BigDataAnalysis BDA3

Helena Llorens Lluís (hllor282) Yi Yang (yiyan338)

May 2025

1 ASSIGNMENT

Implement in Spark (PySpark) a kernel model to predict the hourly temperatures for a date and place in Sweden. To do so, you should use the files temperature-readings.csv and stations.csv from previous labs. Specifically, the forecast should consist of the predicted temperatures from 4 am to 24 pm in an interval of 2 hours for a date and place in Sweden. Use a kernel that is the **sum** of three Gaussian kernels:
The first to account for the distance from a station to the point of interest.
The second to account for the distance between the day a temperature measurement was made and the day of interest.
The third to account for the distance between the hour of the day a temperature measurement was made and the hour of interest.
Choose an appropriate smoothing coefficient or width for each of the three kernels above. You do not need to use cross-validation.

- Questions

1.Show that your choice for the kernels' width is sensible, i.e. it gives more weight to closer points. Discuss why your definition of closeness is reasonable.

The chosen bandwidths for the Gaussian kernels are:

(1) $h_{\text{distance}} = 200\,\text{km}$:
This value allows nearby stations ($\leq$ 200 km) to significantly influence the estimate, while those beyond 400 km have little to no effect. This is useful in Sweden, where stations may be somewhat sparse but still relatively regional in climate characteristics.

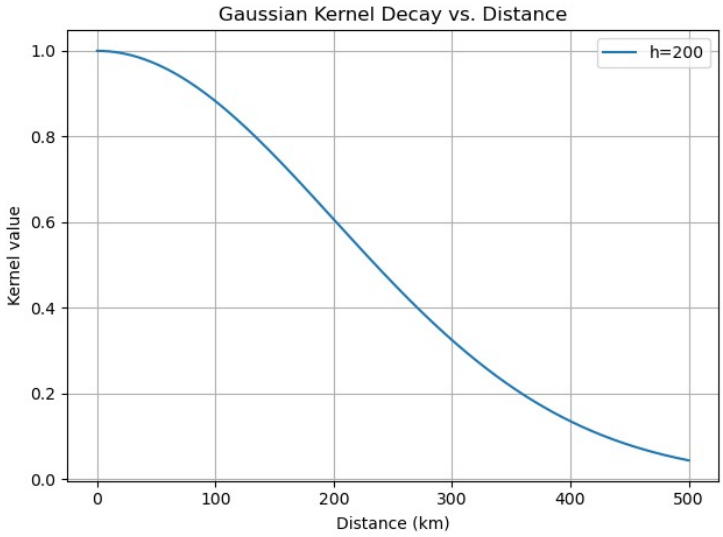

Figure 1: Kernel vs distance difference

(2)$h_{\text{date}} = 20$ days:
This value allows the model to focus primarily on measurements from a $\pm$ 30 day seasonal window, which captures local seasonality, while discounting irrelevant seasonal extremes like comparing July to January.

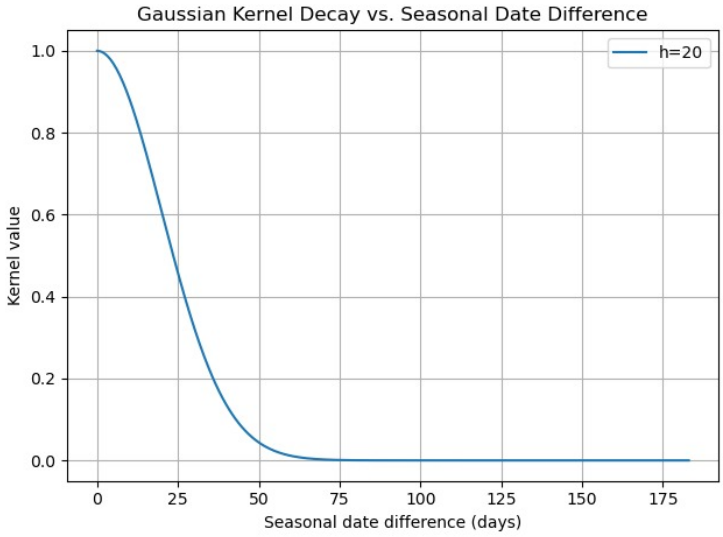

Figure 2: Kernel vs date difference

(3)$h_{\text{time}} = 2$ hours:
With this value, only measurements within $\pm$ 3 hours of the target time meaningfully affect the result. This is important because diurnal temperature variation is high—temperatures at noon differ significantly from those at night. This parameter keeps the estimate time-local.

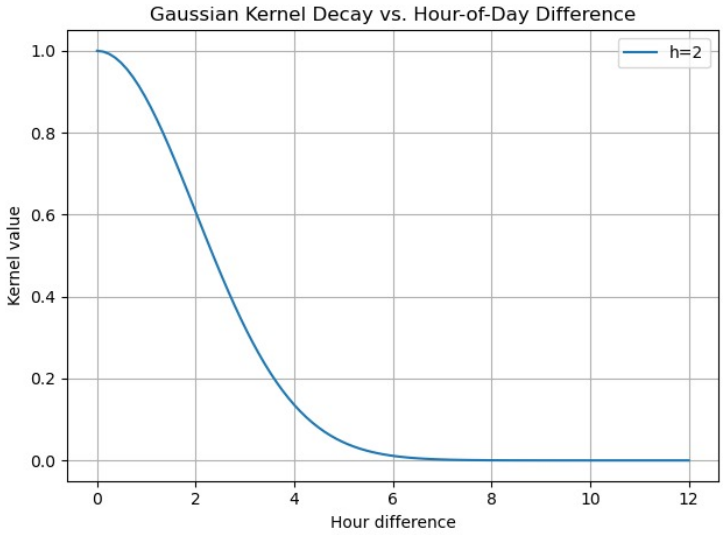

Figure 3: Kernel vs hour difference

- Predictions

```
[
    ('22:00:00', 5.407921709332834)
    ('20:00:00', 5.490161619334796)
    ('12:00:00', 5.8124982097379725)
    ('18:00:00', 5.584785851505983)
    ('04:00:00', 5.255417705289861)
    ('16:00:00', 5.707198861741724)
    ('14:00:00', 5.8053912723052585)
    ('06:00:00', 5.3002249861067146)
    ('10:00:00', 5.696720312117955)
    ('24:00:00', 5.334816321978539)
    ('08:00:00', 5.484662234153178)
]
```

2.Repeat the exercise using a kernel that is the product of the three Gaussian kernels above. Compare the results with those obtained for the additive kernel. If they differ, explain why.

- Predictions

```
[
    ('22:00:00', 12.582276858022263)
    ('20:00:00', 14.331445450487692)
    ('12:00:00', 16.972574499314966)
    ('18:00:00', 15.604706033384101)
    ('04:00:00', 12.1535556004538)
    ('16:00:00', 16.450578096822667)
    ('14:00:00', 17.007376945523802)
    ('06:00:00', 13.35982280999343)
    ('10:00:00', 16.215722747358246)
    ('24:00:00', 11.40789623515976)
    ('08:00:00', 14.724933699841438)
]
```

- Compared the results

The plot of temperature prediction for 2013-07-04 using two different methods shows as follows:

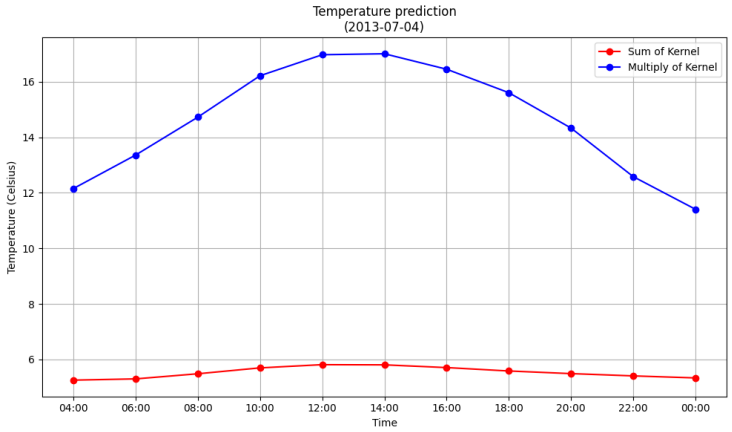

Figure 4: Temperature Prediction Comparison

The temperature predictions using the sum of the three Gaussian kernels range from approximately 5.255 to 5.812 degrees Celsius, with the highest prediction at 12:00. The predicted values show relatively smooth variations, resulting in a gentle and stable prediction curve.

In contrast, the temperature predictions obtained by multiplying the three Gaussian kernels have a much wider range, from about 11.408 to 17.007, with overall higher values and a steeper fluctuation in the prediction curve.

- Explanation

When we use the sum of kernels each factor (distance, date, hour) contributes independently to the similarity. This approach is more inclusive: even if one

of the kernels is low, the others can compensate. Therefore, more observations influence the prediction — leading to smoother and generally lower temperature estimates.

With the product of kernels, all three dimensions must be similar simultaneously for the observation to contribute significantly. If any kernel is close to zero, the whole product becomes small - meaning that only observations that are simultaneously close in space, date, and time influence the prediction. This makes the kernel much more selective, leading to sharper and typically higher temperature estimates.

- Code

```
#!/usr/bin/env python3

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)
    """
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
    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(dist, h):
    return exp(-(dist ** 2) / (2 * (h ** 2)))

def hour_of_aday(time):
    return int(time.split(":")[0])

def seasonal_date_diff(date1, date2):
    doy1 = date1.timetuple().tm_yday
    doy2 = date2.timetuple().tm_yday
    diff = abs(doy1 - doy2)
    return min(diff, 365 - diff)

# Parameters
h_distance = 200000
h_date = 20
h_time = 2
a = 58.4274  # Latitude of point of interest
b = 14.826   # Longitude of point of interest
target_date = "2013-07-04"
targetTime = datetime.strptime(target_date, "%Y-%m-%d")
```

```
# Load and prepare stations
stations = sc.textFile("BDA/input/stations.csv")
stations_lines = stations.map(lambda line: line.split(";"))
stations_lines = stations_lines.map(lambda x: (
    x[0], (
        float(x[3]),  # latitude
        float(x[4])   # longitude
    )))
stations_dict = dict(stations_lines.collect())
broadcast_stations = sc.broadcast(stations_dict)

# Load and process temperature data
temps = sc.textFile("BDA/input/temperature-readings.csv")
temp_lines = temps.map(lambda line: line.split(";"))
temp_lines = temp_lines.map(lambda x: (
    x[0],  # station number
    datetime.strptime(x[1], "%Y-%m-%d"),  # date
    x[2],  # time
    float(x[3]),  # temperature
    broadcast_stations.value.get(x[0], (None, None))[0],  # lat
    broadcast_stations.value.get(x[0], (None, None))[1]   # lon
)).filter(lambda x: x[4] is not None)

# Filter readings before the target date
temp_lines = temp_lines.filter(lambda x: x[1] < targetTime)

# Compute physical and date kernels
def physical_date_kernels(x):
    distance = haversine(b, a, x[5], x[4])
    # date_diff = (targetTime - x[1]).days
    date_diff = seasonal_date_diff(targetTime, x[1])
    dist_kernel = gaussian_kernel(distance, h_distance)
    date_kernel = gaussian_kernel(date_diff, h_date)
    return (x[0], x[1], x[2], x[3], dist_kernel, date_kernel)

kernelCache = temp_lines.map(physical_date_kernels).cache()

def hour_kernels_factory(time_str):
    target_hour = int(time_str.split(":")[0]) % 24

    def hour_kernels(x):
        input_hour = int(x[2].split(":")[0]) % 24
        hour_diff = abs(target_hour - input_hour)
        if hour_diff > 12:
            hour_diff = 24 - hour_diff
        hour_kernel = gaussian_kernel(hour_diff, h_time)
        #combined_kernel = x[4] + x[5] + hour_kernel
        combined_kernel = x[4] * x[5] * hour_kernel  # product of all kernels
        return (time_str, (x[3] * combined_kernel, combined_kernel))

    return hour_kernels


# Estimate temperature for different hours
results = []
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"]:

    hour_kernels = hour_kernels_factory(time)
    mapped = kernelCache.map(hour_kernels)
    reduced = mapped.reduceByKey(lambda a, b: (a[0] + b[0], a[1] + b[1]))
    estimated = reduced.mapValues(lambda x: x[0] / x[1] if x[1] != 0 else None)
    results.append(estimated)

# Combine results and output
joined_result = sc.union(results)

joined_result.saveAsTextFile("BDA/output/temperature_prediction")
```
