Description of the data science process

* **The goal of this project**

The goal of this project is to evaluate multiple machine learning models to see which produces the best result, enabling our company to make a recommendation to the client. If the recommended model is sufficiently accurate, it will be incorporated into a smartphone app for indoor positioning.

* **Description and location of related data sources**

The initial dataset was acquired from the Machine Learning Repository website in the CSV format. It was created in 2013 by means of more than 20 different users and 25 Android devices. There is a total of 19937 observations and 529 variables, 520 of which are Wireless Access Points’ fingerprint values. The intensity values are represented as negative integer values ranging -104dBm (extremely poor signal) to 0dbM. The positive value 100 is used to denote when a WAP was not detected.

The rest 9 features are latitude, longitude, building id, floor and space id. These coordinates are the aim of our research since they are going to be predicted.

Additionally, there were user id, phone id (Android phone models) and timestamp (time when the capture was taken) columns. User id and timestamp were dropped, but phone id was initially dropped but then included again for sake of the experimentation.

* **Data preprocessing**

There were two different approaches were applied for the dimensional reduction. The X subset was reduced by excluding variables with 0 variances, so that it could be processed in a reasonable period of time. To reduce y subset, all columns that contained positioning information were merged into a one column the following way:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| BUILDING ID | FLOOR | SPACE ID | RELATIVE POSITON |  | COMBINED |
| 0 | 3 | 17 | 1 | -> | 030171 |

Since “space id” values are variates from length 1 to 3, leading “0” were added to equalize the length to 3. That way we could simply decode the combined location.

Eventually, only the following columns were kept for further model development:

WAP1 – WAP520 (-zero variance): X subset, numeric

COMBINED: y subset, categorical (905 categories).

* **Known issues and solutions**

Because the testing dataset does not contain “space id” values, it was impossible to use it for the model validation. In order to cope with this issue, the initial dataset was split into two parts – training and testing in a proportion of 75/25.

Comparison of the models

During the research, several algorithms were tested in different modifications. Among them:

C5.0, Random Forest and KNN. Also, GBM failed to finish the training process because of the unknown issue of parallel computation.

**C5.0 (Decision Tree) #1**

Initial state:

Predictors: WAP001 – WAP520 (columns with zero variance are on the place) - 520

Target: Combined location = “building id” + “floor” + “space id” + “relative position”

Training/Testing subsets: 75/25, sampled, whole dataset.

Control:

> fitControl0 <- trainControl(method = "repeatedcv", number = 10, repeats = 1)

(will be used for the all experiments)

Tune length: 1

Outcomes:

15184 samples

520 predictor

905 classes

No pre-processing

Resampling: Cross-Validated (10 fold, repeated 1 times)

Summary of sample sizes: 13692, 13646, 13678, 13650, 13650, 13671, ...

Resampling results across tuning parameters:

model winnow Accuracy Kappa

rules FALSE 0.5953361 0.5947588

rules TRUE 0.5906670 0.5900848

tree FALSE 0.5995897 0.5990263

**tree TRUE 0.6003682 0.5998058**

C5.0 with tuneLength = 1 and all WAP columns resulted in very poor accuracy and kappa. Obviously, this model should be tuned, however, there is a chance for the model to be overfitted.

**C5.0 (Decision Tree) #2**

Initial state:

Predictors: WAP001 – WAP520 (columns with zero variance are on the place) - 520

Target: Combined location = “building id” + “floor” + “space id” + “relative position”

Training/Testing subsets: 75/25, sampled, whole dataset.

Control:

> fitControl0 <- trainControl(method = "repeatedcv", number = 10, repeats = 1)

(will be used for the all experiments)

Tune length: 3

Outcomes:

15184 samples

520 predictor

905 classes

No pre-processing

Resampling: Cross-Validated (10 fold, repeated 1 times)

Summary of sample sizes: 13653, 13669, 13678, 13675, 13644, 13663, ...

Resampling results across tuning parameters:

model winnow trials Accuracy Kappa

rules FALSE 1 0.5953475 0.5947672

rules FALSE 10 0.7203599 0.7199615

rules FALSE 20 0.7339795 0.7336009

rules TRUE 1 0.5912492 0.5906656

rules TRUE 10 0.7210043 0.7206076

rules TRUE 20 0.7386693 0.7382985

tree FALSE 1 0.6058933 0.6053373

tree FALSE 10 0.7185098 0.7181061

tree FALSE 20 0.7304480 0.7300622

tree TRUE 1 0.6009930 0.6004301

tree TRUE 10 0.7224080 0.7220112

**tree TRUE 20 0.7343865 0.7340056**

Increasing tuneLength to 3, we can notice a significant increase in accuracy and kappa. Still, these values are not usable for the positioning app due to their low accuracy.

Using a tuneLength parameter higher than 3 outcomes longer computation time and, again, it causes the model’s overfitting.

**C5.0 (Decision Tree) #3**

Initial state:

Predictors: WAP001 – WAP520 (dropped columns with zero variance) + “phone id” - 466

Target: Combined location = “building id” + “floor” + “space id” + “relative position”

Training/Testing subsets: 75/25, sampled, whole dataset.

Control:

> fitControl0 <- trainControl(method = "repeatedcv", number = 10, repeats = 1)

(will be used for the all experiments)

Tune length: 3

Outcomes:

15184 samples

466 predictor

905 classes

No pre-processing

Resampling: Cross-Validated (10 fold, repeated 1 times)

Summary of sample sizes: 13672, 13695, 13662, 13672, 13652, 13643, ...

Resampling results across tuning parameters:

model winnow trials Accuracy Kappa

rules FALSE 1 0.5991773 0.5986085

rules FALSE 10 0.7254067 0.7250163

**rules FALSE 20 0.7443916 0.7440276**

rules TRUE 1 0.5952700 0.5946926

rules TRUE 10 0.7210855 0.7206887

rules TRUE 20 0.7387474 0.7383754

tree FALSE 1 0.6057704 0.6052167

tree FALSE 10 0.7208407 0.7204420

tree FALSE 20 0.7371576 0.7367817

tree TRUE 1 0.6025927 0.6020320

tree TRUE 10 0.7209752 0.7205765

tree TRUE 20 0.7347792 0.7344005

> C5Pred <- predict(C5Fit0, testing\_loc)

> postResample(C5Pred, testing\_loc$combined\_loc)

Accuracy Kappa

0.7428992 0.7425399

Dropping zero-variance columns does not substantially improve the Accuracy and Kappa score for the C5.0 model

**rf (Random Forest) #1**

Initial state:

Predictors: WAP001 – WAP520 (columns with zero variance are on the place) - 520

Target: Combined location = “building id” + “floor” + “space id” + “relative position”

Training/Testing subsets: 75/25, sampled, whole dataset.

Control:

> fitControl0 <- trainControl(method = "repeatedcv", number = 10, repeats = 1)

(will be used for the all experiments)

Tune length: 1

Outcomes:

15184 samples

520 predictor

905 classes

No pre-processing

Resampling: Cross-Validated (10 fold, repeated 1 times)

Summary of sample sizes: 13691, 13670, 13660, 13698, 13640, 13659, ...

Resampling results:

Accuracy Kappa

0.8088666 0.8085913

Tuning parameter 'mtry' was held constant at a value of 173

> rfPred <- predict(rfFit, testing\_loc)

> postResample(rfPred, testing\_loc$combined\_loc)

Accuracy Kappa

0.8182201 0.8179623

Random Forest algorithm shows a decent result without zero-variance columns dropping. In general, this accuracy could be considered as sufficient for the positioning app building. However, according to the data acquired, the better model could be built.

**rf (Random Forest) #2**

Initial state:

Predictors: WAP001 – WAP520 (dropped columns with zero variance) - 465

Target: Combined location = “building id” + “floor” + “space id” + “relative position”

Training/Testing subsets: 75/25, sampled, whole dataset.

Control:

> fitControl0 <- trainControl(method = "repeatedcv", number = 10, repeats = 1)

Tune length: 1

Outcomes:

15184 samples

465 predictor

905

No pre-processing

Resampling: Cross-Validated (10 fold, repeated 1 times)

Summary of sample sizes: 13679, 13648, 13652, 13691, 13663, 13635, ...

Resampling results:

Accuracy Kappa

0.8109413 0.8106682

Tuning parameter 'mtry' was held constant at a value of 155

> rfPred <- predict(rfFit, testing\_loc)

> postResample(rfPred, testing\_loc$combined\_loc)

Accuracy Kappa

0.8245319 0.8242835

Dropping zero-variance columns did not bring any significant improvement, only 0.6% that is within a statistical error.

**rf (Random Forest) #3 (simplified position)**

Initial state:

Predictors: WAP001 – WAP520 (dropped columns with zero variance) - 465

Target: Combined location = “building id” + “floor” + “space id” (w/o “relative position”)

Training/Testing subsets: 75/25, sampled, whole dataset.

Control:

> fitControl0 <- trainControl(method = "repeatedcv", number = 10, repeats = 1)

(will be used for the all experiments)

Tune length: 1

Outcomes:

15184 samples

465 predictor

905

Resampling results:

> rfFit\_s <- train(combined\_loc~., data = training\_loc\_s, method = "rf", trControl=fitControl0, tuneLength = 1)

Accuracy Kappa

0.817145 0.8167928

Tuning parameter 'mtry' was held constant at a value of 1

> rfPred\_s <- predict(rfFit\_s, testing\_loc\_s)

> postResample(rfPred\_s, testing\_loc\_s$combined\_loc)

Accuracy Kappa

0.8274641 0.8271346

Again, excluding “relative position” from the target variable the overall accuracy increased just slightly, within the statistical error, though it should improve the prediction by a generalization of the y variable.

**rf (Random Forest) #4 (filtering on “phone id”)**

Initial state:

Predictors: WAP001 – WAP520 (dropped columns with zero variance) - 465

Target: Combined location = “building id” + “floor” + “space id” + “relative position”

Training/Testing subsets: 75/25, sampled, whole dataset.

Control:

> df\_gt <- filter(df\_gt, between(PHONEID, 1, 7)) (filtered only models with GT prefix)

> fitControl0 <- trainControl(method = "repeatedcv", number = 10, repeats = 1)

(will be used for the all experiments)

Tune length: 3

Outcomes:

No pre-processing

Resampling: Cross-Validated (10 fold, repeated 1 times)

Summary of sample sizes: 2879, 2866, 2860, 2878, 2875, 2878, ...

Resampling results across tuning parameters:

mtry Accuracy Kappa

2 0.1754065 0.1683280

129 0.8541341 0.8534378

257 0.8415757 0.8408249

Accuracy was used to select the optimal model using the largest value.

The final value used for the model was mtry = 129.

> rfPred\_gt <- predict(rfFit\_gt, testing\_gt)

> postResample(rfPred\_gt, testing\_gt$combined\_gt)

Accuracy Kappa

**0.8634850 0.8628143**

The idea that underlies this experiment stems from the fact that different manufacturers fit their phones with different Wi-Fi modules. Because of this, standing at the same point with different phones, we would get different signal strengths for each. To try this idea, the model trained only using GT series phones (SAMSUNG) was built.

The results seem to be significantly improved. Due to this, it makes sense to develop a positioning app and use there not only one prediction model, but several, trained on different phone models. This could lead to the necessity of frequent app updates but, eventually, would guarantee a higher positioning accuracy.

**knn (k Nearest Neighbours)**

Initial state:

Predictors: WAP001 – WAP520 (dropped columns with zero variance - 465

Target: Combined location = “building id” + “floor” + “space id” + “relative position”

Training/Testing subsets: 75/25, sampled, whole dataset.

Control:

> fitControl0 <- trainControl(method = "repeatedcv", number = 10, repeats = 1)

> knnFit <- train(combined\_loc~., data = training\_loc, method = "knn", trControl=fitControl\_knn, preProcess = c("center","scale"), tuneGrid = expand.grid(k = 1:5))

Outcomes:

15184 samples

466 predictor

905 classes

Pre-processing: centered (466), scaled (466)

Resampling: Cross-Validated (10 fold, repeated 1 times)

Summary of sample sizes: 13680, 13687, 13658, 13649, 13689, 13666, ...

Resampling results across tuning parameters:

k Accuracy Kappa

1 0.6450171 0.6445157

2 0.5974665 0.5968987

3 0.5999946 0.5994296

4 0.5868804 0.5862950

5 0.5762766 0.5756739

Accuracy was used to select the optimal model using the largest value.

The final value used for the model was k = 1.

> knnPred <- predict(knnFit, testing\_loc)

> postResample(knnPred, testing\_loc$combined\_loc)

Accuracy Kappa

0.6587418 0.6582589

The higher the number of k, the lower the model’s accuracy, even though this model seemed to be very promising for this task. The next experiment was conducted in Python to see how training accuracy correlates with testing one.

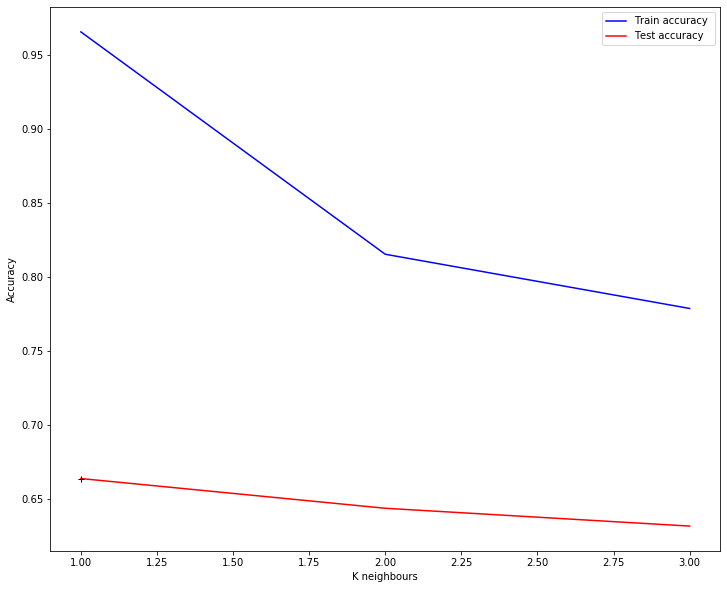
**knn (k Nearest Neighbours)**

Initial state:

Predictors: WAP001 – WAP520 (columns with zero variance are on the place) - 520

Target: Combined location = “building id” + “floor” + “space id” + “relative position”

Training/Testing subsets: 95/5, sampled, whole dataset.



According to the experiment, the best result is shown for k=1. For training set the accuracy is almost perfect, however, applying it to a validation set shows a very poor result. This can be explained by the fact that we use training set as training and testing when this is not correct because allocating training subset, we deprive our model crucial information about the location. Afterward, when we want the model to predict the location this model never taught, we get a weak result.

To overcome this issue, it is reasonable to use a separate validation set instead but since it does not contain “space id” values, this is impossible.

Model selection

To my mind, the best accuracy and kappa showed at the validation test was Random Forest with filtering on the phone’s model. This makes sense because different phones are fitted with different Wi-Fi modules and, consequently, are likely to have different signal sensitivity. Also, compared to C5.0, random forests are less prone to overfitting. The overall accuracy of the model is **86%** and the same for a Cohen Kappa coefficient that is considered as high.

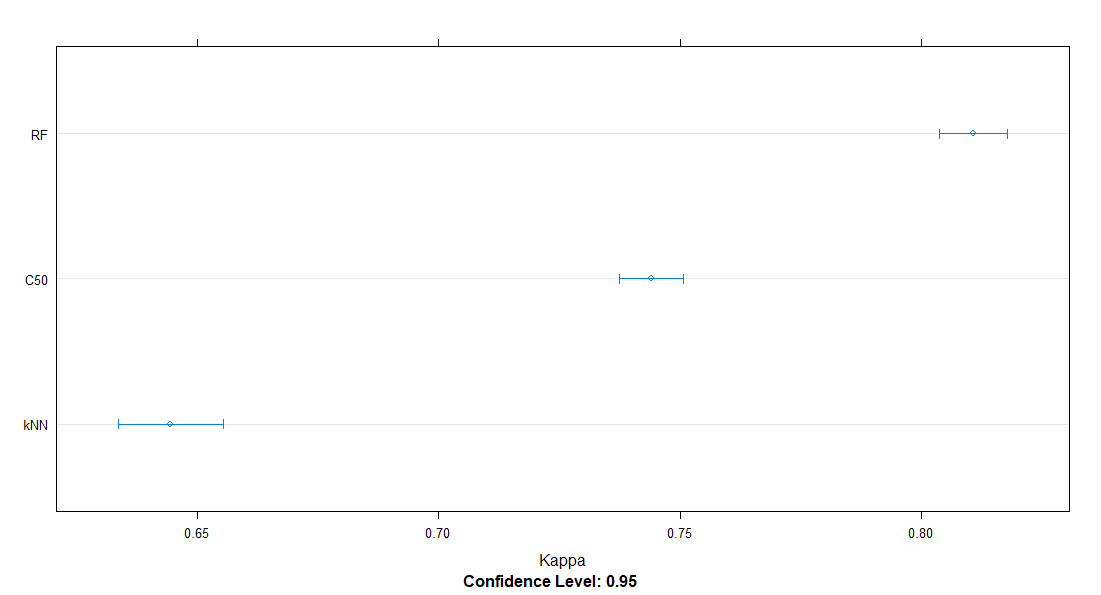
One significant drawback of this model is that the mobile positioning app, based on it, has to be regularly updated and many initial measurements have to be conducted for each phone manufacturer. For the purpose of correct model comparison, I chose an RF model where there was no filtration on the phone’s model. This model still acceptable and shows **82.4%** Kappa score at the validation test.

Recommendations on accuracy improvement

As it was mentioned previously, I believe that kNN with k=1 could be used as the best prediction model for this dataset. This is also confirmed in “UJIIndoorLoc: A New Multi-building and Multi-floor Database for WLAN Fingerprint-based Indoor Localization Problems” where authors managed to get an accuracy rate of 89.92% for the validation dataset using 1NN model

Nonetheless, we do not have the validation dataset complete and have to use another method described above.

Kappa score comparison



Model comparison

> modeldata <- resamples(list(C50 = C5Fit0, RF = rfFit, kNN = knnFit))

> summary(modeldata)

Call:

summary.resamples(object = modeldata)

Models: C50, RF, kNN

Number of resamples: 10

Accuracy

Min. 1st Qu. Median Mean 3rd Qu. Max. NA's

C50 0.7281746 0.7391834 0.7444214 0.7443916 0.7523460 0.7549669 0

RF 0.7926910 0.8055553 0.8122329 0.8109413 0.8132854 0.8297587 0

kNN 0.6138291 0.6353653 0.6511478 0.6450171 0.6547254 0.6628763 0

Kappa

Min. 1st Qu. Median Mean 3rd Qu. Max. NA's

C50 0.7277856 0.7388166 0.7440570 0.7440276 0.7519923 0.7546185 0

RF 0.7923933 0.8052713 0.8119646 0.8106682 0.8130168 0.8295112 0

kNN 0.6132950 0.6348406 0.6506577 0.6445157 0.6542393 0.6623974 0

> summary(kappaDiffs)

Call:

summary.diff.resamples(object = kappaDiffs)

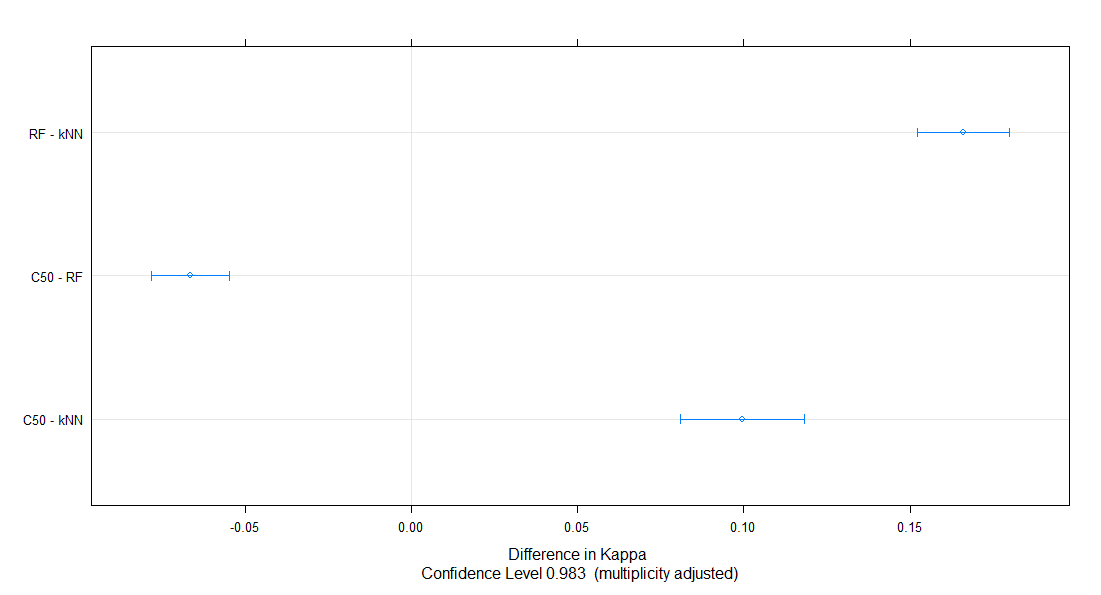
p-value adjustment: bonferroni

Upper diagonal: estimates of the difference

Lower diagonal: p-value for H0: difference = 0

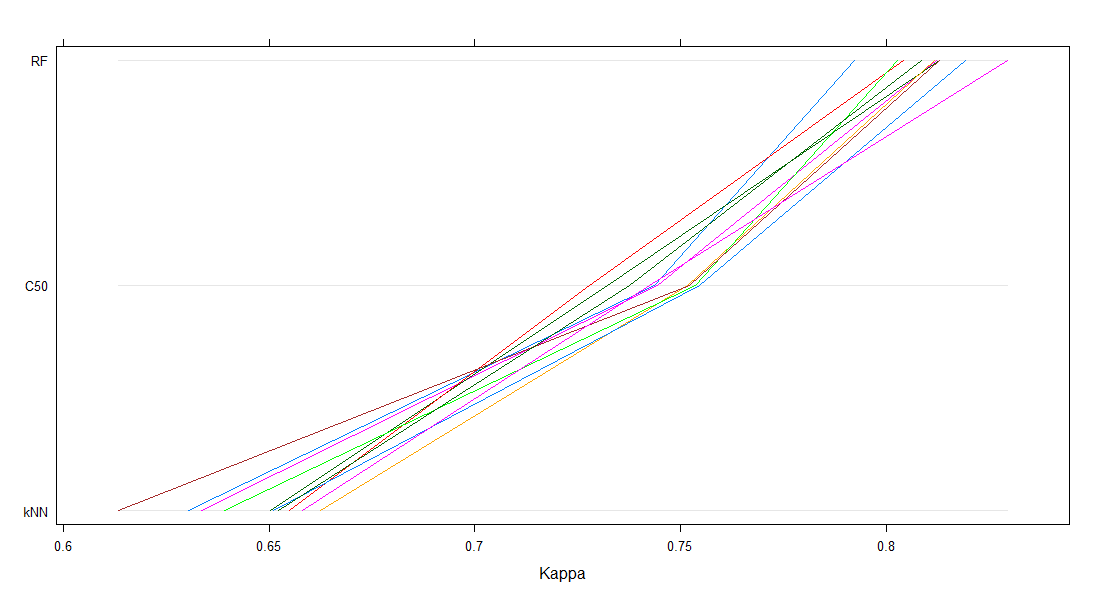
|  |  |  |  |
| --- | --- | --- | --- |
| Kappa  (diff) | C50 | RF | kNN |
| C50 |  | -0.06664 | 0.09951 |
| RF | 1.358e-07 |  | 0.16615 |
| kNN | 2.389e-07 | 1.825e-10 |  |

Kappa score difference comparison

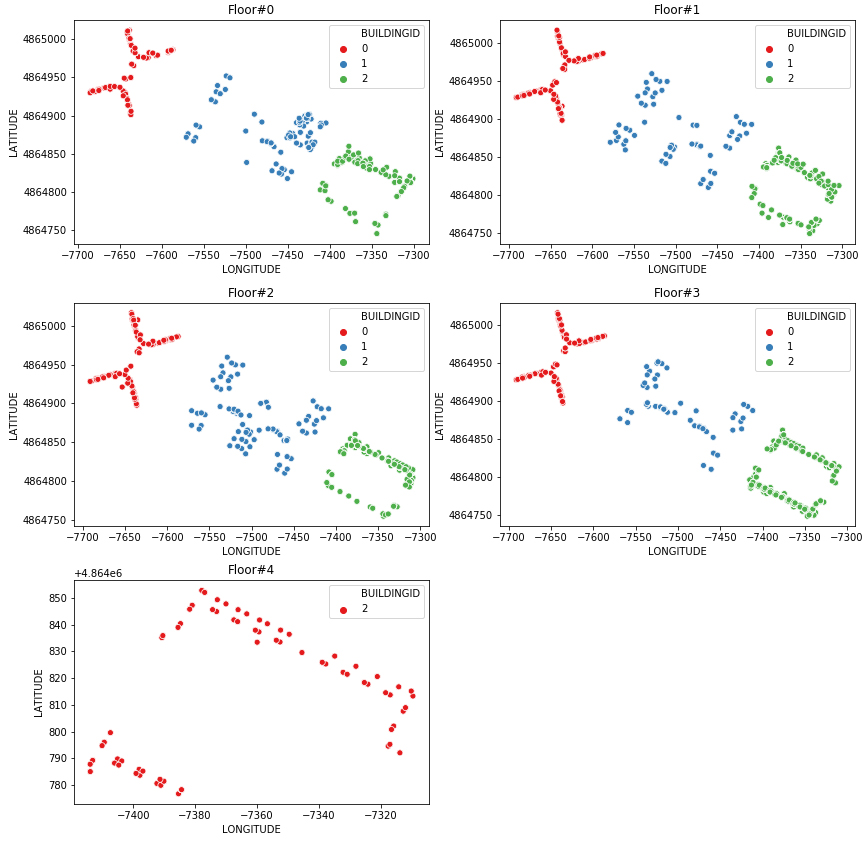


Additional visualizations

Kappa score comparison (parallel plot)



The locations of where the measurements were taken



Script

library(doParallel)

library(dplyr)

library(tidyr)

library(stringr)

library(caret)

# install.packages("devtools")

# library(devtools)

# install\_github("berndbischl/BBmisc")

# library(BBmisc)

registerDoParallel(cores=10)

setwd("C:\\Users\\ms-msi\\Desktop\\Data Scientist\\\_C3\\T3")

getwd()

df <- read.csv(file="./data/trainingData.csv")

str(df, list.len=ncol(df))

# is.na(df)

df$SPACEID <- as.factor(df$SPACEID)

df$FLOOR <- as.factor(df$FLOOR)

df$BUILDINGID <- as.factor(df$BUILDINGID)

df$RELATIVEPOSITION <- as.factor(df$RELATIVEPOSITION)

df$SPACEID <- str\_pad(df$SPACEID, 3, pad = "0")

# df\_smpl <- mutate(df, combined\_loc = paste(BUILDINGID,FLOOR,SPACEID, sep = ''))

df <- mutate(df, combined\_loc = paste(BUILDINGID,FLOOR,SPACEID,RELATIVEPOSITION, sep = ''))

# df\_smpl$SPACEID <- NULL

# df\_smpl$FLOOR <- NULL

# df\_smpl$BUILDINGID <- NULL

# df\_smpl$RELATIVEPOSITION <- NULL

col\_drop <- c('TIMESTAMP','PHONEID','USERID','LATITUDE','LONGITUDE')

#col\_drop <- c('TIMESTAMP','USERID','LATITUDE','LONGITUDE')

df$combined\_loc <- as.factor(df$combined\_loc)

df$SPACEID <- NULL

df$FLOOR <- NULL

df$BUILDINGID <- NULL

df$RELATIVEPOSITION <- NULL

# df\_smpl$combined\_loc <- as.factor(df\_smpl$combined\_loc)

# df\_smpl <- df\_smpl[sample(1:nrow(df\_smpl), nrow(df\_smpl),replace=FALSE),]

df$combined\_loc <- as.factor(df$combined\_loc)

# df <- df[sample(1:nrow(df), nrow(df),replace=FALSE),]

# col\_drop <- c('TIMESTAMP','PHONEID','USERID','LATITUDE','LONGITUDE')

col\_drop <- c('TIMESTAMP','USERID','LATITUDE','LONGITUDE')

# df\_loc\_smpl <- df\_smpl[ , !(names(df\_smpl) %in% col\_drop)]

# Delete columns with zero variance, i.e. constant

df\_loc <- df[ , !(names(df) %in% col\_drop)]

##

# df\_loc$PHONEID <- as.integer(df\_loc$PHONEID)

##

str(df\_loc, list.len=ncol(df\_loc))

# Delete columns with zero variance, i.e. constant

# which(apply(df\_loc, 2, var)==0)

# df\_loc\_smpl <- df\_loc\_smpl[ , apply(df\_loc\_smpl, 2, var) != 0]

df\_loc <- df\_loc[ , apply(df\_loc, 2, var) != 0]

# df\_loc[,1:465]

#df\_loc\_cs <- scale(df\_loc[, 1:465])

#colMeans(df\_loc\_cs)

#apply(df\_loc\_cs, 2, sd)

# df\_loc\_cs <- normalize(df\_loc[, 1:465], method = "standardize", range = c(-1, 1), margin = 2, on.constant = "quiet")

#hist(df\_loc\_cs$WAP005, xlim=c(-100,100), ylim=c(0,50))

# Create train and test sets (simple):

# partition\_s <- createDataPartition(df\_loc\_smpl$combined\_loc, p = .75, list = FALSE)

# training\_loc\_s <- df\_loc\_smpl[partition\_s,]

# testing\_loc\_s <- df\_loc\_smpl[-partition\_s,]

# Create train and test sets:

partition <- createDataPartition(df\_loc$combined\_loc, p = .75, list = FALSE)

training\_loc <- df\_loc[partition,]

testing\_loc <- df\_loc[-partition,]

#C5 model training for location prediction

fitControl0 <- trainControl(method = "repeatedcv", number = 10, repeats = 1)

C5Fit <- train(combined\_loc~., data = training\_loc, method = "C5.0", trControl=fitControl0, tuneLength = 3)

C5Fit0 <- train(combined\_loc~., data = training\_loc, method = "C5.0", trControl=fitControl0, tuneLength = 3)

C5Pred <- predict(C5Fit0, testing\_loc)

postResample(C5Pred, testing\_loc$combined\_loc)

#Random Forest model training for location prediction

rfFit <- train(combined\_loc~., data = training\_loc, method = "rf", trControl=fitControl0, tuneLength = 1)

rfFit\_test <- train(combined\_loc~., data = training\_loc, method = "rf", trControl=fitControl0, tuneLength = 1)

rfPred\_test <- predict(rfFit\_test, testing\_loc)

postResample(rfPred\_test, testing\_loc$combined\_loc)

#Random Forest model training for location prediction (simplified):

rfFit\_s <- train(combined\_loc~., data = training\_loc\_s, method = "rf", trControl=fitControl0, tuneLength = 1)

rfPred\_s <- predict(rfFit\_s, testing\_loc\_s)

postResample(rfPred\_s, testing\_loc\_s$combined\_loc)

str(testing\_loc, list.len=ncol(testing\_loc))

rfFit\_ph <- train(combined\_loc~., data = training\_loc, method = "rf", trControl=fitControl0, tuneLength = 1)

rfPred\_ph <- predict(rfFit\_ph, testing\_loc)

postResample(rfPred\_ph, testing\_loc$combined\_loc)

#Validation

postResample(rfPred, testing\_loc$combined\_loc)

#KNN model training for location prediction

fitControl\_knn <- trainControl(method = "repeatedcv", number = 10, repeats = 1)

#knnFit <- train(combined\_loc~., data = training\_loc, method = "knn", trControl=fitControl\_knn, preProcess = c("center","scale"), tuneLength = 10)

knnFit <- train(combined\_loc~., data = training\_loc, method = "knn", trControl=fitControl\_knn, preProcess = c("center","scale"), tuneGrid = expand.grid(k = 1:5))

knnPred <- predict(knnFit, testing\_loc)

postResample(knnPred, testing\_loc$combined\_loc)

# PCA

df.pr <- prcomp(df\_loc[c(1:465)], center = TRUE, scale = TRUE)

summary(df.pr)

df.pcst <- df.pr$x[,1:400]

df\_pca <- as.data.frame(df.pcst)

df\_pca <- cbind(df\_pca, df\_loc$combined\_loc)

colnames(df\_pca)[401] <- "combined\_loc"

str(df\_pca, list.len=ncol(df\_pca))

# Create train and test sets for PCA:

partitionPCA <- createDataPartition(df\_pca$combined\_loc, p = .75, list = FALSE)

trainPCA <- df\_pca[partitionPCA,]

testPCA <- df\_pca[-partitionPCA,]

rfFit\_PCA <- train(combined\_loc~., data = trainPCA, method = "rf", trControl=fitControl0, tuneLength = 1)

knnFit\_PCA <- train(combined\_loc~., data = trainPCA, method = "knn", trControl=fitControl0, tuneLength = 1)

### Use only GT series phones

str(df\_gt)

df\_gt <- read.csv(file="./data/trainingData.csv")

df\_gt <- filter(df\_gt, between(PHONEID, 1, 7))

df\_gt$SPACEID <- as.factor(df\_gt$SPACEID)

df\_gt$FLOOR <- as.factor(df\_gt$FLOOR)

df\_gt$BUILDINGID <- as.factor(df\_gt$BUILDINGID)

df\_gt$RELATIVEPOSITION <- as.factor(df\_gt$RELATIVEPOSITION)

df\_gt$SPACEID <- str\_pad(df\_gt$SPACEID, 3, pad = "0")

df\_gt <- mutate(df\_gt, combined\_gt = paste(BUILDINGID,FLOOR,SPACEID,RELATIVEPOSITION, sep = ''))

df\_gt$SPACEID <- NULL

df\_gt$FLOOR <- NULL

df\_gt$BUILDINGID <- NULL

df\_gt$RELATIVEPOSITION <- NULL

df\_gt$combined\_gt <- as.factor(df\_gt$combined\_gt)

df\_gt <- df\_gt[sample(1:nrow(df\_gt), nrow(df\_gt),replace=FALSE),]

df\_gt <- df\_gt[ , !(names(df\_gt) %in% col\_drop)]

df\_gt <- df\_gt[ , apply(df\_gt, 2, var) != 0]

summary(df\_gt)

partition\_gt <- createDataPartition(df\_gt$combined\_gt, p = .75, list = FALSE)

training\_gt <- df\_gt[partition\_gt,]

testing\_gt <- df\_gt[-partition\_gt,]

rfFit\_gt <- train(combined\_gt~., data = training\_gt, method = "rf", trControl=fitControl0, tuneLength = 3)

rfPred\_gt <- predict(rfFit\_gt, testing\_gt)

postResample(rfPred\_gt, testing\_gt$combined\_gt)

# GBM

registerDoSEQ()

gbmFit <- train(combined\_loc~., data = training\_loc, method = "gbm", trControl=fitControl0)

# Filter on building id

df\_b <- read.csv(file="./data/trainingData.csv")

df\_b0 <- filter(df\_b, BUILDINGID == "0" )

df\_b1 <- filter(df\_b, BUILDINGID == "1" )

df\_b2 <- filter(df\_b, BUILDINGID == "2" )

df\_b0 <- mutate(df\_b0, combined\_b = paste(BUILDINGID,FLOOR,SPACEID,RELATIVEPOSITION, sep = ''))

df\_b0$SPACEID <- NULL

df\_b0$FLOOR <- NULL

df\_b0$BUILDINGID <- NULL

df\_b0$RELATIVEPOSITION <- NULL

df\_b0$combined\_b <- as.factor(df\_b0$combined\_b)

df\_b0 <- df\_b0[ , apply(df\_b0, 2, var) != 0]

df\_b0 <- df\_b0[ , !(names(df\_b0) %in% col\_drop)]

df\_b1 <- mutate(df\_b1, combined\_b = paste(BUILDINGID,FLOOR,SPACEID,RELATIVEPOSITION, sep = ''))

df\_b1$SPACEID <- NULL

df\_b1$FLOOR <- NULL

df\_b1$BUILDINGID <- NULL

df\_b1$RELATIVEPOSITION <- NULL

df\_b1$combined\_b <- as.factor(df\_b1$combined\_b)

df\_b1 <- df\_b1[ , apply(df\_b1, 2, var) != 0]

df\_b1 <- df\_b1[ , !(names(df\_b1) %in% col\_drop)]

df\_b2 <- mutate(df\_b2, combined\_b = paste(BUILDINGID,FLOOR,SPACEID,RELATIVEPOSITION, sep = ''))

df\_b2$SPACEID <- NULL

df\_b2$FLOOR <- NULL

df\_b2$BUILDINGID <- NULL

df\_b2$RELATIVEPOSITION <- NULL

df\_b2$combined\_b <- as.factor(df\_b2$combined\_b)

df\_b2 <- df\_b2[ , apply(df\_b2, 2, var) != 0]

df\_b2 <- df\_b2[ , !(names(df\_b2) %in% col\_drop)]

partition0 <- createDataPartition(df\_b0$combined\_b, p = .75, list = FALSE)

train\_b0 <- df\_b0[partition0,]

test\_b0 <- df\_b0[-partition0,]

partition1 <- createDataPartition(df\_b1$combined\_b, p = .75, list = FALSE)

train\_b1 <- df\_b1[partition1,]

test\_b1 <- df\_b1[-partition1,]

partition2 <- createDataPartition(df\_b2$combined\_b, p = .75, list = FALSE)

train\_b2 <- df\_b2[partition2,]

test\_b2 <- df\_b2[-partition2,]

# gbmFit0 <- train(combined\_b~., data = train\_b0, method = "gbm", trControl=fitControl0, tuneLength = 1)

rfFit0 <- train(combined\_b~., data = train\_b0, method = "rf", trControl=fitControl0, tuneLength = 3)

rfPred\_b0 <- predict(rfFit0, test\_b0)

postResample(rfPred\_b0, test\_b0$combined\_b)

rfFit1 <- train(combined\_b~., data = train\_b1, method = "rf", trControl=fitControl0, tuneLength = 3)

rfPred\_b1 <- predict(rfFit1, test\_b1)

postResample(rfPred\_b1, test\_b1$combined\_b)

rfFit2 <- train(combined\_b~., data = train\_b2, method = "rf", trControl=fitControl0, tuneLength = 3)

rfPred\_b2 <- predict(rfFit2, test\_b2)

postResample(rfPred\_b2, test\_b2$combined\_b)

###

modeldata <- resamples(list(C50 = C5Fit0, RF = rfFit, kNN = knnFit))

summary(modeldata)

###

kappaDiffs <- diff(modeldata, metric = "Kappa")

summary(kappaDiffs)

parallelplot(modeldata, metric = "Kappa")

dotplot(modeldata, metric = "Kappa")

dotplot(kappaDiffs, metric = "Kappa")

splom(modeldata, metric = "Kappa")

varImp(rfFit)