NATIONAL UNIVERSITY OF SINGAPORE

DATA SCIENCE COMPETITION 2018

FINDING LOCATIONS & MACHINE LEARNING

Group Number: 24

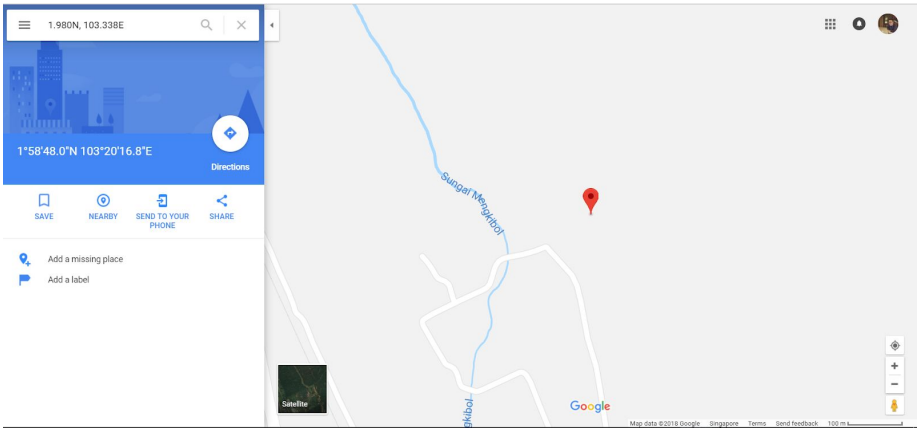
Group Members:

Liu Song Yu James (NUS A0155148A)
Kaustubh Jagtap (NUS A0168820B)
Richard Kang Hwee Chiat(NUS A0158427X)
Kang Hwee Young (NTU U1640393D)

Task 1: Locating the 50 weather stations:

First step: Narrowing our points

1. We visualised the top left coordinate of our area of interest (1.980N, 103.338E). It was found to be in Malaysia.



2. To narrow the data that we are concerned with, we calculated a rough parameter that contains Singapore, in terms of latitude and longitude. This will allow us to figure out the “x” and “y” boundary that we have to work with.

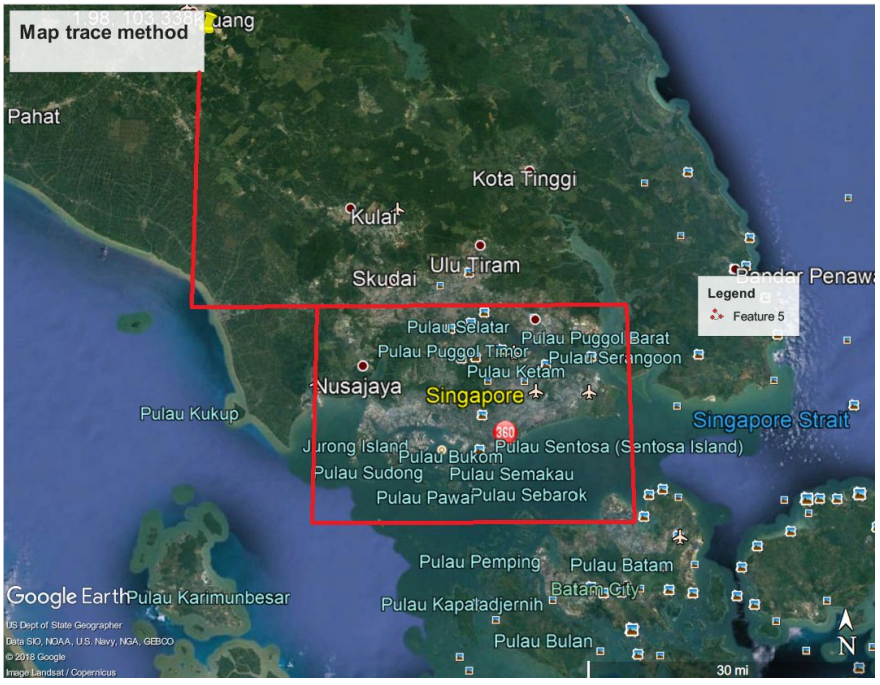


Figure 1: Narrowing down to the size of Singapore to reduce search complexity

The latitudes and longitudes of Singapore:

Point	Latitude	Longitude
Upper Left	1.476788N	103.595061E
Upper right	1.476788N	104.046259E
Lower Left	1.236016N	103.595061E
Lower Right	1.236016N	104.046259E

3. The reason for narrowing our points was because we wanted to reduce the time and space complexity that our algorithm would have to be run in, allowing us to compute the answer faster as we would have to search through lesser points.
4. Also, narrowing the points we have to work with will allow us to better proceed with our next steps.
5. Using these 4 points, we computed the x-y coordinates on our imaginary plane that hold Singapore.
6. Note: We can afford to do this, without compromising any vital data, since all weather stations are located in Singapore, but the backscatter plot shows data for a larger radius.
7. We cross-checked our x-y coordinates that hold Singapore, by using distance calculator on Google Earth.
8. The x-y coordinates on our grid that hold Singapore were found to be:

80 <= x <= 270
181 <= y <= 314

- We combined all our data frames for the radar data, for all the weeks given.
- From here, since we know the fictional x-y coordinates for all our radar, data, we were able to filter out those coordinates that did not lie in Singapore
- The resulting data frame was radar data for all points in that lie only within Singapore.

Second Step:

1. Our next step in locating the 50 stations involved narrowing from the 25270 grids (Singapore) down to just 50 grids, which contain the 50 stations.

2. We followed the equation 1 to assign a "score" or total amount of rainfall to each week's reading from the data.

$$R_{radar}(x\ ,\ y)=\sum A\ e_{Bk}\ c_k\ (x\ ,\ y)\Delta\ t \quad \text{--Equation 1}$$

We assumed A to be 0.079, B to be 0.228 as given from the slides, Ck to be the number of backscatters for bin k, x and y to be lat and long values and delta T to be 1 week.

Before proceeding, we handled the NA values in the gauge dataset by median imputation for each bin as using mean imputation will indirectly affect the correlation of our score and the measured rainfall by changing the standard deviation.

Upon getting the scores and adding it to our dataframe, we ran two for loops to check correlations of each location (0,1,2,3,...49) in gauge data for all 31 weeks vs each location (long,lat) in the "narrowed" data for all 31 weeks.

Our algorithm then extracted the top data point for each weather station location that showed showed the highest correlation. In fact, our average correlation was 0.714, which we consider high. Our highest correlation was 0.89 with our lowest being 0.54.

Figure 2: Maxes representing a array containing the 50 highest correlations

Now, we have the 50 long-lat coordinate points which represent our rain gauges locations. Noting that this is just our estimate, we checked the Singapore Climatological Report 2016, for actual Rain Gauge locations in Singapore. We then ran an algorithm to match the every estimated point to the nearest actual Singapore Rain Gauge (by minimizing Euclidean distance), and assigned that lat-long coordinate to the rain gauge location. This ensures that our rain gauge location falls exactly at a point where there is an actual weather station in Singapore.

Results:

We have attached with our submission the results of these findings, in a csv file called Team 24 with 3 columns - Gauge No., Long (x) and Lat (y).

Task 2: Calibrating the radar data:

Imputed median for missing values - we noticed there were several missing data points in the given data. Hence, we imputed the missing values by filling in the medians.

Henceforth, for calculating the risk function, the below formula will be used:

$$\frac{1}{T}\sum_{k}(R_r - R_g)^2$$

Where Rr is the rainfall from radar readings and Rg is the actual rainfall from gauge readings. We approached the problem from 2 different angles - Random Forest and Particle Swarm Optimizer. Below is a brief overview of our procedures:

Particle Swarm Optimizer:

The reading for each of the rows was calculated using the following function:

$$\sum_{k} Ae^{Bk}c_k(x,y)\Delta t + E$$

From this, the loss function was calculated from the square of the difference between the actual gauge reading and the sum of the exponentiation function applied to each cell. The final values were calculated as such:

A	B	E	Loss (MSE)	Test error (MSE)
1.937	0.371	0.825	720.9	645.46

Snippets of optimizer running

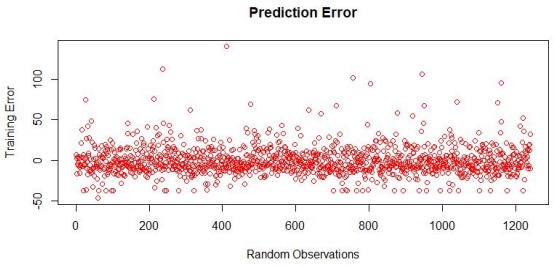
```
C:\Users\ksjag\Documents\Projects\NUS Data Science Competition 2018\datasets
λ python optimizer2.py
split done
starting optimiser
Stopping search: Swarm best position change less than 1e-05
A: 1.936027122520014 B: 0.31655633990380605 E: 0.8246525022964448
Loss: 720.9125160784913
```

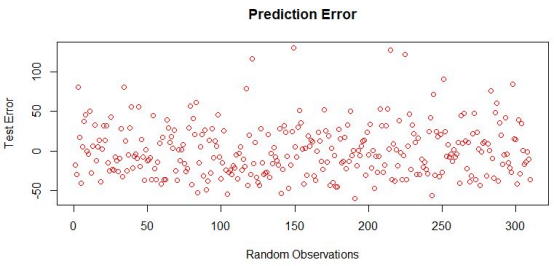
We can conclude that our results are decent, but could be higher in accuracy. This result should be interpreted along with the knowledge that we were in the blind about the actual locations of the rain gauges.

Random Forest:

We considered that we might not be able to fit a function to our data and decided to use a non-parametric machine learning method to predict rainfall. Thus, we used Random Forest algorithm to predict rainfall. For this algorithm, we split the data to 80%-20% for training and test. We also used 10-fold cross validation to find the suitable parameters and variables. Random forest is performed by firstly generating many sets of observations by sampling the original dataset. We then generate regression trees via recursive binary splitting on each dataset and aggregating the trees. However, the key is that we only choose a subset of predictors to be considered for each split in order to reduce chances of overfitting.

Training Error (MSE): 137.8314
Test Error (MSE): 512.5185





Brute Force Approach

Since there are only two variables(A,B) to tune in the original given model, we decided to use a brute force approach by iteratively computing the risk(MSE) over possible values of A and B to find the minimizing A and B. By testing over range 0 to 4 for A with step 0.1 and

range 0 to 2 for B with step 0.1, we managed to achieve MSE of 1165 for A=2.7 and B=0.3. This is computationally intensive and might not be able to find the global minimum.

Task 3: calculate the Euclidean distance of the calibrated radar readings from the gauge data

The euclidean distance is calculated using the following formula.

$$\sqrt{\sum (R_{radar} - R_{gauge})^2}$$

Euclidean Distance: 446.2166

Conclusion

Given the constraint of time, we took a preliminary approach to our 3 model designs which includes parametric and non-parametric models. The most robust model out of the 3 is the Random Forest approach. It stands out the most because the MSE is the lowest out of all 3 and we managed to obtain consistent errors from the cross validation.

We also realised that solving the 50 locations was an important part of the task because we had to fit the correct location to the correct radar data in order to get the correct values of the bins, allowing us to use the most appropriate values for our machine learning model.

The uncertainty of the locations also played a part in our errors. This is because we were only able to **guess** the locations based on correlation of 0.714. Hence, the final error in our model is a cumulation of the errors in locating the gauges, combined with the fit of the model.

Appendix

Correlation Algorithm for location ranking

```
def check_location(df_radar, df_gauge):
    number=set(df_radar['week'])
    for i in range(2,52):

        highest_approx_corr=[]
        highest_approx_lat_long=[]
        counter=0
        #gets rainfall over all the 31 weeks
        rainfall_list=list(df_gauge.iloc[:,i])


        for j in range(0, len(df_radar), 31):

            list_of=list(df_radar['sum'][j:j+31])
            #does correlation with predicted rainfall and actual rainfall for
            #possible combinations
            corr=list(pearsonr(rainfall_list,list_of))[0]

            lat=df_radar['lat'][j:j+31][j]
            longg=df_radar['long'][j:j+31][j]
            lat_long=list((lat,longg))
            highest_approx_corr.append(corr)
            highest_approx_lat_long.append(lat_long)
        #appends all the corr values to each location
        empty_list_corr.append(highest_approx_corr)
        empty_list_ll.append(highest_approx_lat_long)
    check_max_more(empty_list_corr,empty_list_ll)
```

Pyswarm Optimizer

```
from pyswarm import pso
from math import *
import csv
from sklearn.model_selection import train_test_split
from random import uniform

def read_csv(csvfilename):
    rows = []
    with open(csvfilename) as csvfile:
        file_reader = csv.reader(csvfile)
        for row in file_reader:
            rows.append(row)
    return rows

data = read_csv("finalz_vz.csv")
data = data[1:]

def split_data():       # outputs tuple of 2 arrays, train and test
    # break into train and test split
    booleans = []       # generate 80% 1s and 20% 0s
    for i in range(1550):
        n = uniform(0,1)
        if n<0.8:
            booleans.append(1)
        else:
            booleans.append(0)

    train, test = [], []
    for i in range(len(booleans)):
        if booleans[i] == 1:
            train.append(data[i])
        else:
            test.append(data[i])

    return train, test

train, test = split_data()
print("split done")

```

```python
def loss_fn(params):
    # params_array = [A, B, E]
    A = params[0]
    B = params[1]
    E = params[2]

    delta_t = 5/(60*24*7)

    # function to transform one cell one value, given the number and bin no
    f = lambda ck, k: (A * exp(B*k) * int(ck) * delta_t) + E

    def sum_row(row):   # gets score for entire row
        res = 0
        for i in range(6, 38):
            bin_num = i-5
            cell_val = row[i]
            res += f(cell_val, bin_num)
        return res

    loss = 0
    for row in train:
        actual = float(row[1])
        R = sum_row(row)
        diff = (actual - R) ** 2

        loss += diff

    return loss/len(train)

lower_bounds = [7, 0, 0]
upper_bounds = [10, 0.4, 2]

print("starting optimiser")
params_opt, function_opt = pso(loss_fn, lower_bounds, upper_bounds, minstep=1e-5, maxiter=400)

A = params_opt[0]
B = params_opt[1]
E = params_opt[2]

print("A:", A, "B:", B, "E:", E)
print("Loss:", function_opt)
```

```
def test_fn(A, B, E):
    delta_t = 5/(60*24*7)

    # function to transform one cell one value, given the number and bin no
    f = lambda cell_val, bin_num: A * exp(B*bin_num) * int(cell_val) * delta_t + E

    def sum_row(row):   # gets score for entire row
        res = 0
        for i in range(6, 38):
            bin_num = i-5
            cell_val = row[i]
            res += f(cell_val, bin_num)
        return res

    actuals = []
    predicts = []

    for row in test:
        actuals.append(float(row[1]))
        R = sum_row(row)
        predicts.append(R)

    return actuals, predicts

actuals, predicts = test_fn(A,B,E)
test_error = 0
for i in range(len(actuals)):
    actual = actuals[i]
    predicted = round(predicts[i], 1)
    print(actual, "==", predicted, "Error:", abs(actual-predicted))
    test_error = test_error + (actual - predicted)**2

print ("Test Error:", round(test_error/len(test), 2))
```

Median Imputation

```
data_guage_transformed = data_gauge

for (j in 3:ncol(data_guage_transformed))
{
  median_impute=median(data_guage_transformed[,j],na.rm=TRUE)

  for (i in 1:nrow(data_guage_transformed))
  {
    if(is.na(data_guage_transformed[i,j]))
    {
      data_guage_transformed[i,j]=median_impute
    }
  }
}

write.csv(data_guage_transformed,
          file='data_guage_transformed_median.csv',
          row.names=FALSE)
```

Brute Force Approach

```
my_function = function(X)
{
  A=X[1]
  B=X[2]
  L=c()
  R=c()
  Gauge=final[,2]
  for (i in 1:nrow(final))
  {
    C=final[i,6:38]
    sum=0
    for (k in 1:33)
    {
      sum=sum+A*exp(B*k)*as.numeric(C[k])*(5/(60*24*7))
      #sum=sum+A*exp(B*k)*as.numeric(C[k])
    }
    R[i]=sum
  }
  L=(R-Gauge)^2
  risk=(1/1550)*sum(L)

  return(risk)
}
```

```
min=10000
for (a in seq(from = 2.3, to = 10, by =0.1))
{
  for (b in seq(from = 0, to = 4, by =0.1))
  {
    if (my_function(c(a,b))<min)
    {
      min=my_function(c(a,b))
      A_BEST=a
      B_BEST=b
    }

  }
}
```

Random Forest

```
library(randomForest) # random forest
library(caret)

shuffle <- sample(nrow(final)) # shuffle data
final <- final[shuffle, ]

# Validation Set
pseudo_train_rows = sample(nrow(final),0.8*nrow(final))
pseudo_train = final[pseudo_train_rows,]
pseudo_test = final[-pseudo_train_rows,]


##### Random Forest
rf.reg <- randomForest(gauge_value ~ .,
                       data = pseudo_train[,-c(1,3,4,5,39)],
                       mtry = 6,
                       ntree = 1000,
                       replace=TRUE,
                       importance = TRUE)

# Predict
pred.rf <- predict(rf.reg, newdata = pseudo_test, type = "response")

(sum((pred.rf-pseudo_test[,2])^2))/nrow(pseudo_test)
(sum((pred.rf-pseudo_train[,2])^2))/nrow(pseudo_train)
```

10-fold cross validation

```
K = 10
folds = cut(seq(1,nrow(final)), breaks = K, labels = FALSE)

# Perform 10 fold cross validation
cost = matrix(0, K, 1)
colnames(cost) = c("cv error")
for (k in 1:K)
{
  # Divide our data
  test = which(folds==k) # Gives index of the k-th fold
  final.test = final[test,] # Assigns k-th fold as test set
  final.train = final[-test,] # Assign all other as training set

  # Model Training
  rf.reg <- randomForest(gauge_value ~ .,
                         data = final.train[,-c(1,3,4,5,6,7,39)],
                         mtry = 5,
                         ntree = 1000,
                         replace=TRUE,
                         importance = FALSE)

  # Prediction
  rf.pred = predict(rf.reg, newdata = final.test)

  # error
  cost[k, 1] = sum((final.test$gauge_value - rf.pred)^2)/nrow(final.test)
}
```