machine learning(732A99) lab3

Anubhav Dikshit(anudi287)

17 December 2018

Contents

Loading The Libraries

```
if (!require("pacman")) install.packages("pacman")
pacman::p_load(geosphere, kernlab, geosphere, ggplot2,caret)

set.seed(12345)
options("jtools-digits" = 2, scipen = 999)

# colours (colour blind friendly)
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2",
               "#D55E00", "#CC79A7")
```

Assignment 1

1. Implement a kernel method to predict the hourly temperatures for a date and place in Sweden.To do so, you are provided with the files stations.csv and temps50k.csv. These files contain information about weather stations and temperature measurements in the stations at different days and times. The data have been kindly provided by the Swedish Meteorological and Hydrological Institute (SMHI).

```
rm(list=ls())
set.seed(1234567890)
stations <- read.csv("stations.csv")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")
rm(temps, stations)
```

defining the function

```
kernel_method <- function(df, date, loc_long, loc_lat, h1, h2, h3) {

set.seed(1234567890)
start <- as.POSIXct(date)
interval <- 60
end <- start + as.difftime(1, units="days")
time_seq <- seq(from=start, by=interval*120, to=end)
time_seq <- as.data.frame(time_seq)
colnames(time_seq) <- "new_date_time"
time_seq$time_index <- rownames(time_seq)

df_new <- merge.data.frame(df,time_seq,all=TRUE)
rm(df)

df_new$new_date <- as.Date(df_new$new_date_time)
df_new$new_time <- format(df_new$new_date_time,"%H:%M:%S")
df_new$loc_long <- loc_long
df_new$loc_lat <-  loc_lat
```

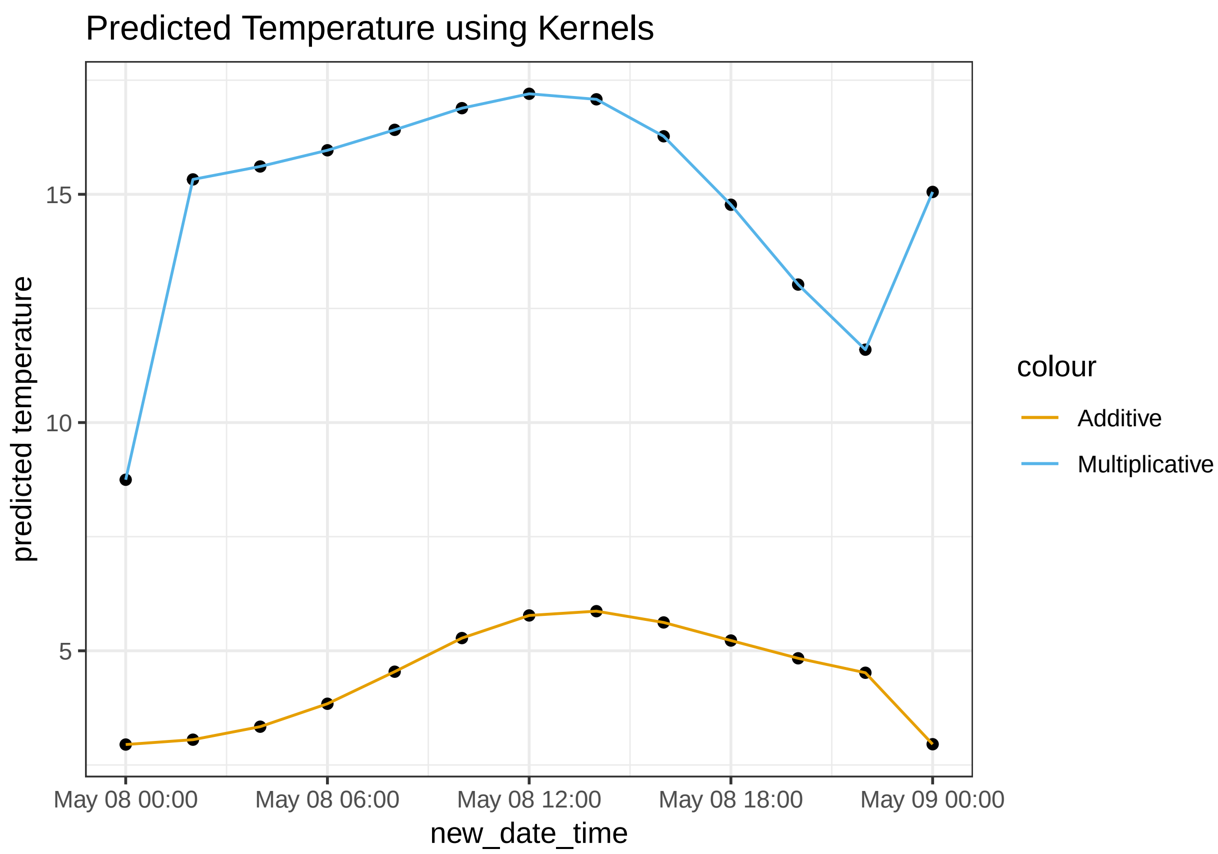
```r
df_new$h_distance <-  abs(distHaversine(p1 = df_new[,c("loc_long", "loc_lat")],
                                        p2 = df_new[,c("longitude", "latitude")]))

df_new$h_date <- as.numeric(abs(difftime(df_new$new_date, df_new$date, units = c("days"))))

df_new$h_time <- as.numeric(abs(difftime(strptime(paste(df_new$new_date,
                                                         df_new$new_time),"%Y-%m-%d%H:%M:%S"),
                                         strptime(paste(df_new$new_date, df_new$time),
                                   "%Y-%m-%d %H:%M:%S"),
                          units = c("hour"))))


df_new$date_time <- paste(df_new$date, df_new$time)
df_new$hd_dist <- as.numeric(difftime(df_new$new_date_time,
                          df_new$date_time,
                          units = c("hour")))

## removing any negative dates and time
df_new$posterior_flag <- as.factor(ifelse(df_new$h_distance > 0 & df_new$hd_dist > 0, "retain", "drop"))


## calculating kernel distance and choosing guassian kernel
df_new$h_distance_kernel <- exp(-(df_new$h_distance/h1)^2)
df_new$h_date_kernel <- exp(-(df_new$h_date/h2)^2)
df_new$h_time_kernel <- exp(-(df_new$h_time/h3)^2)
df_new$total_additive_dist <- (df_new$h_distance_kernel + df_new$h_date_kernel + df_new$h_time_kernel)
df_new$total_mul_dist <- (df_new$h_distance_kernel * df_new$h_date_kernel * df_new$h_time_kernel)

df_new$additive_num <- ifelse(df_new$posterior_flag == "retain",
                              df_new$h_distance_kernel*df_new$air_temperature +
                                df_new$h_date_kernel*df_new$air_temperature +
                                df_new$h_time_kernel*df_new$air_temperature,0)

df_new$mul_num <- ifelse(df_new$posterior_flag == "retain",
                         (df_new$h_distance_kernel) *
                                (df_new$h_date_kernel) *
                                (df_new$h_time_kernel*df_new$air_temperature),0)

df_new$additive_den <- ifelse(df_new$posterior_flag == "retain", df_new$total_additive_dist, 0)
df_new$mul_den <- ifelse(df_new$posterior_flag == "retain", df_new$total_mul_dist, 0)

time = unique(time_seq$time_index)
result <- NULL

for(i in time){
temp <- df_new[df_new$time_index == i,]
additive_temp <- sum(temp$additive_num)/sum(temp$additive_den)
mult_temp <- sum(temp$mul_num)/sum(temp$mul_den)

temp <- cbind(additive_temp, mult_temp, i)
result <- rbind(temp,result)
}
```

```r
result <- as.data.frame(result)
result <- merge(x =result, y = time_seq, by.x = "i", by.y = "time_index", all.x = TRUE)
result$additive_temp <- as.numeric(as.character(result$additive_temp))
result$mult_temp <- as.numeric(as.character(result$mult_temp))

p1 <- ggplot(data=result, aes(x=new_date_time)) +
  geom_point(aes(y = additive_temp)) +
  geom_point(aes(y = mult_temp)) +
  geom_line(aes(y = additive_temp, color = "Additive")) +
  geom_line(aes(y = mult_temp, color = "Multiplicative")) +
  scale_color_manual(values=c("#E69F00", "#56B4E9")) +
  ylab("predicted temperature") +
  theme_bw() +
  ggtitle("Predicted Temperature using Kernels")

final <- list(p1)
return(final)
}
```

calling function

```r
kernel_method(df=st, date = "2000-05-08", loc_long = 17.6935,
              loc_lat = 59.9953, h1 = 30000, h2 = 2, h3 = 5)
```

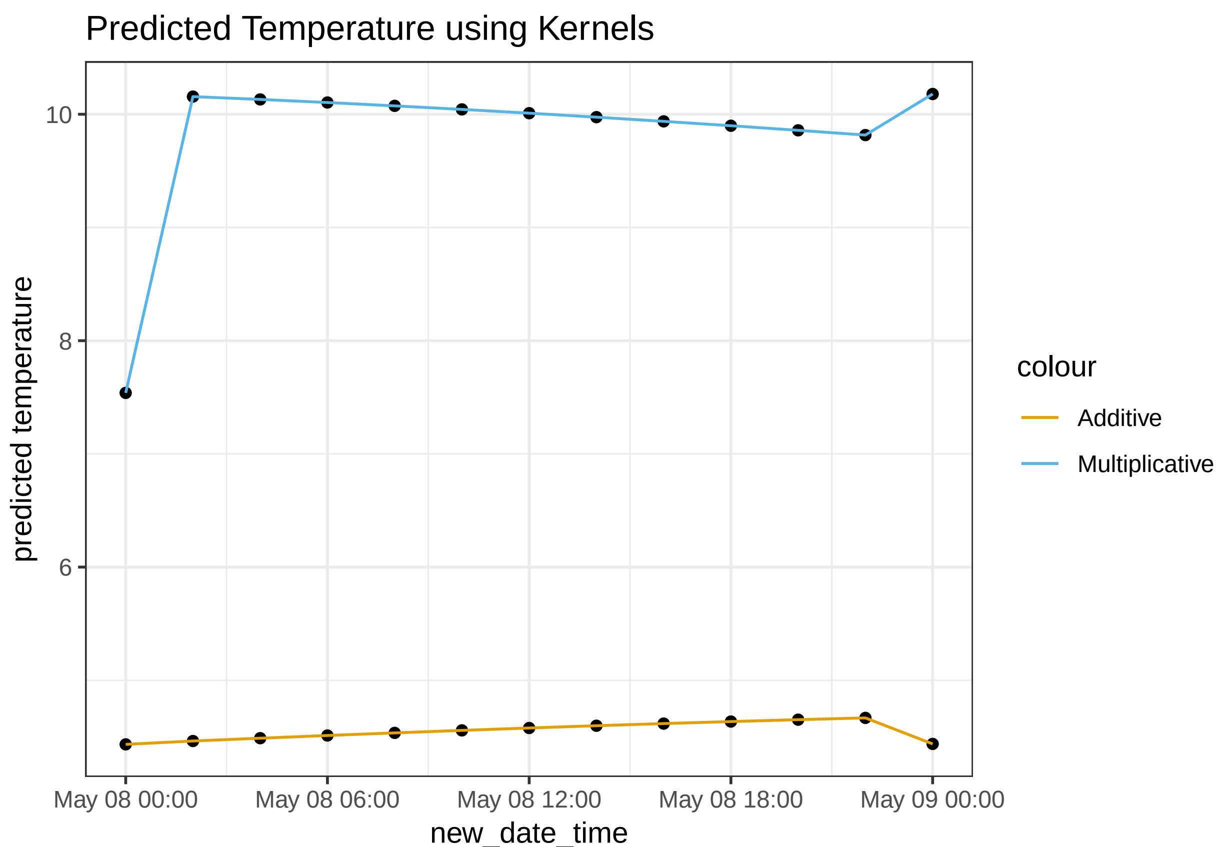
```
## [[1]]
```



high values

```
kernel_method(df=st, date = "2000-05-08", loc_long = 17.6935,
              loc_lat = 59.9953, h1 = 30000, h2 = 100, h3 = 30)
```

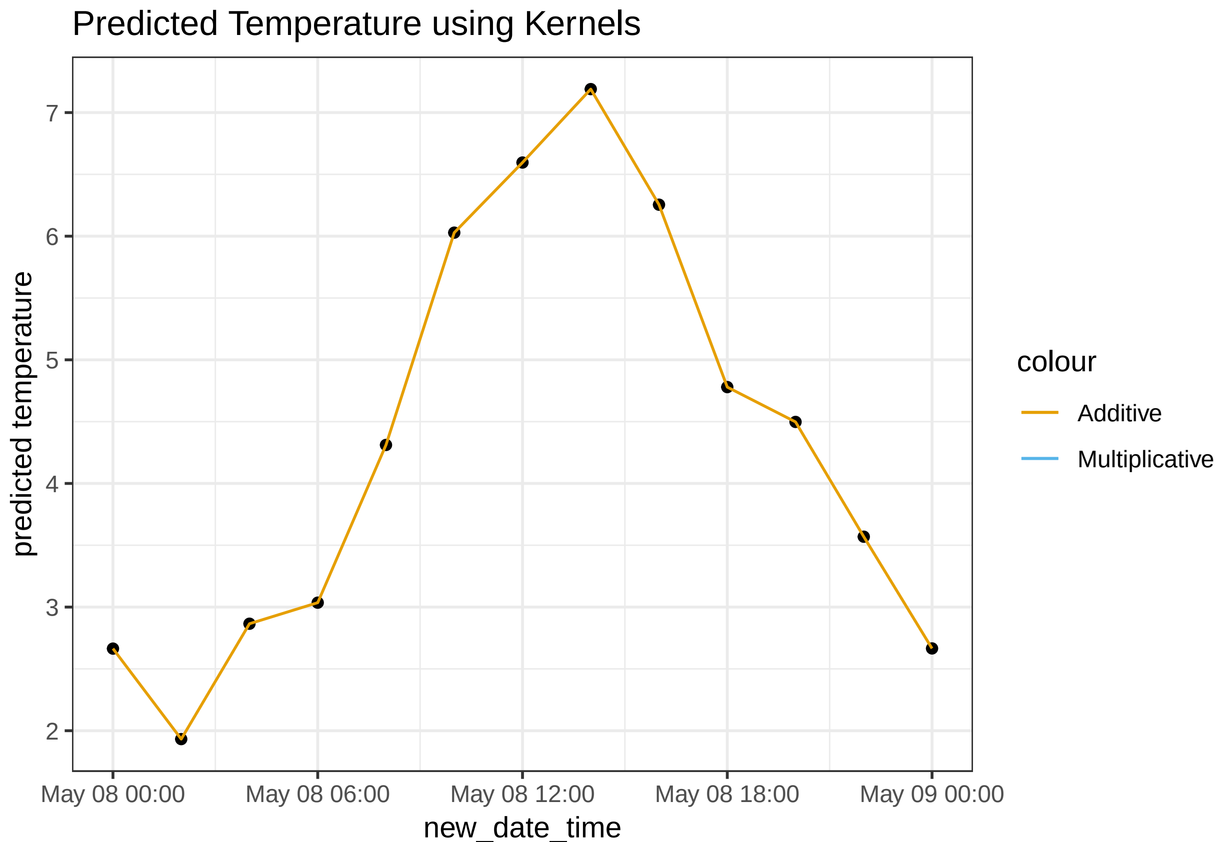
```
## [[1]]
```



low values

```
kernel_method(df=st, date = "2000-05-08", loc_long = 17.6935,
              loc_lat = 59.9953, h1 = 10, h2 = 0.05, h3 = 0.05)
```

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



Analysis:

As evident from the plots using extreme values makes either multicative model or additive model (either terms tend to zero or all terms converge to one).

A good width for the distance is 30Kms, the reasoning behind this is that temperature in Linkoping and Norrkoping tend to be similar but they vary by a few degree, given that sweden is way up north the temperature flucations will be less senstive to distance than compared to equator, thus 30Kms tend to be reasonable.

The width for the distance for day is 2, because I have personally experienced days where one days its freezing and next day I am sweating, thus 2 days is what I have choosen for my width.

For the width of time, considering the shorter winter days I do expect 3 hour of the time to be ideal window for temperature.

Assignment 2

Use the function ksvm from the R package kernlab to learn a SVM for classifying the spam dataset that is included with the package. Consider the radial basis function kernel (also known as Gaussian) with a width of 0.05. For the C parameter, consider values 0.5, 1 and 5. This implies that you have to consider three models.

```r
rm(list=ls())
set.seed(12345)

data(spam)

n = dim(spam)[1]
## create test and training set
id = sample(1:n, floor(n*0.7))
spamtrain = spam[id,]
spamtest = spam[-id,]


## train a support vector machine
model_0.05 <- ksvm(type~., data=spamtrain,
              kernel="rbfdot",
              kpar=list(sigma=0.05),
              C=0.5)

model_1.0 <- ksvm(type~.,data=spamtrain,
              kernel="rbfdot",
              kpar=list(sigma=0.05),
              C=1.0)

model_5.0 <- ksvm(type~.,data=spamtrain,
              kernel="rbfdot",
              kpar=list(sigma=0.05),
              C=5.0)

model_0.05
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 0.5
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.05
##
## Number of Support Vectors : 1391
##
## Objective Function Value : -402.9948
## Training error : 0.048758
```

```r
model_1.0
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.05
##
## Number of Support Vectors : 1287
##
```

```
## Objective Function Value : -605.8998
## Training error : 0.040062
```

```r
model_5.0
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 5
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.05
##
## Number of Support Vectors : 1206
##
## Objective Function Value : -1487.385
## Training error : 0.020807
```

```r
# confusion table

conf_model_0.05 <- table(spamtest[,58], predict(model_0.05,spamtest[,-58]))
names(dimnames(conf_model_0.05)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_model_0.05)
```

```
## Confusion Matrix and Statistics
##
##            Predicted Test
## Actual Test nonspam spam
##     nonspam     806   29
##     spam         86  460
##
##                Accuracy : 0.9167
##                  95% CI : (0.9009, 0.9308)
##     No Information Rate : 0.6459
##     P-Value [Acc > NIR] : < 0.0000000000000022
##
##                   Kappa : 0.8226
##  Mcnemar's Test P-Value : 0.000000177
##
##             Sensitivity : 0.9036
##             Specificity : 0.9407
##          Pos Pred Value : 0.9653
##          Neg Pred Value : 0.8425
##              Prevalence : 0.6459
##          Detection Rate : 0.5836
##    Detection Prevalence : 0.6046
##       Balanced Accuracy : 0.9221
##
##        'Positive' Class : nonspam
##
```

```r
conf_model_1.0 <- table(spamtest[,58], predict(model_1.0,spamtest[,-58]))
names(dimnames(conf_model_1.0)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_model_1.0)
```

```
## Confusion Matrix and Statistics
```

```
##
##            Predicted Test
## Actual Test nonspam spam
##     nonspam     800   35
##     spam         68  478
##
##                Accuracy : 0.9254
##                  95% CI : (0.9103, 0.9387)
##     No Information Rate : 0.6285
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.8424
##  Mcnemar's Test P-Value : 0.001616
##
##             Sensitivity : 0.9217
##             Specificity : 0.9318
##          Pos Pred Value : 0.9581
##          Neg Pred Value : 0.8755
##              Prevalence : 0.6285
##          Detection Rate : 0.5793
##    Detection Prevalence : 0.6046
##       Balanced Accuracy : 0.9267
##
##        'Positive' Class : nonspam
##
```

```r
conf_model_0.05 <- table(spamtest[,58], predict(model_5.0,spamtest[,-58]))
names(dimnames(conf_model_0.05)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_model_0.05)
```

```
## Confusion Matrix and Statistics
##
##            Predicted Test
## Actual Test nonspam spam
##     nonspam     798   37
##     spam         70  476
##
##                Accuracy : 0.9225
##                  95% CI : (0.9071, 0.9361)
##     No Information Rate : 0.6285
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.8362
##  Mcnemar's Test P-Value : 0.001978
##
##             Sensitivity : 0.9194
##             Specificity : 0.9279
##          Pos Pred Value : 0.9557
##          Neg Pred Value : 0.8718
##              Prevalence : 0.6285
##          Detection Rate : 0.5778
##    Detection Prevalence : 0.6046
##       Balanced Accuracy : 0.9236
##
##        'Positive' Class : nonspam
```

##

Analysis:

From the summary of the three models build we can see that the accuracy of models are 90.83%, 91.61%, 91.70% respectively. Accuracy is only half the story, as a good spam detection should never classify a good mail has 'spam', which is something that model1 is doing. However in model 1 also has the least accuracy however its marginally bad. Given a choice i would select model1 has the best model despite the lower accuracy.

Purpose of the 'C' parameter:- C is the cost parameter which penalizes large residuals. So a larger cost will result in a more flexible model with fewer misclassifications. In effect the cost parameter allows you to adjust the bias/variance trade-off. The greater the cost parameter, the more variance in the model and the less bias.The greater the cost, the fewer misclassifications are allowed. Note that here we penalize the residuals resulting in higher variance and lower bias.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
if (!require("pacman")) install.packages("pacman")
pacman::p_load(geosphere, kernlab, geosphere, ggplot2,caret)

set.seed(12345)
options("jtools-digits" = 2, scipen = 999)

# colours (colour blind friendly)
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2",
               "#D55E00", "#CC79A7")

rm(list=ls())
set.seed(1234567890)
stations <- read.csv("stations.csv")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")
rm(temps, stations)
kernel_method <- function(df, date, loc_long, loc_lat, h1, h2, h3) {

set.seed(1234567890)
start <- as.POSIXct(date)
interval <- 60
end <- start + as.difftime(1, units="days")
time_seq <- seq(from=start, by=interval*120, to=end)
time_seq <- as.data.frame(time_seq)
colnames(time_seq) <- "new_date_time"
time_seq$time_index <- rownames(time_seq)

df_new <- merge.data.frame(df,time_seq,all=TRUE)
rm(df)

df_new$new_date <- as.Date(df_new$new_date_time)
df_new$new_time <- format(df_new$new_date_time,"%H:%M:%S")
df_new$loc_long <- loc_long
df_new$loc_lat <-  loc_lat
```

```r
df_new$h_distance <-  abs(distHaversine(p1 = df_new[,c("loc_long", "loc_lat")],
                                    p2 = df_new[,c("longitude", "latitude")]))

df_new$h_date <- as.numeric(abs(difftime(df_new$new_date, df_new$date, units = c("days"))))

df_new$h_time <- as.numeric(abs(difftime(strptime(paste(df_new$new_date,
                                                   df_new$new_time),"%Y-%m-%d%H:%M:%S"),
                                         strptime(paste(df_new$new_date, df_new$time),
                                   "%Y-%m-%d %H:%M:%S"),
                          units = c("hour"))))


df_new$date_time <- paste(df_new$date, df_new$time)
df_new$hd_dist <- as.numeric(difftime(df_new$new_date_time,
                          df_new$date_time,
                          units = c("hour")))

## removing any negative dates and time
df_new$posterior_flag <- as.factor(ifelse(df_new$h_distance > 0 & df_new$hd_dist > 0, "retain", "drop"))


## calculating kernel distance and choosing guassian kernel
df_new$h_distance_kernel <- exp(-(df_new$h_distance/h1)^2)
df_new$h_date_kernel <- exp(-(df_new$h_date/h2)^2)
df_new$h_time_kernel <- exp(-(df_new$h_time/h3)^2)
df_new$total_additive_dist <- (df_new$h_distance_kernel + df_new$h_date_kernel + df_new$h_time_kernel)
df_new$total_mul_dist <- (df_new$h_distance_kernel * df_new$h_date_kernel * df_new$h_time_kernel)

df_new$additive_num <- ifelse(df_new$posterior_flag == "retain",
                              df_new$h_distance_kernel*df_new$air_temperature +
                                df_new$h_date_kernel*df_new$air_temperature +
                                df_new$h_time_kernel*df_new$air_temperature,0)

df_new$mul_num <- ifelse(df_new$posterior_flag == "retain",
                         (df_new$h_distance_kernel) *
                                (df_new$h_date_kernel) *
                                (df_new$h_time_kernel*df_new$air_temperature),0)

df_new$additive_den <- ifelse(df_new$posterior_flag == "retain", df_new$total_additive_dist, 0)
df_new$mul_den <- ifelse(df_new$posterior_flag == "retain", df_new$total_mul_dist, 0)

time = unique(time_seq$time_index)
result <- NULL

for(i in time){
temp <- df_new[df_new$time_index == i,]
additive_temp <- sum(temp$additive_num)/sum(temp$additive_den)
mult_temp <- sum(temp$mul_num)/sum(temp$mul_den)

temp <- cbind(additive_temp, mult_temp, i)
result <- rbind(temp,result)
}
```

```r
result <- as.data.frame(result)
result <- merge(x =result, y = time_seq, by.x = "i", by.y = "time_index", all.x = TRUE)
result$additive_temp <- as.numeric(as.character(result$additive_temp))
result$mult_temp <- as.numeric(as.character(result$mult_temp))

p1 <- ggplot(data=result, aes(x=new_date_time)) +
  geom_point(aes(y = additive_temp)) +
  geom_point(aes(y = mult_temp)) +
  geom_line(aes(y = additive_temp, color = "Additive")) +
  geom_line(aes(y = mult_temp, color = "Multiplicative")) +
  scale_color_manual(values=c("#E69F00", "#56B4E9")) +
  ylab("predicted temperature") +
  theme_bw() +
  ggtitle("Predicted Temperature using Kernels")

final <- list(p1)
return(final)
}

kernel_method(df=st, date = "2000-05-08", loc_long = 17.6935,
              loc_lat = 59.9953, h1 = 30000, h2 = 2, h3 = 5)
kernel_method(df=st, date = "2000-05-08", loc_long = 17.6935,
              loc_lat = 59.9953, h1 = 30000, h2 = 100, h3 = 30)
kernel_method(df=st, date = "2000-05-08", loc_long = 17.6935,
              loc_lat = 59.9953, h1 = 10, h2 = 0.05, h3 = 0.05)
rm(list=ls())
set.seed(12345)

data(spam)

n = dim(spam)[1]
## create test and training set
id = sample(1:n, floor(n*0.7))
spamtrain = spam[id,]
spamtest = spam[-id,]


## train a support vector machine
model_0.05 <- ksvm(type~., data=spamtrain,
               kernel="rbfdot",
               kpar=list(sigma=0.05),
               C=0.5)

model_1.0 <- ksvm(type~.,data=spamtrain,
               kernel="rbfdot",
               kpar=list(sigma=0.05),
               C=1.0)

model_5.0 <- ksvm(type~.,data=spamtrain,
               kernel="rbfdot",
               kpar=list(sigma=0.05),
               C=5.0)
```

```r
model_0.05
model_1.0
model_5.0

# confusion table

conf_model_0.05 <- table(spamtest[,58], predict(model_0.05,spamtest[,-58]))
names(dimnames(conf_model_0.05)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_model_0.05)

conf_model_1.0 <- table(spamtest[,58], predict(model_1.0,spamtest[,-58]))
names(dimnames(conf_model_1.0)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_model_1.0)

conf_model_0.05 <- table(spamtest[,58], predict(model_5.0,spamtest[,-58]))
names(dimnames(conf_model_0.05)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_model_0.05)
```