OSM Code explained

PART - 1 - Converting the coordinates from polar to planar

Getting started

Setting the work directory

```
setwd("T:\\2.0 Pricing\\45. Machine learning\\OpenStreetMap")
```

Loading the following libraries

Let us load the following libraries

```
library(data.table)
library(spatstat)
library(rgdal)
library(rgeos)
library(sp)
library(raster)
library(foreign)
library(doParallel)
library(foreach)
library(rsai)
library(reshape2)
```

Reducing the dimension of .dbf files

The .dbf files are called DataBase Files. These are attribute files.The following code shrinks the file by reducing an unwanted column. This is done for increasing the processing time.

For Scotland

```
sp_dbf <- read.dbf(file=".\\Scotland\\gis_osm_pois_free_1.dbf", as.is = FALSE)
write.dbf(sp_dbf[,1:3], file=".\\Scotland\\gis_osm_pois_free_1.dbf")
```

For England

```
sp_dbf <- read.dbf(file=".\\England\\gis_osm_pois_free_1.dbf", as.is = FALSE)
write.dbf(sp_dbf[,1:3], file=".\\England\\gis_osm_pois_free_1.dbf")
```

For Wales

```
sp_dbf <- read.dbf(file=".\\Wales\\gis_osm_pois_free_1.dbf", as.is = FALSE)
write.dbf(sp_dbf[,1:3], file=".\\Wales\\gis_osm_pois_free_1.dbf")
```

For Ireland

```
sp_dbf <- read.dbf(file=".\\Ireland\\gis_osm_pois_free_1.dbf", as.is = FALSE)
write.dbf(sp_dbf[,1:3], file=".\\Ireland\\gis_osm_pois_free_1.dbf")
```

For Isle of Man

```
sp_dbf <- read.dbf(file=".\\IoM\\gis_osm_pois_free_1.dbf", as.is = FALSE)
write.dbf(sp_dbf[,1:3], file=".\\IoM\\gis_osm_pois_free_1.dbf")
```

Creating a Spatial data file

In this step we create a spatial data file which has UTM coordinate information for all the post codes in the United Kingdom. We perform this in multiple steps

> #### Step - 1

Check whether the R data (.RDA) file is already there in the work directory. If it is not there then we create that file in the directory by performing the following steps.

Step - 2

Load Post code data which is with lattitude and longitude information

Step - 3

Loading the point of interests shape files for England, Scotland, Wales, Isles of Man and Northern Ireland

Step - 4

In this step we convert the lattitude and longitdue from polar to planar coordinates (**UTM - Universal Transverse Mercator coordinates**). The lattitude and longitude are calculated by assuming the earth's shape to be ellipsoidal. But in UTM system the ellipsoidal shape is projected to a 2 dimensional plane and locations are identified using cartesian coordinate system.Also, For United Kingdom, plane reference comes under the Zone number 30.

Step - 5

Save this converted coordinates data of **Postcode** and **Point of interest** data into a R Data file in the work directory

```
  #STEP - 1
if (!file.exists('spatial_data_UK.RDA'))
{

  #STEP - 2
  colsToKeep <- c("PostcodeCompress","lat","long","imd")
  dt_ONSPD <-fread("Postcodes.csv", select = colsToKeep)

  #Renaming the PostcodeCompress column as pcds
  names(dt_ONSPD)[1]<-"pcds"

  #Creating a column for Post town
  dt_ONSPD$PostTown <- gsub("(^[A-Z]{1,2})(.*)","\\1",dt_ONSPD$pcds)

  #STEP - 3
  sp_POI_W <- shapefile("./Wales/gis_osm_pois_free_1.shp")
  sp_POI_E <- shapefile("./England/gis_osm_pois_free_1.shp")
  sp_POI_S <- shapefile("./Scotland/gis_osm_pois_free_1.shp")
  sp_POI_N <- shapefile("./Ireland/gis_osm_pois_free_1.shp")
  sp_POI_I <- shapefile("./IoM/gis_osm_pois_free_1.shp")

  #Combining all the shape files into one file
  sp_POI <- rbind(sp_POI_W, sp_POI_E, sp_POI_S, sp_POI_N, sp_POI_I)

  #STEP - 4
  postcode.spdf = SpatialPointsDataFrame(
    coords = dt_ONSPD[, c('long', 'lat')],
    proj4string = CRS("+proj=longlat +datum=WGS84"),
    data = dt_ONSPD)

  postcode.spdf = spTransform(postcode.spdf[postcode.spdf$lat < 99,], CRS("+proj=utm +zone=30 +north +ellps=
WGS84"))
  POI.spdf = spTransform(sp_POI, CRS("+proj=utm +zone=30 +north +ellps=WGS84"))

  #STEP - 5
  save(postcode.spdf, POI.spdf, file = 'spatial_data_UK.RDA')
}
```

After this step, let us load this saved R Data file to R

```
load(file = "spatial_data_UK.RDA")
```

The following spatial objects POI.spdf and postcode.spdf belong to the following class

```
## [1] "SpatialPointsDataFrame"
## attr(,"package")
## [1] "sp"
```

This is the plot of the spatial object POI.spdf, which produces the map of United Kingdom

Extracting the spatial points (points of interest) for each post town

In this step we extract the all the post towns

```
lst_postTown <- unique(postcode.spdf$PostTown)
```

In this block of code the following steps are performed

- The loop is iterated over all the post towns
- In a single iteration
    - All the spatial points of the postcodes in that particular post town is extracted
    - A spatial polygon which encloses all the above spatial points is created
    - The above spatial polygon is increased further by radius 20 KM
    - Finally the points that are common to enlarged spatial points(**posttownbuff.spdf**) and the points of interests from Open Street Map(**POI.spdf**) are extracted and saved

```
for(pt in lst_postTown[1:length(lst_postTown)]){
  cat(paste0("Creating data files for ", pt, "\n"))
  posttown.spdf = postcode.spdf[postcode.spdf$PostTown == pt,]
  posttown.extent <- as(extent(posttown.spdf), "SpatialPolygons")
  posttownbuff.spdf = gBuffer(posttown.extent, width = c(20000), byid = T)
  crs(posttownbuff.spdf) <- "+proj=utm +zone=30 +north +ellps=WGS84"

  POIbuff.spdf = intersect(POI.spdf, posttownbuff.spdf)

  save(posttown.spdf, POIbuff.spdf,
       file = paste0('./post_town data/spatial_data_', pt, '.RDA'))
  gc()
}
```

PART - 2 - Finding distances

In the part 2 of the code we calculate the distance between all the postcodes and the poinst of interests (building, pub, post office, post box, etc) which are placed within the radius of 20KM, 5KM and 1 KM

The following big chunk of code performs the above task. It consists of nested for loops, a new function holding nested if statements and a for loop in it.

```
#STEP - 1
lst_files<-list.files(path = "./post_town data")

#STEP - 2
for(lst_file in lst_files){
  cat(paste0("Calculating distances for ", lst_file, "\n"))
  pt <- gsub("spatial_data_", "",x = lst_file)
  pt <- gsub(".RDA", "",x = pt)
  load(paste0('./post_town data/spatial_data_', pt, '.RDA'))
  lst_postcode <- unique(posttown.spdf$pcds)

#STEP - 3
  lst_distances <- c(20000, 5000, 1000) #These are buffering distances of 20 km, 5km and 1 km
  get_distances <- function (i)
  {
    i_postcode <- lst_postcode[i]
    dt_pcd_dists <- data.table(pcds= character(),type = factor(),radius= numeric(),count=numeric(),distTo=nu
meric())

    # select the postcode
    point.spdf = posttown.spdf[posttown.spdf$pcds == i_postcode,]
    point.sp <- SpatialPoints(point.spdf@coords, CRS("+proj=utm +zone=30 +north +ellps=WGS84"))
    sp_POI_clipped <- POIbuff.spdf[gBuffer(point.sp, width = c(lst_distances[1]), byid = T), ]
    cat(paste0("\n",i_postcode," dist..."))

#STEP - 4
    if (length(sp_POI_clipped) > 0) {
      for (i_distance in lst_distances){
        cat(paste0(" ",i_distance,"..."))
        # create the buffer zone
        pointbuff.spdf = gBuffer(point.sp, width = c(i_distance), byid = T)
        #clip points to buffer
        points <- sum(!is.na(over(sp_POI_clipped, pointbuff.spdf)))

#STEP - 5
        if (points > 1){
          sp_POI_clipped <- intersect(sp_POI_clipped, pointbuff.spdf)
          if(is.null(sp_POI_clipped)) break
          count_points <-
            data.table(
              minDist = as.vector(gDistance(point.spdf, sp_POI_clipped, byid = T)),
              type = sp_POI_clipped@data$fclass
            )

#STEP - 6
          count_points$count <- rep(1,nrow(count_points))
          dt_pcd_summary <- count_points[,.(count= sum(count), distTo = min(minDist)), by= type]
          dt_pcd_summary$pcds <- i_postcode
          dt_pcd_summary$radius <- i_distance
          dt_pcd_dists <- rbind(dt_pcd_dists,dt_pcd_summary)
        }}
    }
    return(dt_pcd_dists)
  }

  no_cores <- detectCores()
  cl <- makeCluster(no_cores)
  registerDoParallel(cl)

#STEP - 7
  result <- foreach(i=1:length(lst_postcode), .combine = rbind, .packages=c("sp", "raster", "rgdal", "rgeos"
, "data.table")) %dopar% get_distances(i)
  stopImplicitCluster()
  save(result, file=paste0('T:\\2.0 Pricing\\45. Machine learning\\OpenStreetMap\\POI_dist\\POI_dists_', pt,
'.RDA'))
}
```

To understand the code above, let us break it and consider a simplest case: a single iteration

> ## Step - 1

We extract the names of the **.RDA** files saved for all the post towns in the PART - 1

```
lst_files<-list.files(path = "./post_town data")
head(lst_files,3)
```

```
## [1] "spatial_data_AB.RDA" "spatial_data_AL.RDA" "spatial_data_B.RDA"
```

Step - 2

The for loop begins here. It is iterated over all the posttown files (having spatial point details). For understanding purpose we take the 1st file name (1st iteration value). The post town name will be **AB**. We extract all the post code in the post town **AB**

```
#for(lst_file in lst_files){ #for loop begins here
  lst_file <- lst_files[1]
  pt <- gsub("spatial_data_", "",x = lst_file)
  pt <- gsub(".RDA", "",x = pt)
  load(paste0('./post_town data/spatial_data_', pt, '.RDA'))
  lst_postcode <- unique(posttown.spdf$pcds)

  head(lst_postcode)
```

```
## [1] "AB101QJ" "AB101SQ" "AB106AL" "AB106HR" "AB106HT" "AB106JU"
```

Step - 3

- Here we create a function called **get_distances** for finding distance between the post codes and the points of interests.
- The value of **i** is the value of all the post codes in the town.
- For better understanding, let us take the 1st post code in post town AB: **AB101QJ**
- We then extract the spatial point of the post code
- Then, we store all the **POIs** that are within 20 KM from the spatial coordinate of post code **AB101QJ**, in the variable **sp_POI_clipped**

```
lst_distances <- c(20000, 5000, 1000) #This vector stores the values of the radius of 20 km, 5km and 1 km
  #get_distances <- function (i)
  #{
    #i_postcode <- lst_postcode[i]

    i_postcode<-lst_postcode[1]
    dt_pcd_dists <- data.table(pcds= character(),type = factor(),radius= numeric(),count=numeric(), distTo=n
umeric()) #data table in which the final values are to be stored

    #Extracting the spatial point of the post code
    point.spdf = posttown.spdf[posttown.spdf$pcds == i_postcode,]
    point.sp <- SpatialPoints(point.spdf@coords, CRS("+proj=utm +zone=30 +north +ellps=WGS84"))

    #Including all the Points of interests which are within 20 KM from the post code point
    sp_POI_clipped <- POIbuff.spdf[gBuffer(point.sp, width = c(lst_distances[1]), byid = T), ]
```

Step - 4

- Now we create a loop within this, iterating over distance values: 20 KM, 5Km and 1KM
- For better understanding we take the **i_distance** value to be 5000 (5KM)
- In the variable **pointbuff.spdf** we include all the spatial points within the radius of 5KM from the post code spatial point
- Using the over function, we find the list of spatial points(**POIs**) in **sp_POI_clipped** that fall in 5KM radius from the post code

```
    #if (length(sp_POI_clipped) > 0) {
      #for (i_distance in lst_distances){

        i_distance <- lst_distances[2] #Setting distance value as 5KM

        # Including all the spatial points within the radius of 5KM
        pointbuff.spdf = gBuffer(point.sp, width = c(i_distance), byid = T)

        #Checks the number of points common to  within the radius of 20KM and 5KM
        points <- sum(!is.na(over(sp_POI_clipped, pointbuff.spdf)))
        points
```

```
## [1] 1314
```

Step - 5

- From the above step, if the number of points>0 then we use those points (**POIs**) to find the distance
- Using **gDistance** function we find the distance between the coordinate of **AB101QJ** - postcode and all the POIs within the radius of 5KM
- In the data table we have declared the **fclass** variable as **type** variable - which is the type of POIs (Building, pub, school, etc)

```
#if (points > 1){
          sp_POI_clipped <- intersect(sp_POI_clipped, pointbuff.spdf)
          if(is.null(sp_POI_clipped)) break #If there are no points then we break the loop
          count_points <-
            data.table(
              minDist = as.vector(gDistance(point.spdf, sp_POI_clipped, byid = T)),
              type = sp_POI_clipped@data$fclass
            )
```

Step - 6

- In this step, we find the minimum distance between the POIs and the coordinate of **AB101QJ**, grouped by POIs
- The resultant data set which looks like the below output is returned as a result of this function

```
          count_points$count <- rep(1,nrow(count_points))
          dt_pcd_summary <- count_points[,.(count= sum(count), distTo = min(minDist)), by= type]
          dt_pcd_summary$pcds <- i_postcode
          dt_pcd_summary$radius <- i_distance
          dt_pcd_dists <- rbind(dt_pcd_dists,dt_pcd_summary)
          head(dt_pcd_dists,10)
```

```
##        pcds          type radius count    distTo
##  1: AB101QJ      post_box   5000   173  102.9786
##  2: AB101QJ   supermarket   5000    21  285.1082
##  3: AB101QJ sports_centre   5000     2 3463.5851
##  4: AB101QJ       library   5000     7  384.0788
##  5: AB101QJ          bank   5000    26  340.1345
##  6: AB101QJ           pub   5000    85  252.8376
##  7: AB101QJ        police   5000     3  828.0585
##  8: AB101QJ    lighthouse   5000     4 2225.9141
##  9: AB101QJ        ruins   5000     2 3226.3440
## 10: AB101QJ      monument   5000     8  171.8889
```

```
        #}}
    #}
    #return(dt_pcd_dists)
  #}
```

Step - 7

- The above function **get_distances** is invoked by calling it for the radii values 20KM, 5KM and 1KM
- This loop is iterated over all the post town values

- The outcome values are saved individually for different post towns

```
  result <- foreach(i=1:length(lst_postcode), .combine = rbind, .packages=c("sp", "raster", "rgdal", "rgeos"
, "data.table")) %dopar% get_distances(i)
  save(result, file=paste0('T:\\2.0 Pricing\\45. Machine learning\\OpenStreetMap\\POI_dist\\POI_dists_', pt,
'.RDA'))
#} - For loop ends here
```

In the end of 7 steps in PART - 2, we found: * For each **Posttowns**: + For each **Postcodes** in them: The distance between the postcodes and POIs that are within the radius of: 1. 20 KM 2. 5 KM 3. 1 KM

PART - 3 - Final outcome with features for all the post towns

In this step, we produce the final outcome in which all the post codes will have 1. Number of POIs (hospital, post office, post box, school) from the coordinates of the postcode, within the radius of 1 KM, 5 KM and 20 KM 2. Closest distance of POIs from the coordinates of the postcode

First we extract the names all the distance files creted for each post towns from PART - 2

```
#identify list of data files
lst_files  <- list.files("./POI_dist/")
```

We read the unique values of Types of POIs and set the key as type

```
#read in mapping file for POI Types
dt_POITypeMap <- fread("./Callum's Code/POI_Type.csv")
setkey(dt_POITypeMap, type)
```

The **For loop** below creates the final outcome with features for different post towns. It is iterated over the distance file names created in PART - 2

```
for(lst_file in lst_files){

  cat(paste0("Processing file ", lst_file, "... \n"))

  pt <- gsub("POI_dists_", "",x = lst_file)
  pt <- gsub(".RDA", "",x = pt)

  # load in data file and setkey for merge
  load(paste0('./POI_dist/', lst_file))
  setkey(result, type)

  # merge in summarised POITypes and remove original
  result <- merge(result, dt_POITypeMap, all.x = TRUE)

  #create datatables that sum counts and calc min dists to
  result_count <- result[, .(count = sum(count)), by = .(pcds, radius, type)]
  result_dist <- result[, .(minDist = min(distTo)), by = .(pcds, type)]

  #now turn from long into wide datasets using dcast
  #dcast formula LHS ~ RHS ; LHS = Id variables,  RHS = var to transpose to columns
  result_pcd <- data.table(dcast(result_count, formula = pcds ~ type + radius, value.var = "count"))
  result_pcd2 <- data.table(dcast(result_dist, formula = pcds ~ type , value.var = "minDist"))

  # tidy up the names of the minDist file
  names(result_pcd2)[-1] <- paste0(names(result_pcd2)[-1], "_minDistTo")

  #replace NA with 0 for length and 20000 for minDist
  for (col in 1:ncol(result_pcd)) set(result_pcd, which(is.na(result_pcd[[col]])), col, 0)
  for (col in 1:ncol(result_pcd2)) set(result_pcd2, which(is.na(result_pcd2[[col]])), col, 20000)

  #now merge the 2 files to create the output
  setkey(result_pcd, pcds)
  setkey(result_pcd2, pcds)

  output <- merge(result_pcd, result_pcd2, all.x = TRUE)

  for (col in 1:334) set(output, which(is.na(output[[col]])), col, 0)
  for (col in 1:334) set(output, which((output[[col]]) == 20000), col, 0)

  for (col in 335:459) set(output, which(is.na(output[[col]])), col, 20000)
  for (col in 335:459) set(output, which((output[[col]]) == 0 ), col, 20000)

  write.csv(output, paste0('./POI_Distance_File_', pt, '.csv'), row.names = FALSE)
}
```

Let us break down the above For Loop for 1 iteration

Step - 1

The for loop begins here. But for better understanding, we take a distance for the post town **BA**

```
#for(lst_file in lst_files){
  lst_file <- lst_files[4]
  pt <- gsub("POI_dists_", "",x = lst_file)
  pt <- gsub(".RDA", "",x = pt)
```

Step - 2

- In this step we load the distance file for town BA
- We set the type column (types of POIs) as the key
- Merge it with the **dt_POITypeMap** variable which has the unique values of the POIs types

```r
# load in data file and setkey for merge
load(paste0('./POI_dist/', lst_file))
setkey(result, type)

# merge in summarised POITypes
result <- merge(result, dt_POITypeMap, all.x = TRUE)
```

Step - 3

- Firstly, we count the number of unique values of post codes,radius and type grouped together
- Secondly, we calculate the minimum distance of a post code and a POI type

```r
#create datatables that sum counts and calc min dists to
result_count <- result[, .(count = sum(count)), by = .(pcds, radius, type)]
result_dist <- result[, .(minDist = min(distTo)), by = .(pcds, type)]
```

Step - 4

In this step we transpose the **result_count** data set and **result_dist** from long to a wide format in the following way and we clean the names

```r
#now turn from long into wide datasets using dcast
#dcast formula LHS ~ RHS ; LHS = Id variables,  RHS = var to transpose to columns
result_pcd <- data.table(dcast(result_count, formula = pcds ~ type + radius, value.var = "count"))
result_pcd2 <- data.table(dcast(result_dist, formula = pcds ~ type , value.var = "minDist"))

head(result_pcd[,c(1:4)])
```

```
##       pcds archaeological_1000 archaeological_5000 archaeological_20000
## 1: BA100DD                  NA                  NA                   17
## 2: BA111DH                  NA                   1                   53
## 3: BA111DW                  NA                   1                   53
## 4: BA111JS                  NA                   1                   53
## 5: BA111LT                  NA                   1                   53
## 6: BA111PN                  NA                   1                   53
```

```r
head(result_pcd2[,c(1:4)])
```

```
##       pcds archaeological arts_centre  artwork
## 1: BA100DD       6812.097          NA 9557.498
## 2: BA111DH       3948.479          NA 7141.178
## 3: BA111DW       3904.642          NA 7109.035
## 4: BA111JS       4117.049          NA 6719.038
## 5: BA111LT       4300.577          NA 6659.015
## 6: BA111PN       4459.442          NA 6603.077
```

```r
# tidy up the names of the minDist file
names(result_pcd2)[-1] <- paste0(names(result_pcd2)[-1], "_minDistTo")
```

Step - 5

Here if there are missing values in result_pcd and result_pcd2 then we fill them with 0 and 20000 respectively. If the count column has missing value then it is converted to 0 which means there is no count. Similarly, if we have missing values in the distance columns then it is converted to 20000, which means the minimum distance is 20 KM from the post code to that particular POI

```r
#replace NA with 0 for length and 20000 for minDist
for (col in 1:ncol(result_pcd)) set(result_pcd, which(is.na(result_pcd[[col]])), col, 0)
for (col in 1:ncol(result_pcd2)) set(result_pcd2, which(is.na(result_pcd2[[col]])), col, 20000)
```

Step - 6

We now merge both the dsitance and the count features by post code and write the output for the post town AB. The for loop ends here

```
  #now merge the 2 files to create the output
  setkey(result_pcd, pcds)
  setkey(result_pcd2, pcds)

  output <- merge(result_pcd, result_pcd2, all.x = TRUE)

  for (col in 1:334) set(output, which(is.na(output[[col]])), col, 0)
  for (col in 1:334) set(output, which((output[[col]]) == 20000), col, 0)

  for (col in 335:459) set(output, which(is.na(output[[col]])), col, 20000)
  for (col in 335:459) set(output, which((output[[col]]) == 0 ), col, 20000)

  write.csv(output, paste0('./POI_Distance_File_', pt, '.csv'), row.names = FALSE)
#} The for loop ends here
```