Machine Learning - Lab03 - Group A2

Thijs Quast (thiqu264), Anubhav Dikshit(anudi287), Lennart Schilling(lensc874)

18-12-2018

Contents

Contributions

For this report Thijs and Anubhav focused on assignment 1. Lennart focused on assignment 2. All code was written individually and independently.

Loading The Libraries

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

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 - Kernel Methods

```
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


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"),
```

```r
                                       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)
}

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)) +
```

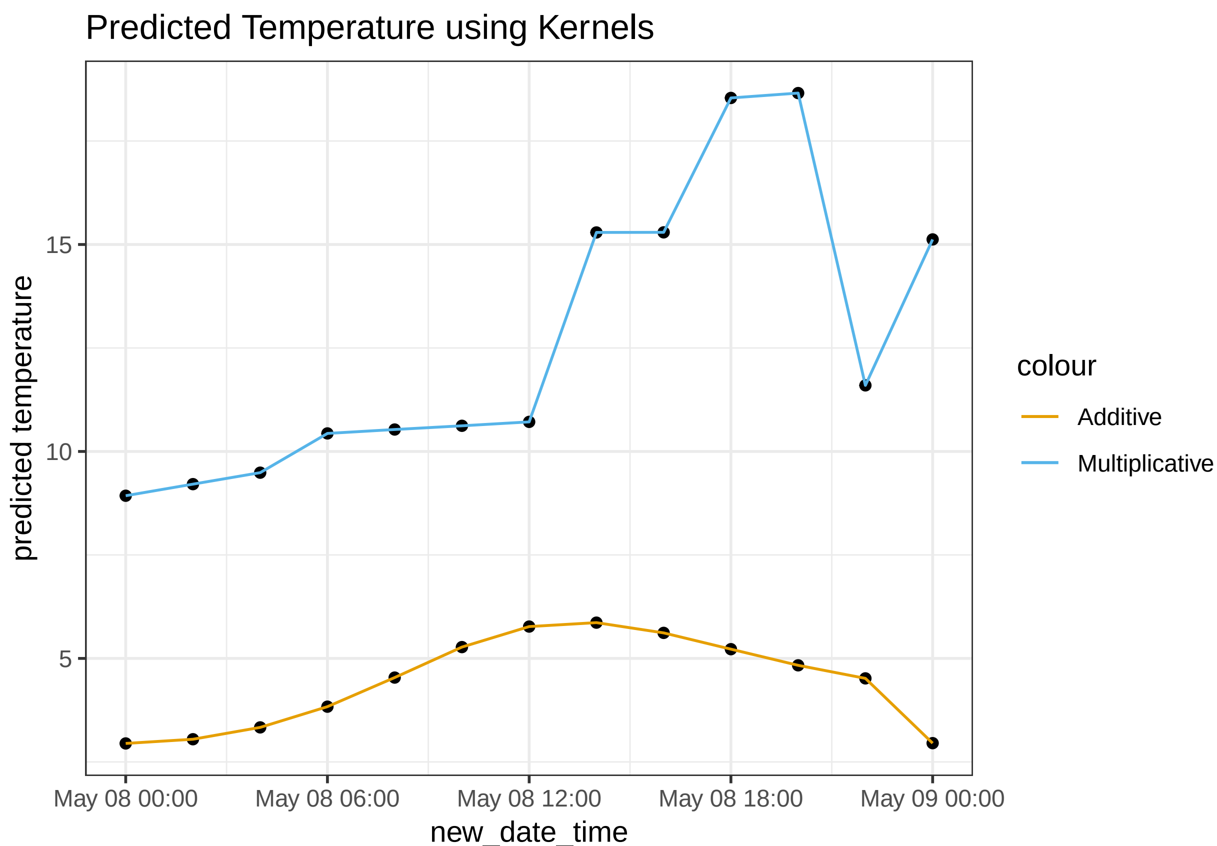
```
  geom_point(aes(y = mult_temp)) +
  geom_line(aes(y = additive_temp, color = "Additive")) +
  geom_line(aes(y = mult_temp, color = "Multiplicative")) +
  scale_color_manual(values=c("#E69F00", "#56B4E9")) +
  ylab("predicted temperature") +
  theme_bw() +
  ggtitle("Predicted Temperature using Kernels")

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

calling function

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

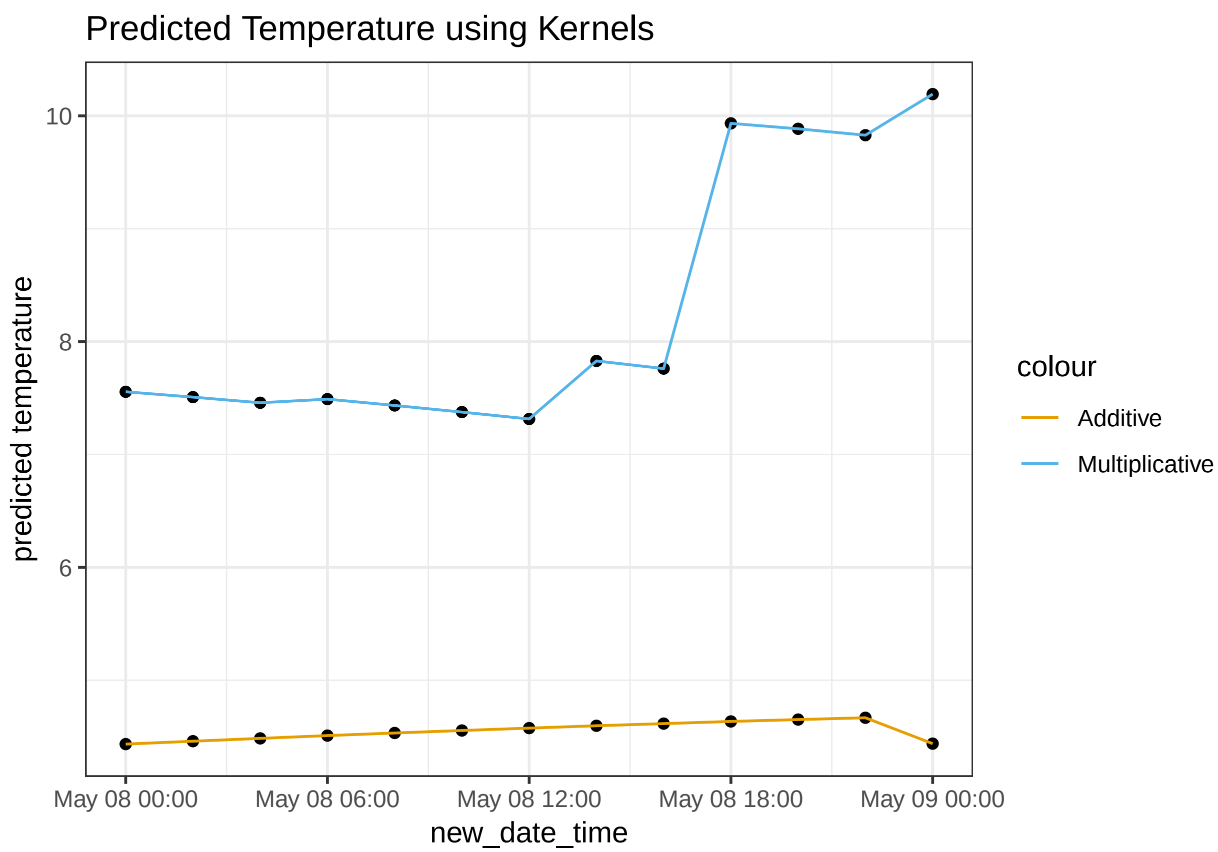
```
## [[1]]
```



high values

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

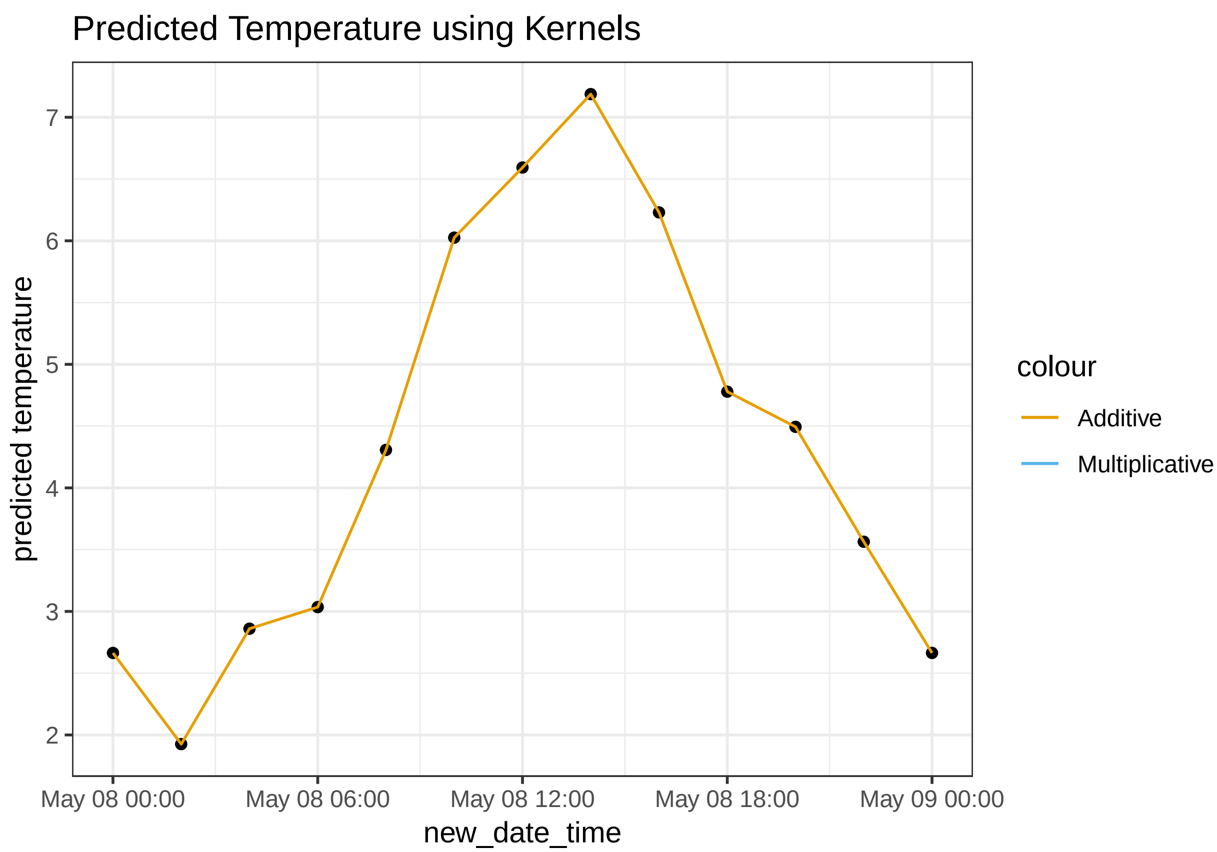
```
## [[1]]
```



low values

```r
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:

For the kernel widths I have chosen the following values:

Distance: 30 kilometers

Date: 2 days

Time: 5 hours 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).When the widths are extremely large the values are saturated and there is no variations while when the values are really small the prediction by multiplicative effectively yields 0 while summation kernel predicted highly varying curve.

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 - Support Vector Machines

In this assignment, a Support Vector Machine (SVM), a supervised learning model, will be used to classify spam dataset that is included within the *kernlab*-package which will be used for this assignment.

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

data(spam)

# separate data to training, validation, and test - 40/30/30
n <- NROW(spam)
set.seed(12345)
id <- sample(1:n, floor(n*0.4))
train <- spam[id,]
id1 <- setdiff(1:n, id)
id2 <- sample(id1, floor(n*0.3))
valid <- spam[id2,]
id3 <- setdiff(id1,id2)
test <- spam[id3,]


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

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

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

# confusion table
conf_model_0.05 <- table(test[,58], predict(model_0.05,test[,-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     805   26
##     spam        105  445
##
##                 Accuracy : 0.9051
##                   95% CI : (0.8885, 0.9201)
##     No Information Rate : 0.6589
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                    Kappa : 0.7972
##  Mcnemar's Test P-Value : 0.000000000009433
##
##              Sensitivity : 0.8846
```

```
##             Specificity : 0.9448
##          Pos Pred Value : 0.9687
##          Neg Pred Value : 0.8091
##              Prevalence : 0.6589
##          Detection Rate : 0.5829
##    Detection Prevalence : 0.6017
##       Balanced Accuracy : 0.9147
##
##        'Positive' Class : nonspam
##
```

```
conf_model_1.0 <- table(test[,58], predict(model_1.0,test[,-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     801   30
##     spam         82  468
##
##                Accuracy : 0.9189
##                  95% CI : (0.9032, 0.9328)
##     No Information Rate : 0.6394
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.828
##  Mcnemar's Test P-Value : 0.000001442
##
##             Sensitivity : 0.9071
##             Specificity : 0.9398
##          Pos Pred Value : 0.9639
##          Neg Pred Value : 0.8509
##              Prevalence : 0.6394
##          Detection Rate : 0.5800
##    Detection Prevalence : 0.6017
##       Balanced Accuracy : 0.9234
##
##        'Positive' Class : nonspam
##
```

```
conf_model_0.05 <- table(test[,58], predict(model_5.0,test[,-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   33
##     spam         77  473
##
##                Accuracy : 0.9203
##                  95% CI : (0.9048, 0.9341)
```

```
##     No Information Rate : 0.6336
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.8315
##  Mcnemar's Test P-Value : 0.00004133
##
##             Sensitivity : 0.9120
##             Specificity : 0.9348
##          Pos Pred Value : 0.9603
##          Neg Pred Value : 0.8600
##              Prevalence : 0.6336
##          Detection Rate : 0.5778
##    Detection Prevalence : 0.6017
##       Balanced Accuracy : 0.9234
##
##        'Positive' Class : nonspam
##
```

```r
# choosing the first model C=0.05, now trainning the model on the full dataset to get generalised error
train_test <- rbind(train,test)

final_svm_model <- ksvm(type~., data=train_test,
               kernel="rbfdot",
               kpar=list(sigma=0.05),
               C=0.5)

final_model_0.05 <- table(valid[,58], predict(final_svm_model, valid[,-58]))
names(dimnames(final_model_0.05)) <- c("Actual Validatation", "Predicted Validatation")
caret::confusionMatrix(final_model_0.05)
```

```
## Confusion Matrix and Statistics
##
##                      Predicted Validatation
## Actual Validatation nonspam spam
##             nonspam     787   32
##             spam         89  472
##
##                Accuracy : 0.9123
##                  95% CI : (0.8961, 0.9267)
##     No Information Rate : 0.6348
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.8153
##  Mcnemar's Test P-Value : 0.0000003564
##
##             Sensitivity : 0.8984
##             Specificity : 0.9365
##          Pos Pred Value : 0.9609
##          Neg Pred Value : 0.8414
##              Prevalence : 0.6348
##          Detection Rate : 0.5703
##    Detection Prevalence : 0.5935
##       Balanced Accuracy : 0.9175
##
##        'Positive' Class : nonspam
```

##

Analysis:

From the summary of the three models build we can see that the accuracy of models are 90.51%, 91.89%, 92.03% (for C=0.05, 1 and 5 respectively). Accuracy is only half the story, as a good spam detection should never classify a good mail has 'spam' (university acceptance has the same words as a spam mail eg: 'congratulations', 'opportunity', 'pleased', 'deadline/hurry'), which is something that model1 is doing, however our model 1 also has the least accuracy.

Additionally 'spamers' are 'evolutionary' by nature, the subject and style of their mails keep changing (nigerian prince providing you money, bitcoins from Elon Musk), thus our model should never be overtrained or I daresay it should be a bit more on the underpredicted side. Thus from the above two reasons I would select model1 which has the highest ability of identifying non-spam mails and is off by only ~1% in terms of accuracy compared to our best model.

Best on our selection the generalized model error for our chosen model is 8.77% (accuracy 91.23%) and its ability to correctly identify nonspam mail is 96.09%.

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

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

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)
```

```r
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


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)
}

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)

# separate data to training, validation, and test - 40/30/30
n <- NROW(spam)
set.seed(12345)
id <- sample(1:n, floor(n*0.4))
train <- spam[id,]
id1 <- setdiff(1:n, id)
id2 <- sample(id1, floor(n*0.3))
valid <- spam[id2,]
id3 <- setdiff(id1,id2)
test <- spam[id3,]


## train a support vector machine
model_0.05 <- ksvm(type~., data=train,
```

```
                kernel="rbfdot",
                kpar=list(sigma=0.05),
                C=0.5)

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

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

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

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

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

# choosing the first model C=0.05, now trainning the model on the full dataset to get generalised error
train_test <- rbind(train,test)

final_svm_model <- ksvm(type~., data=train_test,
                kernel="rbfdot",
                kpar=list(sigma=0.05),
                C=0.5)

final_model_0.05 <- table(valid[,58], predict(final_svm_model, valid[,-58]))
names(dimnames(final_model_0.05)) <- c("Actual Validatation", "Predicted Validatation")
caret::confusionMatrix(final_model_0.05)
```