BDA3 - Machine Learning with Spark - Exercises

Saewon Jun (saeju204) and Anubhav Dikshit (anudi287)

2019-05-24

Contents

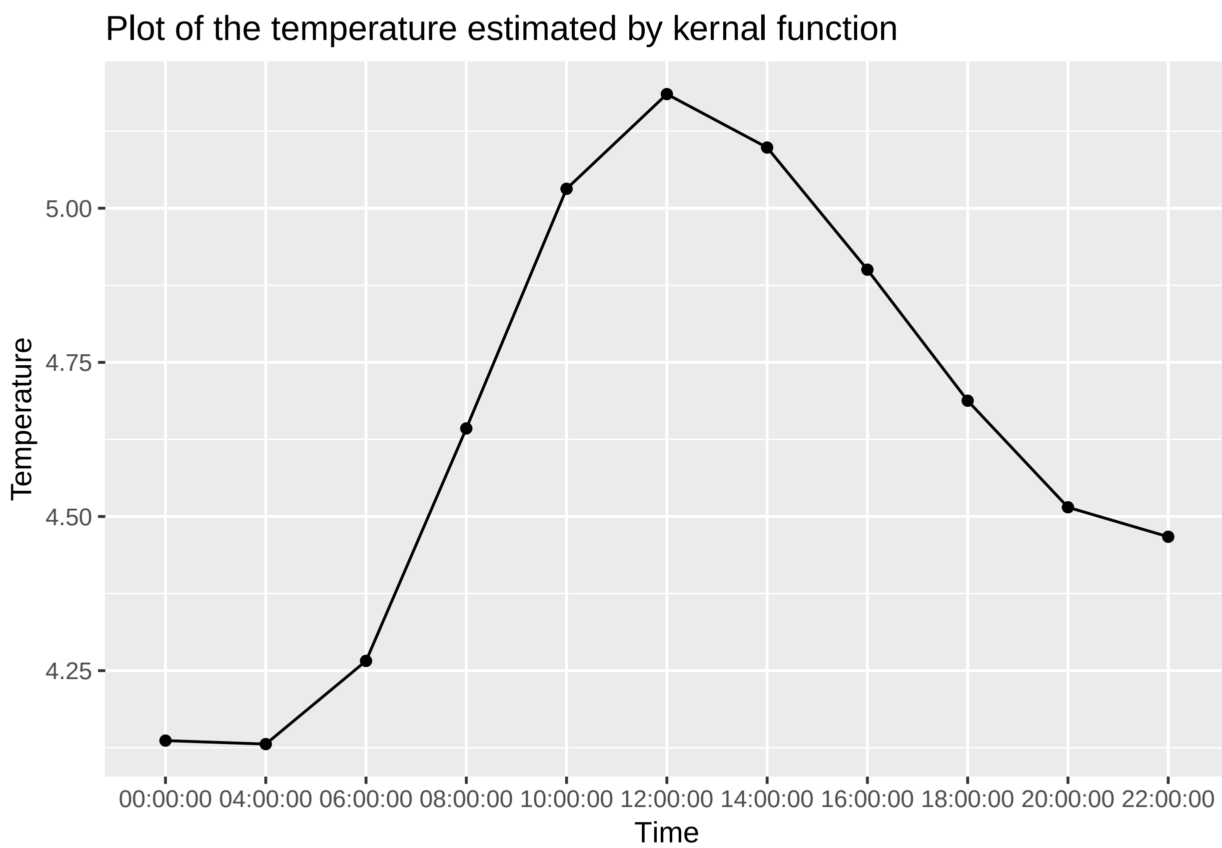
Introduction:

In this lab we were supposed to implement a kernel to predict temperatures. The kernel is a linear combination of three Gaussian kernels (date, time of day and geographical distance)

The estimated temperatures below were predicted in Linkoping 2013-08-04. As we can notice that the lower temperatures in the morning and evening with a higher temperatures in the afternoon. However, the values of the temperatures doesn't reflects the common temperatures during the winter time which normlly below zero, This indicates that the kernel does not consider seasonal time.

Result

```r
library("tidyverse")

temps <- read.csv('temp.csv', header = TRUE, sep = ',')
ggplot(temps, aes(time, temperature, group = 1)) +
geom_point() +
geom_line() +
labs(x = "Time",
y = "Temperature") +
ggtitle("Plot of the temperature estimated by kernal function")
```



Kernals:

Date kernal:

For the date kernal, we set the width to 4 days as we assumed that the temperature of a particular day is close to what it has been the 4 previous days.

Time kernal:

For the time kernel, We set the width to be 2 hours since the last few hours should have a a closer temperatures as the current time.

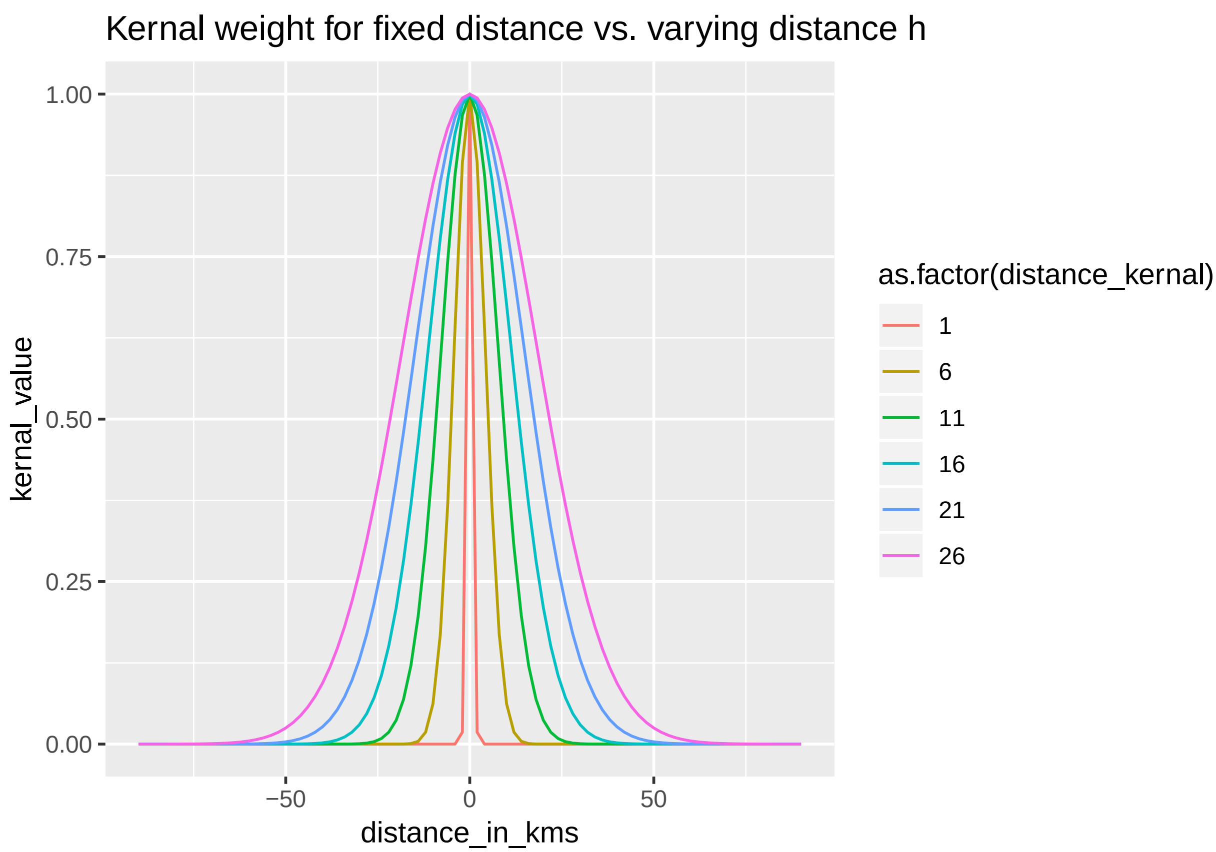
Time kernal:

For the distance kernel we set the width of 100 Kms which we assumed that is a reasonable measure based on the size of Sweden as the location affects the temperature quite much where the further to the north the colder.

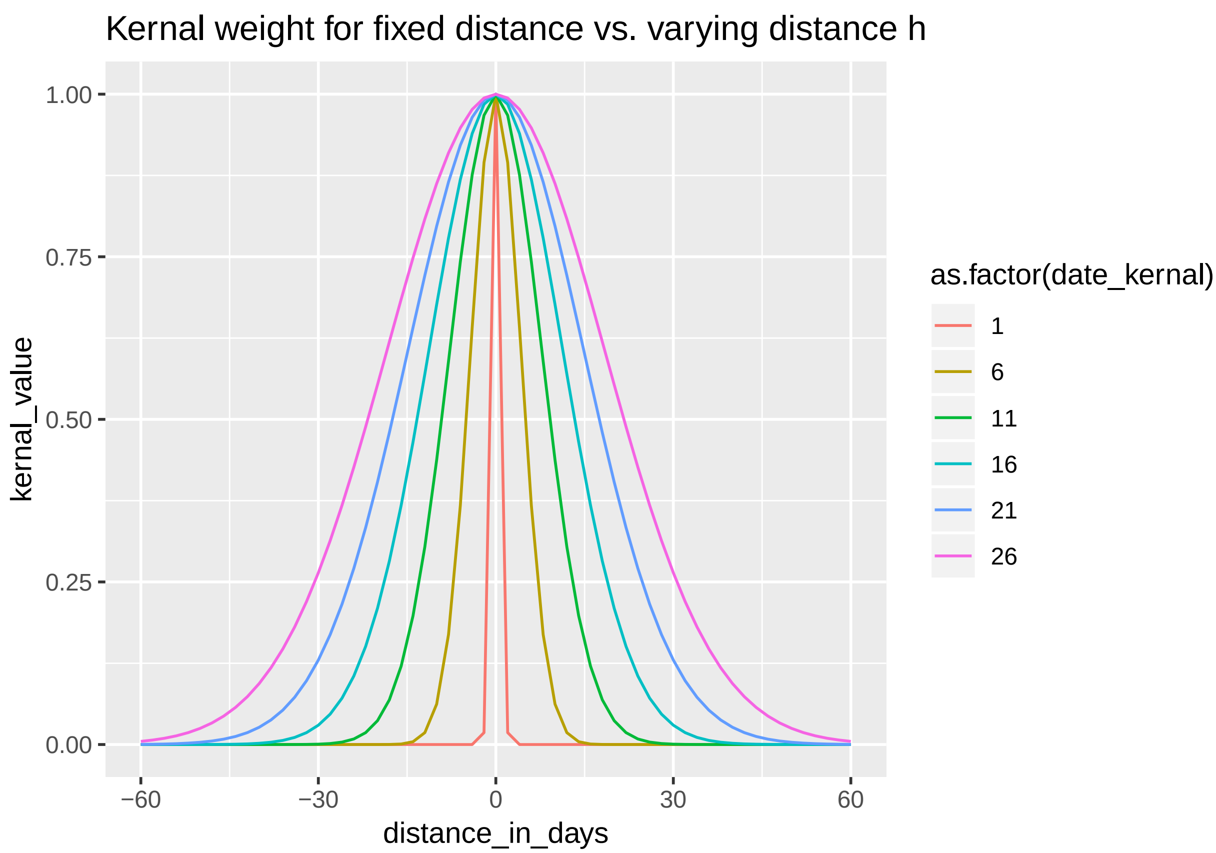
Reasoning for kernal:

```
h_distance_seq <- seq(-90,90,2) %>% as.data.frame()
colnames(h_distance_seq) <- c("distance_in_kms")
```

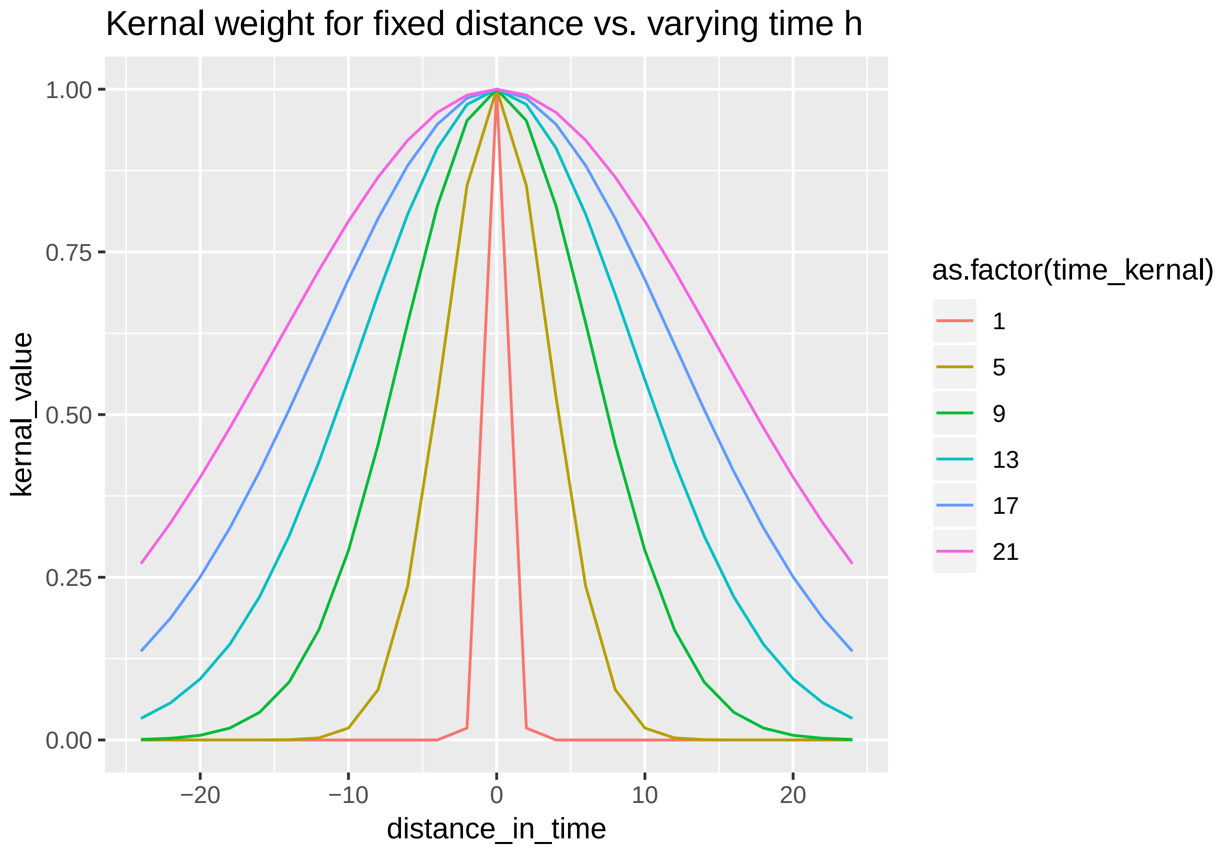
```
h_date_seq <- seq(-60,60,2) %>% as.data.frame()
colnames(h_date_seq) <- c("distance_in_days")

h_time_seq <- seq(-24,24,2) %>% as.data.frame()
colnames(h_time_seq) <- c("distance_in_time")

distance_kernal <- seq(1, 30, 5)
date_kernal <- seq(1, 30, 5)
time_kernal <- seq(1, 24, 4)

h_distance_seq <- crossing(h_distance_seq, distance_kernal)
h_date_seq <- crossing(h_date_seq, date_kernal)
h_time_seq <- crossing(h_time_seq, time_kernal)


distance_value <- NULL
for(i in 1:NROW(h_distance_seq)){
distance_value[i] <- exp(-(h_distance_seq[i,1]/h_distance_seq[i,2])^2)
}

date_value <- NULL
for(i in 1:NROW(h_date_seq)){
date_value[i] <- exp(-(h_date_seq[i,1]/h_date_seq[i,2])^2)
}

time_value <- NULL
for(i in 1:NROW(h_time_seq)){
time_value[i] <- exp(-(h_time_seq[i,1]/h_time_seq[i,2])^2)
}

h_distance_seq$kernal_value <- distance_value
h_date_seq$kernal_value <- date_value
h_time_seq$kernal_value <- time_value

ggplot(data=h_distance_seq, aes(x=distance_in_kms, y=kernal_value, color=as.factor(distance_kernal))) +
  geom_line() +
  ggtitle("Kernal weight for fixed distance vs. varying distance h")
```




```
ggplot(data=h_date_seq, aes(x=distance_in_days, y=kernal_value, color=as.factor(date_kernal))) +
  geom_line() +
  ggtitle("Kernal weight for fixed distance vs. varying distance h")
```




```
ggplot(data=h_time_seq, aes(x=distance_in_time, y=kernal_value, color=as.factor(time_kernal))) +
  geom_line() +
  ggtitle("Kernal weight for fixed distance vs. varying time h")
```



Analysis: As evident the kernel value of 4 days, 2 hours and 100 kms makes sense since it accounts for local changes but ignores far of values

Conclusion:

We find that altough the predicted valuea are reasonable using multiplicative kernals could be even better approach.

Code:

```
from __future__ import division
from math import radians, cos, sin, asin, sqrt, exp
from datetime import datetime
from pyspark import SparkContext
sc = SparkContext(appName="lab3_anudi")

stations_file = sc.textFile("/user/x_andui/Data/stations.csv")
temps_file = sc.textFile("/user/x_anudi/Data/temperature-readings.csv")
#temps_file = temps_file.sample(False,0.1)

def haversine(lon1, lat1, lon2, lat2):
    """Calculate the great circle distance between two points on the
    earth (specified in decimal degrees)"""
    # convert decimal degrees to radians
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
    # haversine formula
    dlon = lon2 -lon1
    dlat = lat2 -lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    km = 6367 * c
    return km

h_distance = 100 #km
h_date = 4 # Days
h_time = 2*60*60 # Seconds
a = 58.4274 # Up to you
b = 14.826 # Up to you
date = "2013-08-04" # Up to you
times = ["04:00:00","06:00:00","08:00:00","10:00:00","12:00:00",
    "14:00:00","16:00:00","18:00:00","20:00:00","22:00:00","00:00:00"]
temp = [0]*len(times)
temp_sum = [0]*len(times)
temp_mul = [0]*len(times)

stationLines = stations_file.map(lambda line: line.split(";"))
stations = stationLines.map(lambda x: (int(x[0]), float(x[3]),
    float(x[4])))

tempLines = temps_file.map(lambda line: line.split(";"))
temps = tempLines.map(lambda x: (int(x[0]),x[1],x[2],float(x[3])))

def gaussianKernelDist(data,coords, h):
    u = data.map(lambda x: (x[0],haversine(x[2],x[1],coords[0],coords[1])/h))
    k = u.map(lambda x: (x[0],exp(-(x[1]**2))))
    #print k.collect()
    return k

def gaussianKernelDate(x,date,h):
    diff_date = (datetime(int(x[0:4]),int(x[5:7]),int(x[8:10]))
        - datetime(int(date[0:4]),int(date[5:7]),int(date[8:10]))).days / h
```

```
    k = exp(-(diff_date**2))
    #print(k.collect())
    return k

def gaussianKernelTime(x,time,h):
    diff_time = (datetime(2000,1,1,int(x[0:2]),int(x[3:5]),int(x[6:8]))
        - datetime(2000,1,1,int(time[0:2]),int(time[3:5]),int(time[6:8]))).seconds / h
    k = exp(-(diff_time**2))
    #print k.collect()
    return k


def predict():
    k_dist = gaussianKernelDist(stations,[b,a],h_distance)
    k_dist_broadcast = k_dist.collectAsMap()
    stations_dist = sc.broadcast(k_dist_broadcast)

    #Filter on date
    filtered_dates = temps.filter(lambda x:
        (datetime(int(x[1][0:4]),int(x[1][5:7]),int(x[1][8:10]))
        <= datetime(int(date[0:4]),int(date[5:7]),int(date[8:10]))))
    filtered_dates.cache()

    for time in times:
        #Filter on time
        filtered_times = filtered_dates.filter(lambda x:
            ((datetime(int(x[1][0:4]),int(x[1][5:7]),int(x[1][8:10]))
            == datetime(int(date[0:4]),int(date[5:7]),int(date[8:10])))) and
            (datetime(2000,1,1,int(x[2][0:2]),int(x[2][3:5]),int(x[2][6:8]))
            <= datetime(2000,1,1,int(time[0:2]),int(time[3:5]),int(time[6:8]))))


        kernel = filtered_times.map(lambda x: (stations_dist.value[x[0]],
            gaussianKernelDate(x[1],date,h_date),
            gaussianKernelTime(x[2],time,h_time),x[3]))

        k_sum = kernel.map(lambda x: (x[0] * x[1] * x[2],x[3]))
        k_sum = k_sum.map(lambda x: (x[0]*x[1],x[0]))
        k_sum = k_sum.reduce(lambda x,y: (x[0]+y[0],x[1]+y[1]))
        temp_sum[times.index(time)] = (time,k_sum[0]/k_sum[1])

predict()
temp_sum.saveAsTextFile("/user/x_anudi/Data/lab3_result")
```