TDDE31 - Lab 3

Jesper Svensson (jessv553),
Oskar Holmström (oskho486)

15 May, 2020

1 Results and reflection

The kernel is a similarity function and is used to find the similarities between our data point of interest and the underlying data. In this lab three different kernels are implemented with the Gaussian kernel function. They are based on distance to a station, distance to a date, and distance to a time of day. For each kernel a width is chosen, which will determine the closeness of an observations to a chosen data. The decided width is based on observations of the cut-off point, were the resulting value is 0, and on our personal expectations of how temperature changes with the geography, date and time.

The following kernel widths were chosen, presented with their corresponding cut-off point:

	Kernel width (σ)	Cut-off
Station distance	30km	70km
Date distance	2 days	5 days
Time distance	1.5 hours	4 hours

It is our belief that the a cut-off points matches our prior beliefs of where differences in temperature should be noticeable: 70 km for station distance should capture geographic temperature differences, in a 5 days time it is reasonable to observe temperature changes, and a difference in 4 hours should capture the shifting temperature of the day as the sun rises and sets.

The kernel values and cut-off can be observed in figure 1, 2, and 3.

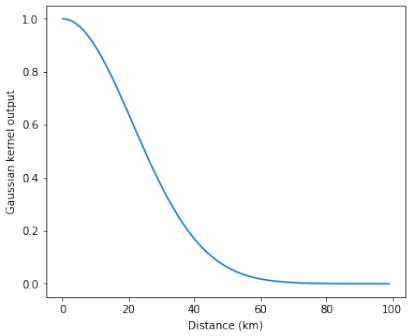

Figure 1: Gaussian kernel for station distance with $\sigma = 30$, resulting in cut-off at 70 km.

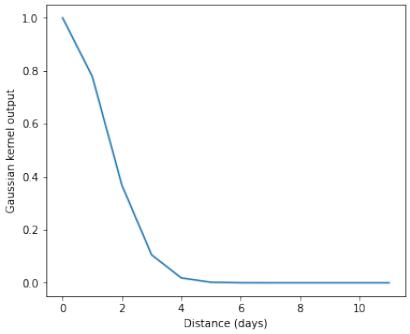

Figure 2: Gaussian kernel for date distance with $\sigma = 2$, resulting in cut-off at 5 days.

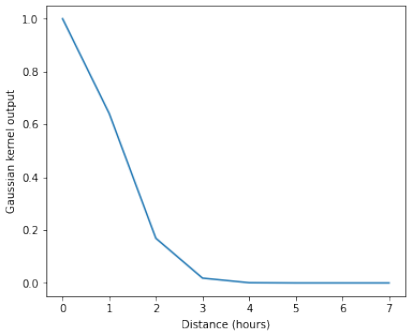

Figure 3: Gaussian kernel for time distance with $\sigma = 1.5$, resulting in cut-off at 4 hours.

The date of interest is 2013-07-04 and the location is in the vicinity of Vadstena (58.4274,14.826). The results can be observed in Table 1 and in Figure 4. Given our experience of temperatures in July in Östergötland the predicted temperatures from the product of kernels give a closer estimation than the sum of kernels. This is due to the three kernels being independent of each other in the sum of kernels, and dependent on each other in the product of kernels. In the sum of kernels a zero score from one of the kernels will not affect the result from the other two kernels. Therefore the prediction will, as in case of our results, give weight to observations from stations that are close to Vadstena but take place in the winter months. This will give a lower predicted temperature. The product of kernels would in opposite only give weight to those observations where all of the kernels differ from zero, i.e. where there is closeness in all three dimensions.

	Temperature	
Time of day	Sum of kernels	Prod. of kernels
22:00:00	3.72	15.07
20:00:00	4.38	16.25
18:00:00	5.10	17.65
16:00:00	6.08	18.40
14:00:00	6.69	18.43
12:00:00	6.66	17.80
10:00:00	5.96	16.89
08:00:00	4.69	15.75
06:00:00	3.34	14.58
04:00:00	2.90	13.49
00:00:00	2.91	13.53

Table 1: Predicted temperatures for different times of day, and for different methods of combining kernels.

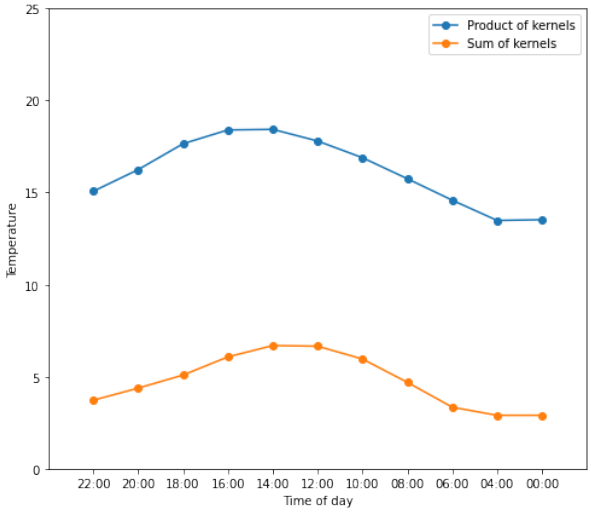

Figure 4: Comparison between predicted temperature from a product of kernels and sum of kernels.

2 Code

```
from __future__ import division
from math import radians, cos, sin, asin, sqrt, exp
from datetime import datetime, date
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


def gaussian_kernel(data, h):
    return (exp(-(data/h)**2))


def kernel_station_distance(cords, obs_cords, h_dist):
    """
    Calculate the haversine distance between two coordinates
    and return the corresponding value from a gaussian curve with
    the standard deviation h_dist.
    """
    distance_haversine = haversine(cords[0], cords[1], obs_cords[0]
                                       , obs_cords[1])
    return gaussian_kernel(distance_haversine, h_dist)


def kernel_date_distance(date_str, obs_date_str, h_date):
    """
    Calculate the absolute between two dates and return the
    corresponding value from a gaussian curve with
    the standard deviation h_dist.
    """
    date = datetime(int(date_str[0:4]), int(date_str[5:7]), int(
                                       date_str[8:10]))
    obs_date = datetime(int(obs_date_str[0:4]), int(obs_date_str[5:
                                       7]), int(obs_date_str[8:10]))
    delta = abs(date - obs_date)
    delta = 365-delta.days if delta.days > 365/2 else delta.days
    return gaussian_kernel(delta, h_date)
```

```
def kernel_time_distance(time_str, obs_time_str, h_time):
    """
    Calculate the distance in time between two times of day
    and return the corresponding value from a gaussian curve
    with the standard deviation h_dist.
    """
    time = datetime(2020, 5, 13, int(time_str[0:2]), int(time_str[3
                                    :5]), int(time_str[6:7]))
    obs_time = datetime(2020, 5, 13, int(obs_time_str[0:2]), int(
                                    obs_time_str[3:5]), int(
                                    obs_time_str[6:7]))
    delta = abs(time - obs_time)
    delta = delta.total_seconds()/3600.0
    return gaussian_kernel(delta, h_time)


# Choice of kernel widths so that the cut-off for
# the kernels (return value == 0) is aligned with
# expectations of temperature.
h_distance = 30
h_date = 3
h_time = 1.5

# Choice of coordinates and date to find
# temperature for.
latitude = 58.4274
longitude = 14.826
cords = (latitude, longitude)
date = "2013-07-04"

# Read the station data, split into rows and create an
# RDD with (station_num, (latitude, longitude).
station_data = sc.textFile("BDA/input/stations.csv")
station_rows = station_data.map(lambda line: line.split(";"))
stations = station_rows.map(lambda x: (int(x[0]), (float(x[3]),
                                float(x[4]))))

# Save the stations as a map (key: station_num, val: (lat, lon))
# and broadcast it to make it available to all clusters.
stations = sc.broadcast(stations.collectAsMap())

# Read the temperature data, split into rows and create an
# RDD with (station_numb, date, time, temp)
temp_data = sc.textFile("BDA/input/temperature-readings.csv")
temp_rows = temp_data.map(lambda line: line.split(";"))
temps = temp_rows.map(lambda x: (int(x[0]), x[1], x[2], float(x[3])
                                ))

# Remove all observerations where the date is past the
# date we want to find the temperature for.
temps_filtered = temps.filter(lambda x: int(x[1][0:4]) <= int(date[
                                0:4]) and int(x[1][5:7]) <= date[
                                5:7] and int(x[1][8:10]) < date[8
                                :10])
```

```
# Calculate the kernel values for station and date distance.
temps_filtered = temps_filtered.map(lambda x: (
                                    kernel_station_distance(cords,
                                    stations.value[x[0]], h_distance)
                                    , kernel_date_distance(date, x[1]
                                    , h_date), x[2], x[3]))

# Cache the data, in ordet to not read all
# the into an RDD again.
temps_filtered = temps_filtered.cache()


for time_string in ["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", "00:00:00"]:

    #temps_filtered = temps_filtered.filter(lambda x: (x[1][0] <
                                        date) or (x[1][0] <= date)
                                        and (x[1][1] < time_obj))

    # Calculate the kernel value for time distance.
    station_kernels = temps_filtered.map(lambda x: (x[0],  x[1],
                                        kernel_time_distance(
                                        time_string, x[2], h_time), x
                                        [3]))

    # Calculate the sum and product of the three kernels.
    station_kernels = station_kernels.map(lambda x: (x[0]+x[1]+x[2]
                                        , x[0]*x[1]*x[2], x[3]))

    # Multiply the sum of kernels and product of kernels with
    # the true temperature for each observation.
    station_kernels = station_kernels.map(lambda x: (x[0], x[1], x[
                                        0]*x[2], x[1]*x[2]))

    # Sum the sum of kernels, product of kernels, sum of kernels *
                                        temperature,
    # and prod of kernels * temperature
    station_kernels = station_kernels.reduce(lambda v1, v2: (v1[0]+
                                        v2[0], v1[1]+v2[1], v1[2]+v2[
                                        2],v1[3]+v2[3]))

    # Calculate the predicted temperature given the sum of kernels
                                        and
    # prod of kernels respectively.
    pred_temp =  (station_kernels[2]/station_kernels[0],
                                        station_kernels[3]/
                                        station_kernels[1])
    print(pred_temp)
```