 list = FALSE)
1438
1439
1440 # ~~~~~ Training and Test sets ~~~~~
1441
1442 trainSet <- training [trainIndex,]
1443 testSet <- training [-trainIndex,]
1444
1445 # ~~~~~ K-NN ~~~~~
1446
1447 df <- trainSet
1448
1449 checkMeans <- df
1450 checkMeans[,1:104][checkMeans[,1:104] == 0] <- NA
1451 rowMeans(checkMeans[,1:104], na.rm = T)
1452 # Keep only rows where mean value of detected WAPs is more than 0.5
1453 checkMeans <- subset(checkMeans, rowMeans(checkMeans[,1:104], na.rm = T) > 0.6)
1454 checkMeans[is.na(checkMeans)] <- 0
1455
1456 df <- checkMeans
1457
1458

```

```

1459 set.seed(123)
1460 ctrl <- trainControl(method="cv",number = 10)
1461 knnFit <- train((LONGITUDE ~ .), data = df, method = "knn", trControl = ctrl, tu
1462
1463 #Output of knn fit
1464 knnFit
1465
1466 # test the k-NN model
1467 knnPredict <- predict(knnFit,newdata = testSet)
1468 postResample(knnPredict, testSet$LONGITUDE)
1469
1470 # Check results on validation dataset
1471 # Apply k-NN model to the validation data
1472 knnPredictttest <- predict(knnFit,newdata = validation)
1473 postResample(knnPredictttest, validation$LONGITUDE)
1474
1475 #           ~~~~~ KNN ~~~~~
1476 # Performance   RMSE      Rsquared      MAE
1477 #           12.8063785  0.8358643  9.0960994
1478
1479
1480 # Save results in csv file
1481 #write.csv(knnPredictttest, file = "knnPredict.csv")
1482 LongPred3 <- as.data.frame(knnPredictttest)
1483
1484 # LATITUDE
1485 try <- Building3[,1:203]
1486 # Remove columns (WAP) where all the values = 0 (WAP was not detected)
1487 uniquelength <- sapply(try,function(x) length(unique(x)))
1488 try <- subset(try, select=uniquelength>1)
1489 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
1490 keep <- apply(try[,1:203], 1, function(x) length(unique(x[!is.na(x)])) != 1)
1491 try <- try[keep, ]
1492
1493 # Normalize the data
1494 data_norm <- as.data.frame(t(apply(try, 1, function(x) (x - min(x))/(max(x)-min(
1495
1496
1497 # Build knn model
1498 uniquelength <- sapply(Building3,function(x) length(unique(x)))
1499 Building3 <- subset(Building3, select=uniquelength>1)
1500 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
1501 keep <- apply(Building3[,1:203], 1, function(x) length(unique(x[!is.na(x)])) !=
1502 Building3 <- Building3[keep, ]
1503

```

```

1504 Building3[,1:203] <- data_norm
1505
1506 Building3$LONGITUDE <- Building3$LONGITUDE
1507 cols <- c(1:203, 205)
1508 training <- Building3[ , cols]
1509 training$LATITUDE <- training$LATITUDE
1510
1511 # Import validation dataset
1512 validation <- subset(testData, BUILDINGID == 2)
1513 testData3 <- subset(testData, BUILDINGID == 2)
1514
1515 # Remove columns (WAP) where all the values = 0 (WAP was not detected)
1516 uniquelength <- sapply(validation,function(x) length(unique(x)))
1517 validation <- subset(validation, select=uniquelength>1)
1518 # Remove rows (WAP) where all the values = 0 (WAP was not detected)
1519 keep <- apply(validation[,1:123], 1, function(x) length(unique(x[!is.na(x)])) !=
1520 validation <- validation[keep, ]
1521
1522 # Drop columns from validation that do not match with training set
1523 cols_to_keep <- intersect(colnames(training),colnames(validation))
1524 training <- training[,cols_to_keep, drop=FALSE]
1525 validation <- validation[,cols_to_keep, drop=FALSE]
1526 validation <- as.data.frame(t(apply(validation[,1:104], 1, function(x) (x - min(
1527
1528 validation$LATITUDE <- testData3$LATITUDE
1529 validation$LATITUDE <- validation$LATITUDE
1530
1531 validationLAT <- as.data.frame(testData3$LATITUDE)
1532
1533 set.seed(123)
1534 trainIndex <- createDataPartition(y = training$LATITUDE, p = 0.75,
1535                                   list = FALSE)
1536
1537
1538 # ~~~~~ Training and Test sets ~~~~~
1539
1540 trainSet <- training [trainIndex,]
1541 testSet <- training [-trainIndex,]
1542
1543 # ~~~~~ K-NN ~~~~~
1544
1545 df <- trainSet
1546
1547 checkMeans <- df
1548 checkMeans[,1:104][checkMeans[,1:104] == 0] <- NA

```

```

1549 rowMeans(checkMeans[,1:104], na.rm = T)
1550 # Keep only rows where mean value of detected WAPs is more than 0.5
1551 checkMeans <- subset(checkMeans, rowMeans(checkMeans[,1:104], na.rm = T) > 0.6)
1552 checkMeans[is.na(checkMeans)] <- 0
1553
1554 df <- checkMeans
1555
1556
1557 set.seed(123)
1558 ctrl <- trainControl(method="cv", number = 10)
1559 knnFit <- train((LATITUDE ~ .), data = df, method = "knn", trControl = ctrl, tun
1560
1561 #Output of kNN fit
1562 knnFit
1563
1564 # test the k-NN model
1565 knnPredict <- predict(knnFit, newdata = testSet)
1566 postResample(knnPredict, testSet$LATITUDE)
1567
1568 # Check results on validation dataset
1569 # Apply k-NN model to the validation data
1570 knnPredictttest <- predict(knnFit, newdata = validation)
1571 postResample(knnPredictttest, validation$LATITUDE)
1572
1573 # ~~~~~ KNN ~~~~~
1574 # Performance RMSE Rsquared MAE
1575 # 9.9643518 0.8955963 7.4651590
1576
1577 # Save results in csv file
1578 #write.csv(knnPredictttest, file = "knnPredictLAT.csv")
1579 LatPred3 <- as.data.frame(knnPredictttest)
1580 LONGLAT_PREDICTIONS <- as.data.frame(c(LongPred3, LatPred3))
1581
1582 validationLONG <- testData3$LONGITUDE
1583 validationLAT <- testData3$LATITUDE
1584
1585 validationONGLAT <- as.data.frame(c(validationLONG, validationLAT))
1586 colnames(validationONGLAT)[1] <- 'LONGITUDE'
1587 colnames(validationONGLAT)[2] <- 'LATITUDE'
1588
1589 # change column names
1590 colnames(LONGLAT_PREDICTIONS)[1] <- 'LONGITUDE'
1591 colnames(LONGLAT_PREDICTIONS)[2] <- 'LATITUDE'
1592
1593 # Plot real and predicted results

```

```
1594 # Training and Validation log in locations
1595 ggplot() +
1596   geom_point(data = LONGLAT_PREDICTIONS , aes(x = LONGITUDE, y = LATITUDE, colour = "Predicted")) +
1597   geom_point(data = testData3 , aes(x = LONGITUDE, y = LATITUDE, colour = "Real"))
1598   ggtitle("Log In Locations")
1599
1600 # Distribution of distance error (in meters)
1601 Error = sqrt((LONGLAT_PREDICTIONS$LONGITUDE - testData3$LONGITUDE)^2 + (LONGLAT_PREDICTIONS$LATITUDE - testData3$LATITUDE)^2)
1602 hist(Error, freq = T, xlab = " Absolute error (m)", col = "red", main = "Error distribution")
1603
1604
1605
1606
```