Lab 3 - Report

Jonathan Reimertz & Viktor Gustafsson

2020-05-19

Width of Gaussian kernels

For this lab a kernel model was used to predict hourly temperatures for a date and place in Sweden. The used kernel is the sum of three Gaussian kernels accounting for the following:

- Physical distance
- Time of the day
- Time of the year

For each gaussian Kernel a smoothing coefficient (width) was chosen as decided appropriate. Below the chosen *widths* are presented and discussed.

The three h-values for the smoothing coefficients used was 100 km (*physical distance*), 3 hours (*time of the day*) and 25 days (*time of the year*). This implicates that a reading 100 km away from the place of interest weights the same as a reading with a time difference of 3 hours from the time of interest or a reading 25 days apart from the day of interest. Presumed that other variables are constant.

Physical distance

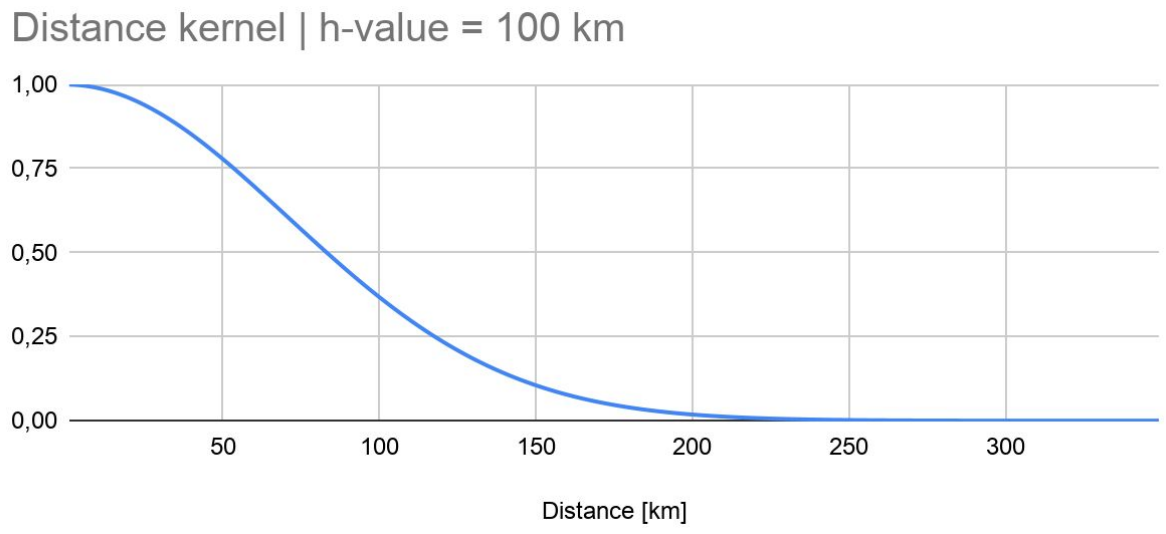

Figure 1. *Illustration of distance-kernel*

The h-value for the distance kernel was chosen to 100 km. As we can see, readings over ~150 km are barely used for prediction. We also consider a kernel value at ~ 0.36 for 100 km sensible, as climate can vary across these distances and only a small correlation should be assumed for readings with distances above 100km.

Time of the day

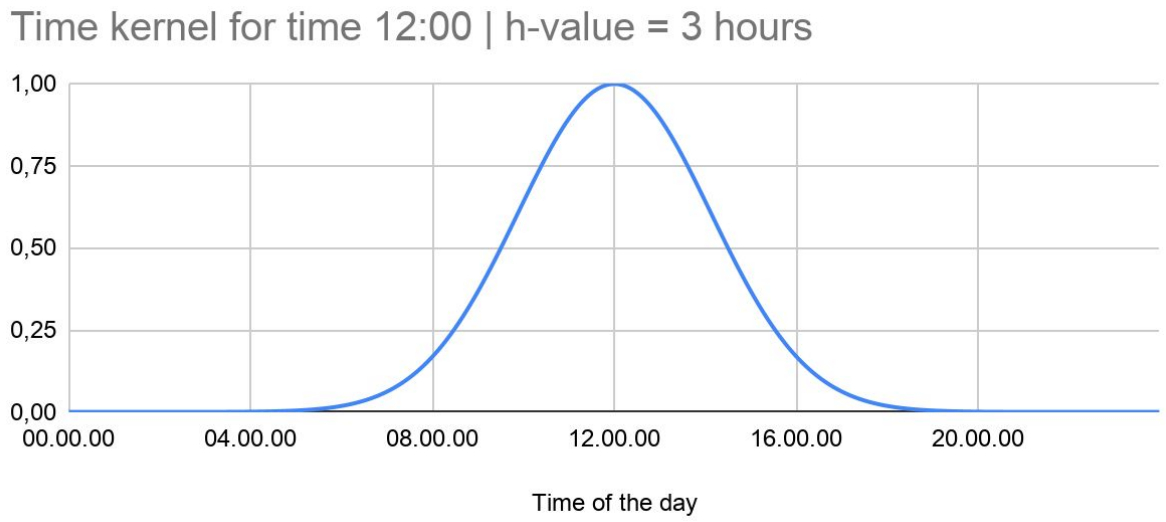

Figure 2. *Illustration of time-kernel*

The h-value for the time kernel was chosen to 3 hours. The time kernel mainly considered readings within 5 hours apart when predicting for a new time. This seems sensible as the temperature can vary a lot across the day.

Time of the year

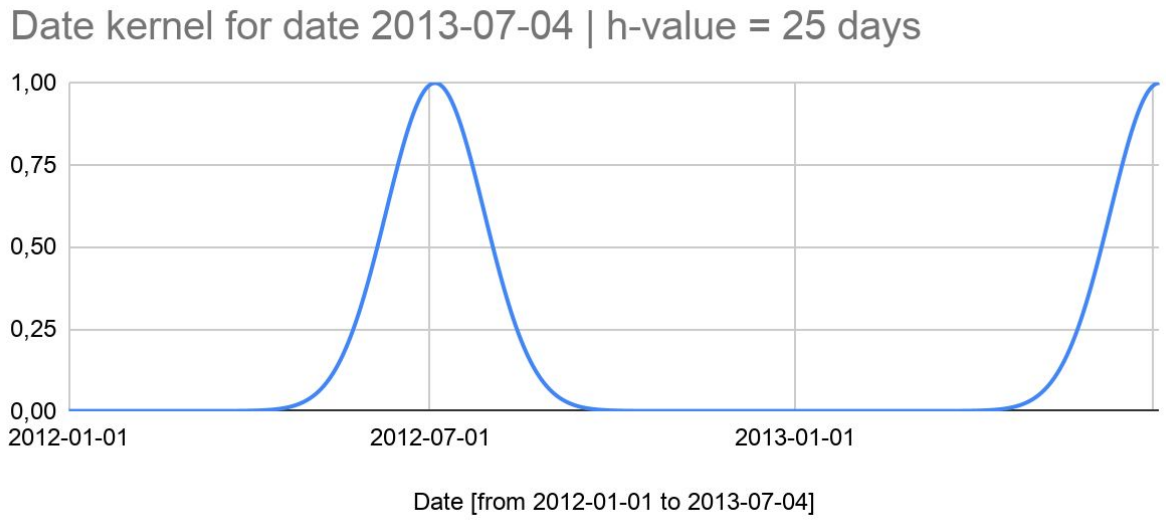

Figure 3. *Illustration of date-kernel*

The h-value for the date kernel was chosen to 25 days, which is illustrated in Figure 3 above. Readings from about one month apart are mainly considered. This is sensible, as the possible seasonal changes of across one month to another makes comparing these readings unreliable.

Sum of Gaussian kernels

The date predicted was 2013-07-04 with longitude and latitude 58.4274, 14.826 (Vadstena N). The resulting values are shown below in Figure 4. One can observe that the values are quite low for june. The shape of the curve seems right for the pattern that one can expect for the temperature during a summer day, hottest at midday and coldest at night.

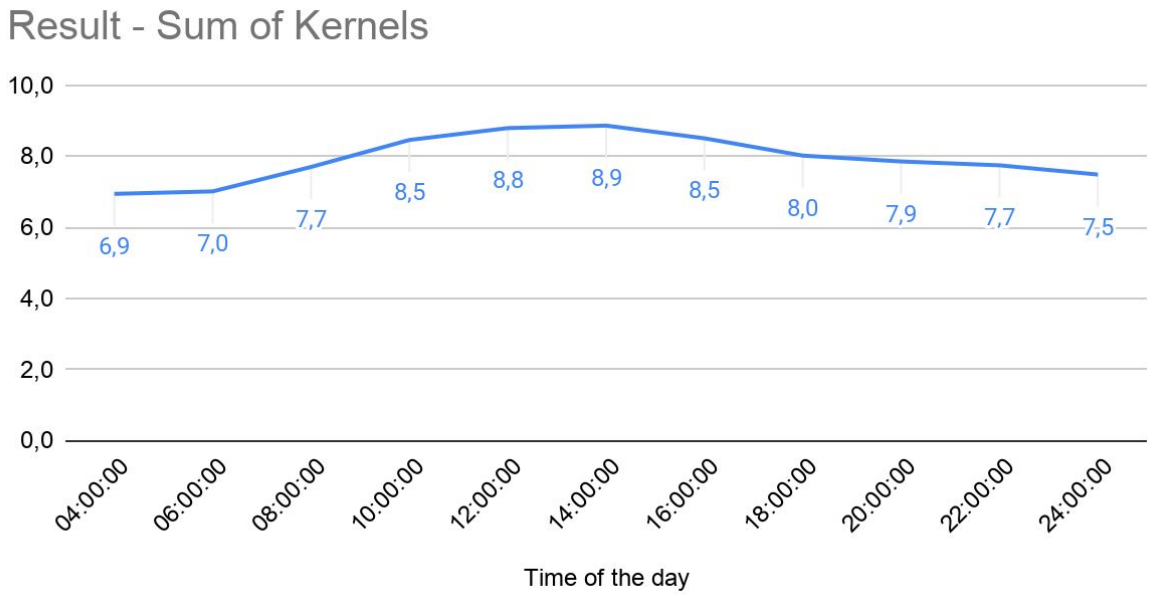

Figure 4. *Temperature prediction by summing gaussian kernels.*

Product of Gaussian kernels

The date predicted was 2013-07-04 with longitude and latitude 58.4274, 14.826. The resulting values are shown below in Figure 5. The temperature curve's appearance is in accordance with our prior expectancy. The temperature levels also seems reasonable for a summer day in Sweden, albeit on the lower end with ~18 degrees celsius as the peak value.

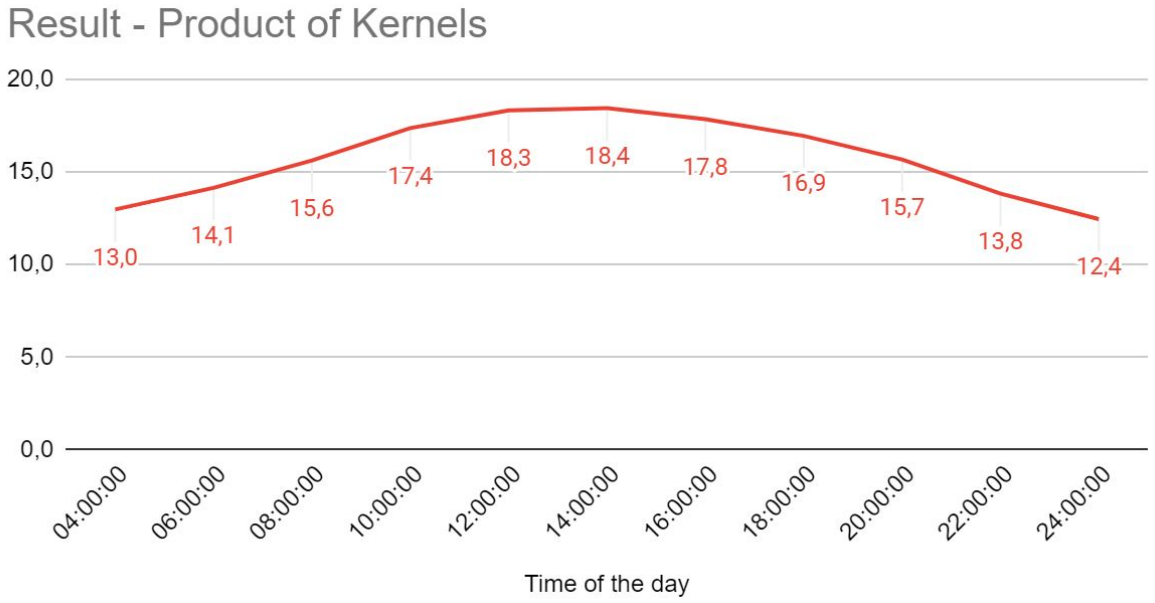

Figure 5. *Temperature prediction by multiplying gaussian kernels.*

Comparison of Results

The product of kernels seems to be a better model compared to a model using sum of kernels. This is much due to how that the sum of kernels is predicting lower temperatures, which does not seem realistic based on one's own experience.

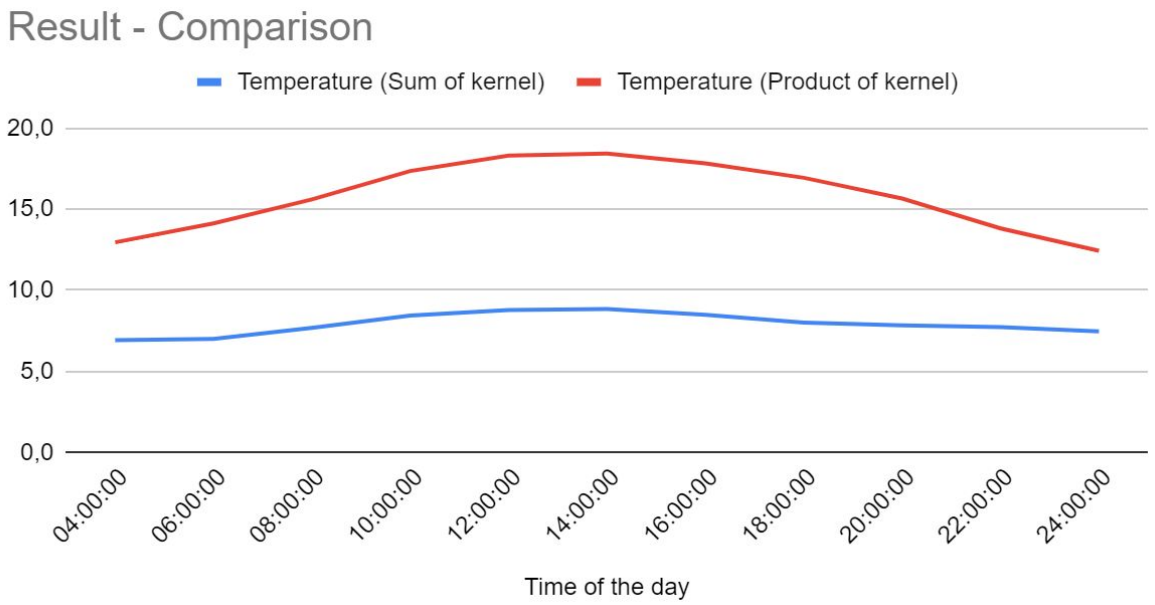

Figure 6. *Comparison of results from summing and multiplying gaussian kernels.*

This difference is most likely because that a model using sum of kernels considers lesser weighted values relatively more compared to a model using the product of kernels. For example, consider two readings with values according to Table 1.

Table 1. *Example values*

Reading #	Kernel value (distance)	Kernel value (time)	Kernel value (date)	Temperature [°C]
1	0,8	0,8	0,4	18
2	0,4	0,4	0,2	9

From the readings above it is possible to calculate the sum of sum and product of the kernels (weight), the weighted temperature and the predicted temperature for each of the two models as presented in Table 2 and Table 3 below.

Table 2. *Calculations of a model using sum of kernels.*

Reading #	Sum of kernels	Weighted temperatures	Predicted temperature [°C]
1	2	36	15
2	1	9	

Table 3. *Calculations of a model using product of kernels.*

Reading #	Product of kernels	Weighted temperatures	Predicted temperature [°C]

1	0,256	4,608	17
2	0,032	0,288	

The results are showing that at model using product of kernels is weighting the better reading relatively higher to the the model using sum of kernels. From the results we can also calculate how much each reading is taken into account when predicting the temperature by using Equation 1 and Equation 2.

$T_i = Temperature\ of\ reading\ i$
$w_1 = Weight\ of\ reading\ i$
$P = Predicted\ temperature$

$$T_1 \cdot w_1 + T_2 \cdot w_2 = P \qquad \text{Equation 1}$$

$$w_1 + w_2 = 1 \qquad \text{Equation 2}$$

Solving this equation for both of the models it is visible that the model using sum of kernels is accounting for the better reading **twice** as much compared to the lesser. The model using product of kernels is accounting for the better reading **eight times** the more compared to the lesser reading.

Table 4. *Result of weight-equations.*

Weight	Model using sum of kernels	Model using product of kernels
W1	66,7%	88,9%
W2	33,3%	11,1%

Appendix I - PySpark 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="lab_kernel")

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


# Compare dates for filtering
def date_comp(x, date):
    x = datetime.strptime(x, "%Y-%m-%d")
    date = datetime.strptime(date, "%Y-%m-%d")
    return x <= date

# Compare times for filtering
def time_comp(xdate, date, xtime, time):
  if (time == "24:00:00"):
    time = "00:00:00"
  xtime = datetime.strptime(xtime, "%H:%M:%S")
  time = datetime.strptime(time, "%H:%M:%S")
  if ((xdate == date) & (time != "00:00:00")):
    boolean = xtime < time
  else:
    boolean = True
  return boolean

# Number of days between two dates
from datetime import datetime
def date_diff(x, date):
    d1 = datetime.strptime(x, "%Y-%m-%d")
    d2 = datetime.strptime(date, "%Y-%m-%d")
```

```
    tot_days = abs((d2 - d1).days)
    diff = round(tot_days % 365.25)
    if(diff>182):
      diff=365-diff
    return diff

# Number of hours between two times
def time_diff(x, time):
  if (time == "24:00:00"):
    time = "00:00:00"
  t1 = datetime.strptime(x, "%H:%M:%S")
  t2 = datetime.strptime(time, "%H:%M:%S")
  diff = abs((t2-t1).total_seconds()/3600)
  if(diff>12):
    diff=24-diff
  return diff

# Kernel-value
def k_value(diff, smoothing_koefficient):
  u = diff/smoothing_koefficient
  k = exp(-(u**2))
  return k

# Smoothing coefficient
h_distance = 100
h_date = 25
h_time = 3

# Place and date of interest
a = 58.4274
b = 14.826
date = "2013-07-04"

# Read data from file
stations = sc.textFile("BDA/input/stations.csv")
temps = sc.textFile("BDA/input/temperature-readings.csv")
s_lines = stations.map(lambda line: line.split(";"))
t_lines = temps.map(lambda line: line.split(";"))

# Stations file structure: (Station, name, measurementHeight, latitude, longitude, readingsFrom,
readingsTo, elevation)
stations_data = s_lines.map(lambda x: (str(x[0]), (float(x[3]), float(x[4]))))
bc = sc.broadcast(stations_data.collectAsMap())

# Temperature file structure: (Station, YYYY-MM-DD, HH:MM, temp, quality)
data = t_lines.map(lambda x: ((str(x[0]), str(x[1]), str(x[2]),
                              bc.value[str(x[0])][0], bc.value[str(x[0])][1]) ,
                             float(x[3])))

#  Data (Key, Value) = ((Station number, YYYY-MM-DD, HH:MM, lat, lon) , temp)
```

```
# Filter posterior dates, keep date of interest and previous dates.
filteredByDate = data.filter(lambda x: (date_comp(x[0][1], date)))

# Calculate values for date and distance kernels.
loop_data = filteredByDate.map(lambda x: (x[0],
                                (k_value(date_diff(x[0][1], date), h_date),
                                 k_value(haversine(a,b, x[0][3],x[0][4]), h_distance),
                                 x[1])))
#Loop_data (Key, Value) = ((Station, Date, Time, lat, lon), (k_date, k_distance, temp))

# Loop data is saved in memory
loop_data.cache()

for time in ["24:00:00", "22:00:00", "20:00:00", "18:00:00", "16:00:00", "14:00:00",
          "12:00:00", "10:00:00", "08:00:00", "06:00:00", "04:00:00"]:

  # Filter posterior times of the date of interest
  loop_data = loop_data.filter(lambda x: (time_comp(x[0][1], date, x[0][2], time)))
  # Calculate value for time kernel.
  data_temp = loop_data.map(lambda x: (x[0],
                                  (x[1][0], x[1][1],
                                  k_value(time_diff(x[0][2], time), h_time),
                                  x[1][2])))
  # Data_temp (Key, Value) = ((Station, Date, Time, lat, lon), (k_date, k_distance, k_time, temp))

  # Sum and multiply kernels for each reading.
  data_sum = data_temp.map(lambda x: (x[0],
                                (x[1][0] + x[1][1] + x[1][2],
                                x[1][0] * x[1][1] * x[1][2],
                                x[1][3])))
  # Data_sum (Key, Value) = ((Station, Date, Time, lat, lon), (kernel_sum, kernel_product, temp))

  # Calculate "weighted" temperature for each reading.
  # Map with current time as key to enable reduceByKey in the next step.
  data_sum = data_sum.map(lambda x: (time,(x[1][0], x[1][0] * x[1][2], x[1][1], x[1][1] * x[1][2])))
  # Data_sum (Key, Value) = (Time, (kernel_sum, temp_sum, kernel_product, temp_product))

  # Predict temperature for time of interest, for both models.
  summed_data = data_sum.reduceByKey(lambda a,b: (a[0]+b[0], a[1]+b[1], a[2]+b[2], a[3]+b[3]))
  summed_data = summed_data.map(lambda x: (x[0], x[1][1]/x[1][0], x[1][3]/x[1][2]))

  # Add result to RDD for output.
  if (time == "24:00:00"):
    result = summed_data
  else:
    result = result.union(summed_data)

result.coalesce(1, shuffle = False).saveAsTextFile("BDA/output")
```

Appendix II - Results from Code execution

Below is the result in the format:
(Time, Predicted temperature using sum of kernels, Predicted temperature using product of kernels)

('24:00:00', 7.493928123888683, 12.513611783504148)
('22:00:00', 7.751396938429644, 13.879416198925329)
('20:00:00', 7.857925566503087, 15.709293781226227)
('18:00:00', 8.020825077590198, 16.983746589894018)
('16:00:00', 8.507322677921557, 17.870334169121552)
('14:00:00', 8.865399557793832, 18.408684773579065)
('12:00:00', 8.80062408073727, 18.254040061326435)
('10:00:00', 8.467468392959827, 17.31296804860092)
('08:00:00', 7.705534894081659, 15.606207463192062)
('06:00:00', 7.020651710432002, 14.16960364917795)
('04:00:00', 6.955149391140815, 13.022097414862289)

Table 5. *Output represented in table.*

Time of the day	Temperature (Sum of kernel)	Temperature (Product of kernel)
04:00:00	6,9	13,0
06:00:00	7,0	14,1
08:00:00	7,7	15,6
10:00:00	8,5	17,4
12:00:00	8,8	18,3
14:00:00	8,9	18,4
16:00:00	8,5	17,8
18:00:00	8,0	16,9
20:00:00	7,9	15,7
22:00:00	7,7	13,8
24:00:00	7,5	12,4