Urban Road Accidents in Britain

Qi

2023-04-11

Data set available on website: https://archive.ics.uci.edu/ml/datasets/UrbanGB%2C+urban+road+accidents+coordinates+labelled+by+the+urban+center.

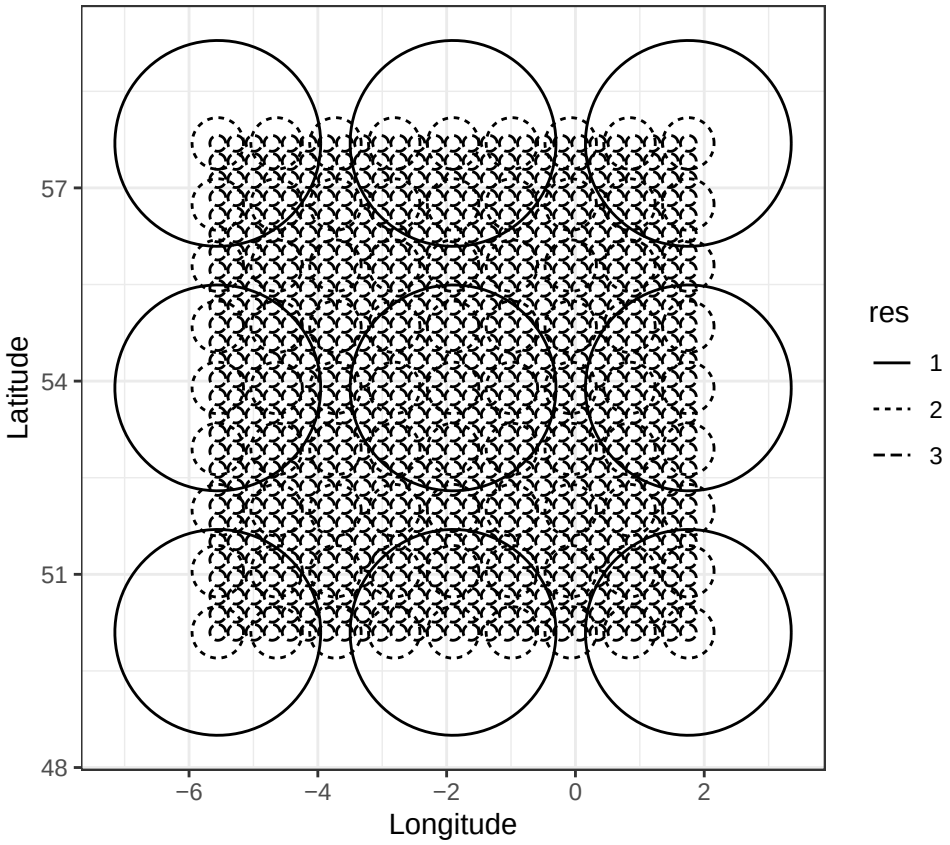
```
acc_loc <- read.table(here::here("urban/urbanGB.txt"), sep = ",")
acc_num <- read.table(here::here("urban/urbanGB.labels.txt"))

map_data <- map_data("world",region = "UK")
gb_data <- map_data[which(map_data$subregion == "Great Britain"),]
```

```
p1 <-
ggplot() +
  geom_path(aes(x = gb_data$long, y = gb_data$lat), color = "red", linewidth = 0.8) +
  labs(x = "Longitude", y = "Latitude") +
  geom_point(aes(x = acc_loc[,1], y = acc_loc[,2]), size = 0.5)

x.res <- 500
y.res <- 500
surf <- mba.surf(cbind(acc_loc, acc_num), no.X = x.res, no.Y = y.res, h = 5, m = 2, extend = TRUE)$xyz.
exp_grid <- expand.grid(surf$x, surf$y)
all_inside = spBayes::pointsInPoly(as.matrix(gb_data[,1:2]),as.matrix(exp_grid))


obs_acc <- surf$z[all_inside]
obs_long <-  as.matrix(exp_grid)[all_inside,1]
obs_lat <-  as.matrix(exp_grid)[all_inside,2]

p2 <-
ggplot() +
  geom_contour_filled(aes(x = obs_long, y = obs_lat, z = obs_acc))+
   geom_path(data = gb_data, aes(x = long, y = lat, group = group), color = "red")
```

```
dat_gb <- as.data.frame(cbind(acc_loc, acc_num))
colnames(dat_gb) <- c("long","lat","acc")
coordinates(dat_gb) <- ~ long + lat

gridbasis1 <- auto_basis(mainfold = plane(), data = dat_gb, nres = 1, type = "Gaussian", regular = 1)
gridbasis2 <- auto_basis(mainfold = plane(), data = dat_gb, nres = 2, type = "Gaussian", regular = 1)
gridbasis3 <- auto_basis(mainfold = plane(), data = dat_gb, nres = 3, type = "Gaussian", regular = 1)


show_basis(gridbasis3) +
```

```
  coord_fixed() +
  xlab("Longitude") +
  ylab("Latitude")
```

```
## Note: show_basis assumes spherical distance functions when plotting
```



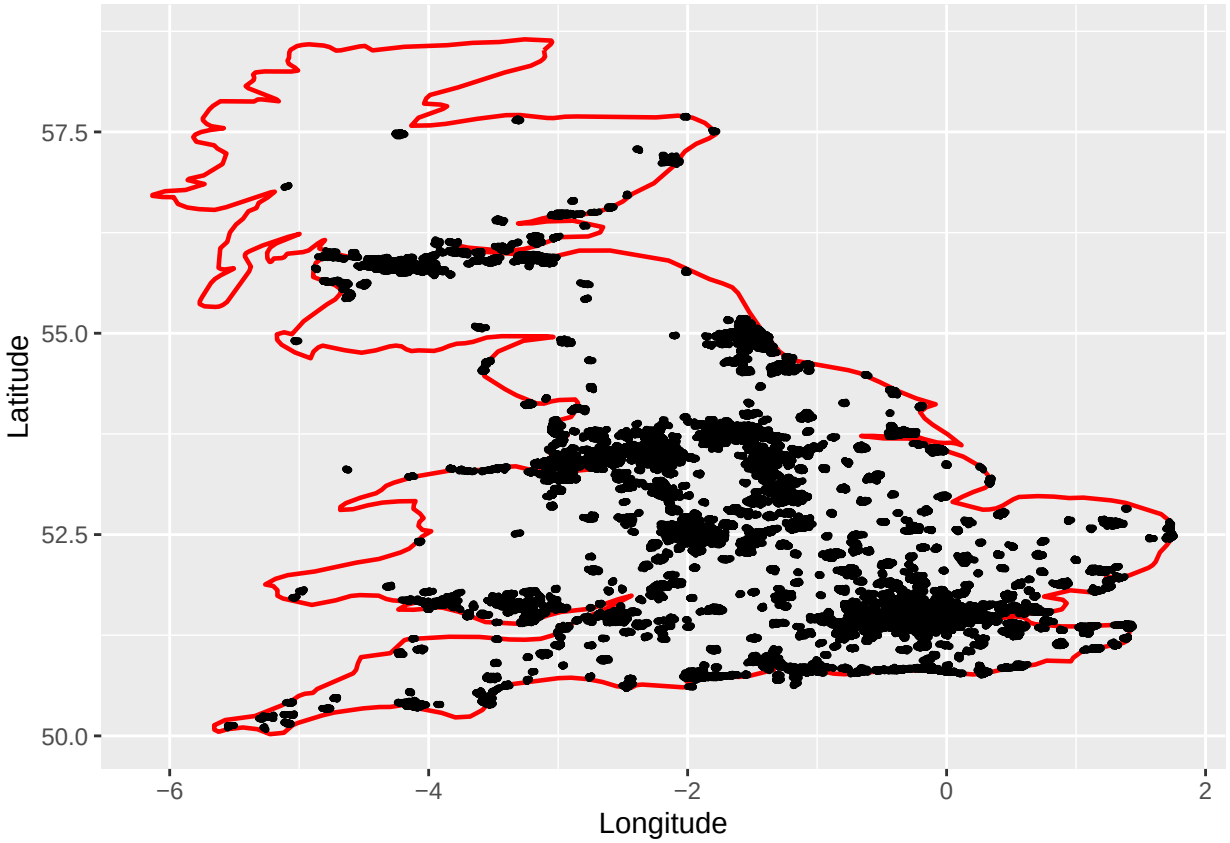

```
basis_1 <- matrix(NA, nrow = nrow(dat_gb), ncol = length(gridbasis1@fn))
for (i in 1:length(gridbasis1@fn)) {
  basis_1[,i] <- gridbasis1@fn[[i]](coordinates(dat_gb))
}

basis_2 <- matrix(NA, nrow = nrow(dat_gb), ncol = length(gridbasis2@fn))
for (i in 1:length(gridbasis2@fn)) {
  basis_2[,i] <- gridbasis2@fn[[i]](coordinates(dat_gb))
}

basis_3 <- matrix(NA, nrow = nrow(dat_gb), ncol = length(gridbasis3@fn))
for (i in 1:length(gridbasis3@fn)) {
  basis_3[,i] <- gridbasis3@fn[[i]](coordinates(dat_gb))
}


basis_use <- basis_3[,-(1:ncol(basis_2))]
```

```
depth <- 3
shape_row <- length(table(gridbasis3@df[which(gridbasis3@df$res == depth) , 2 ]))
shape_col <- length(table(gridbasis3@df[which(gridbasis3@df$res == depth) , 1 ]))
basis_arr <- array(NA, dim = c(nrow(dat_gb), shape_row, shape_col))
for (i in 1:nrow(dat_gb)) {
  basis_arr[i,,] <- matrix(basis_use[i,], nrow = shape_row, ncol = shape_col, byrow = T)
}
```

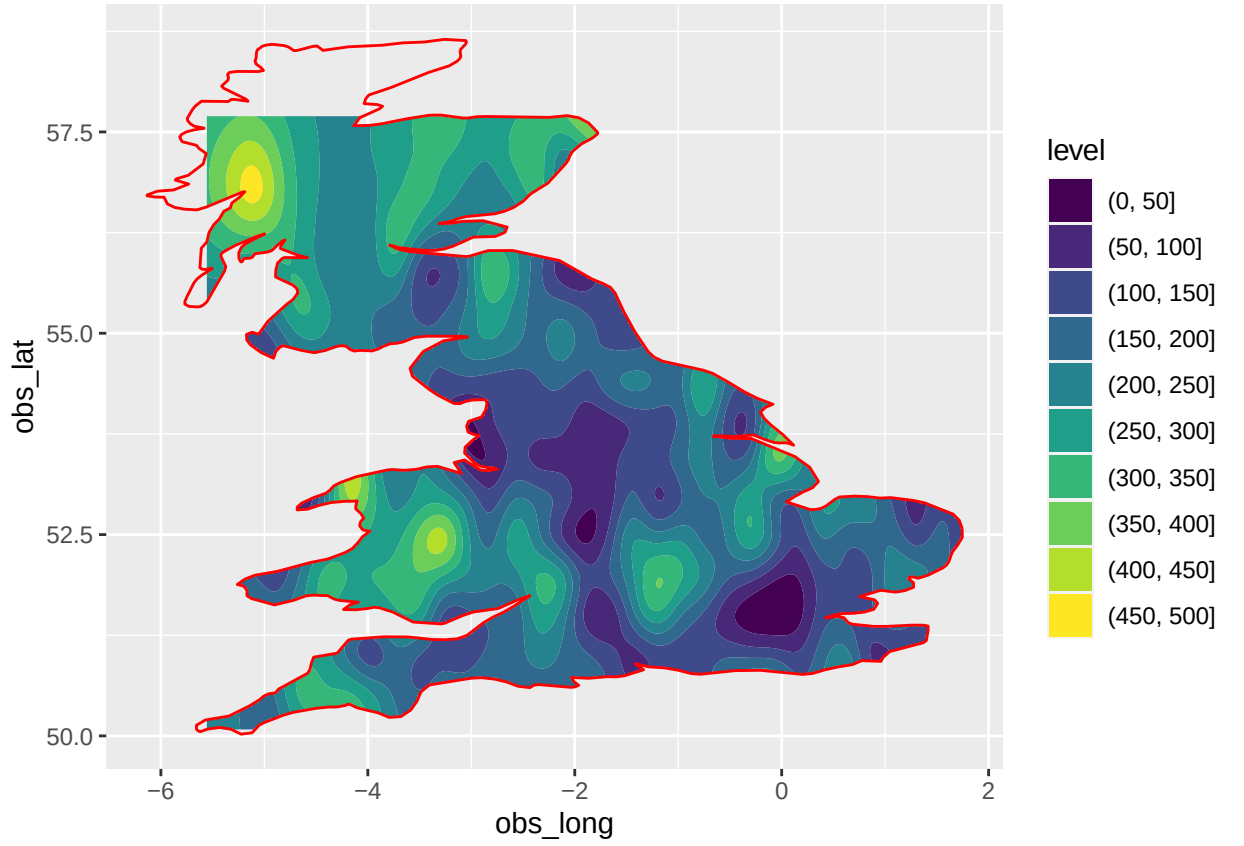
Deep Kriging Model

```
set.seed(0)
train_index <- sample(1:nrow(dat_gb), 300000, replace = FALSE)
```

```
basis_tr <- basis_arr[train_index,,]
basis_te <- basis_arr[-train_index,,]


x_tr <- array_reshape(basis_tr, c(nrow(basis_tr), shape_row*shape_col))  # So we want to reshape each o
x_te <- array_reshape(basis_te, c(nrow(basis_te), shape_row*shape_col))  # Same as prervious step

acc_tr <- acc_num[train_index,1]
acc_te <- acc_num[-train_index,1]


model_dk <- keras_model_sequential()

model_dk %>%
  layer_dense(units = 256, activation = 'relu', input_shape = c(ncol(x_tr))) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 1, activation = 'linear')

# Compile the model

model_dk %>% compile(
  loss = "mse",
  optimizer = optimizer_adam(),
  metrics = list("mse")
)

mod_train_dk <- model_dk %>%
  fit(x = x_tr, y = acc_tr, epochs = 30, batch_size = 128,
      validation_split = 0.2)

loss_dk <- model_dk %>%
           evaluate(x_te, acc_te)

p1
```




```
p2
```



Convolutional Kriging Model

```
# Define a few parameters to be used in the CNN model
batch_size <- 128
epochs <- 30

# Input image dimensions
img_rows <- shape_row
img_cols <- shape_col



x_tr <- array_reshape(basis_tr, c(nrow(basis_tr), img_rows, img_cols, 1))
x_te <- array_reshape(basis_te, c(nrow(basis_te), img_rows, img_cols, 1))
input_shape <- c(img_rows, img_cols, 1)




model_ck <- keras_model_sequential() %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu', input_shape = input_shape) %>%
  #layer_conv_2d(filters = 32, kernel_size = c(2,2), activation = 'relu') %>%
  #layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_dropout(rate = 0.1) %>%
  layer_flatten() %>%
  layer_dense(units = 256, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
```

```
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 1, activation = 'linear')


model_ck %>% compile(
  loss = "mse",
  optimizer = optimizer_adam(),
  metrics = list("mse")
)


mod_train_ck <- model_ck %>%
  fit(x = x_tr, y = acc_tr, epochs = 30, batch_size = 128,
      validation_split = 0.2)

loss_ck <- model_ck %>%
          evaluate(x_te, acc_te)
```

```
rbind(loss_dk, loss_ck)
```

```
##              loss       mse
## loss_dk 234.42836 234.42836
## loss_ck  92.80222  92.80222
```