Lab 3

Ali Etminan, Ahmed Alhasan

2020-06-13

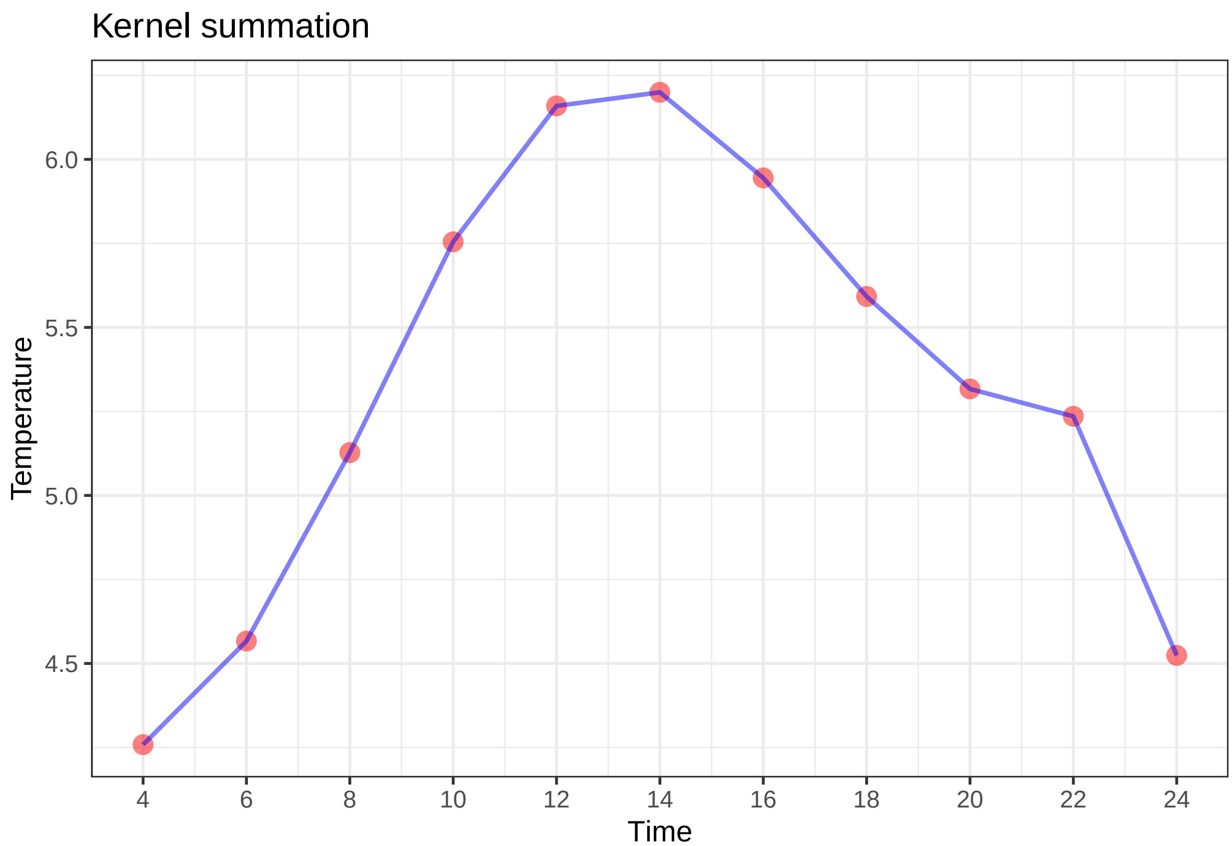
Q1

```
from __future__ import division
from math import radians, cos, sin, asin, sqrt, exp
from datetime import datetime
from pyspark import SparkContext, SparkConf

conf = SparkConf().setAppName("lab_kernel")#.setMaster("local[*]")
sc = SparkContext.getOrCreate()

### Parameters
h_distance = 100
h_date = 30
h_time = 3
lat = 58.4274
long = 14.826

### Forecasted Date & Time
date = "2013-07-04"
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'

### Data
temps = sc.textFile("BDA/input/temperature-readings.csv").map(lambda line: line.split(";"))
# (station, (date, time, temp))
temps = temps.map(lambda x: (x[0], (x[1], x[2], float(x[3]))))

stations = sc.textFile("BDA/input/stations.csv").map(lambda line: line.split(";"))
# (station, (lat, long))
stations = stations.map(lambda x: (x[0],(x[3], x[4])))

station_loc = stations.collectAsMap()
bc = sc.broadcast(station_loc)

# (station, (date, time, temp), (lat, long))
joined_rdd = temps.map(lambda x: (x[0], x[1], bc.value.get(x[0])))
joined_rdd = joined_rdd.filter(lambda x: x[1][0] < date)

def haversine(lon1, lat1, lon2, lat2):
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
```

```python
    """
    # convert decimal degrees to radians
    lon1, lat1, lon2, lat2 = map(radians, [float(lon1), float(lat1), float(lon2), float(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

def dateDist(day_1, day_2):
    """
    Returns date distance as number of days
    """
    day_1 = datetime.strptime(day_1, "%Y-%m-%d")
    day_2 = datetime.strptime(day_2, "%Y-%m-%d")
    dist  = day_1 - day_2
    return abs(dist.days)

def timeDist(time_1, time_2):
    """
    Takes time in hours and returns the distance in seconds
    """
    time_1 = datetime.strptime(time_1, "%H:%M:%S")
    time_2 = datetime.strptime(time_2, "%H:%M:%S")
    dist = time_1 - time_2
    return abs(dist.total_seconds()/3600)

def getKernel(h,dist):
    """
    Return the kernel given the distance function and spread parameter h
    """
    var = 2 * (h**2)
    dist = dist**2
    kernel = exp(-dist/var)
    return kernel

cached_sum = joined_rdd.map(lambda x: (getKernel(h_date, dateDist(date, x[1][0])) + \
                                       getKernel(h_distance, haversine(long, lat, x[2][1], x[2][0])),
                                       x[1][1],
                                       float(x[1][2]))).persist()


sums = []
for time in times:
    num, den = cached_sum.map(lambda x: (x[0] + getKernel(h_time, timeDist(time, x[1])), x[2])) \
                         .map(lambda x: (x[0]*x[1], x[0])) \
                         .reduce(lambda x,y: (x[0]+y[0], x[1]+y[1]))
    sums.append(num/den)

print(sums)
sums_file = open("sums.txt", "w+")
```

```
sums_file.write(str(sums))
sums_file.close()
```

```
sum_forecast <- c(4.2584, 4.5666, 5.1275, 5.7547, 6.1591, 6.1995, 5.9448, 5.5920, 5.3169, 5.2354, 4.524
sum_forecast <- data.frame(time = seq(4,24,2), temp = sum_forecast)


library(ggplot2)
ggplot(sum_forecast, aes(x = time, y = temp)) +
  geom_point(color="red", alpha=0.5, size=3) +
  geom_line(color="blue", alpha=0.5, size=0.8) +
  labs(title="Kernel summation",
       x = "Time", y = "Temperature") +
  scale_x_continuous(breaks = seq(4,24,2)) +
  theme_bw()
```

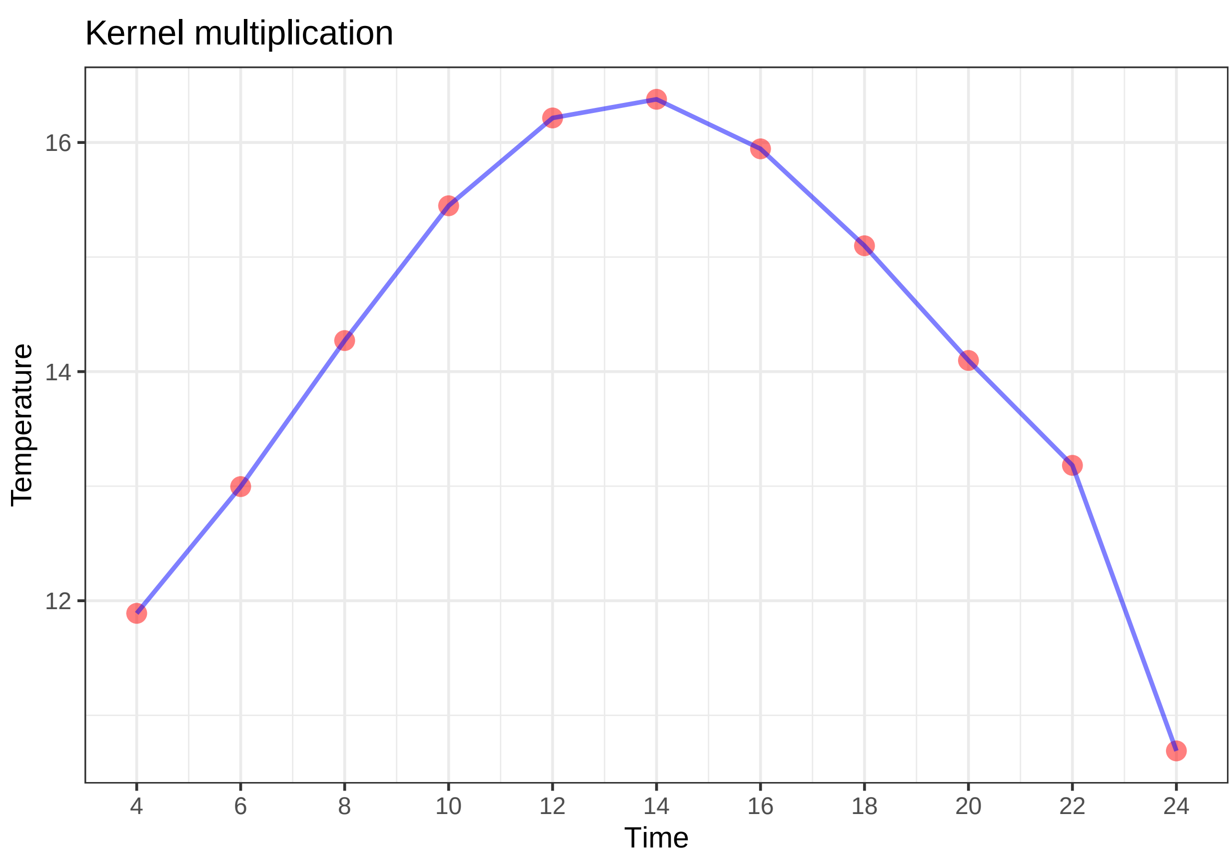


Q2

```
cached_mult = joined_rdd.map(lambda x: (getKernel(h_date, dateDist(date, x[1][0])) * \
                                        getKernel(h_distance, haversine(long, lat, x[2][1], x[2][0])),
                                        x[1][1],
                                        float(x[1][2]))).persist()
```

```
mult = []
for time in times:
    num, den = cached_mult.map(lambda x: (x[0] * getKernel(h_time, timeDist(time, x[1])), x[2])) \
                            .map(lambda x: (x[0]*x[1], x[0])) \
                            .reduce(lambda x,y: (x[0]+y[0], x[1]+y[1]))
    mult.append(num/den)

print(mult)
sums_file = open("mult.txt", "w+")
sums_file.write(str(mult))
sums_file.close()
```

```
mult_forecast <- c(11.8899, 12.9963, 14.2709, 15.4477, 16.2142, 16.3770, 15.9446, 15.0984, 14.0975, 13.
mult_forecast <- data.frame(time = seq(4,24,2), temp = mult_forecast)


library(ggplot2)
ggplot(mult_forecast, aes(x = time, y = temp)) +
  geom_point(color="red", alpha=0.5, size=3) +
  geom_line(color="blue", alpha=0.5, size=0.8) +
  labs(title="Kernel multiplication",
       x = "Time", y = "Temperature") +
  scale_x_continuous(breaks = seq(4,24,2)) +
  theme_bw()
```



- Multiplying is more sensetive to changes in h values, because every kernel is affected by the others, which means choosing a higher value of h for one kernel can inflate the other kernels and require more tuning. In summation each kernel is not affected by the others so we can get reasonable results only from one kernel even if the other two perform poorly.