732A54 Big Data Analytics

Lab Exercise 3: Machine Learning

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

- 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.
- 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.

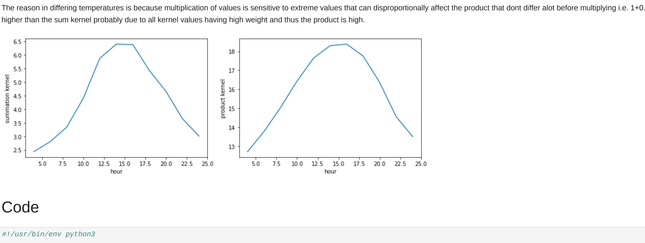
Chosen parameters

- h_distance = 1000
- h_date = 10
- h_time = 4

The distance is defined to 1000 as 1000 km away from the point should be a reasonably similar temperature. Further than that and there is room for a lot of differing. The same reasoning is applied to the date and time, where a 10 days difference seems like a reasonable cut off for temperatures to influence each other. Temperatures can vary a lot in a short time span, so 4 hours seems reasonable.

- latitude = 62.2857
- longitude = 15.3735
- date = 2013-06-24

The chosen point and date is at Enåsengruvan at Midsummer eve 2013. This spot is approximately the middle point of Sweden.



The plots above show how the kernel width gives importance to physical, hourly and daily distance away from the point and date of interest. The plot of the kernel is decreasing as one expects, where close points has high importance and then decreases well. After a reasonable distance passed for the measures the importance is negligible.

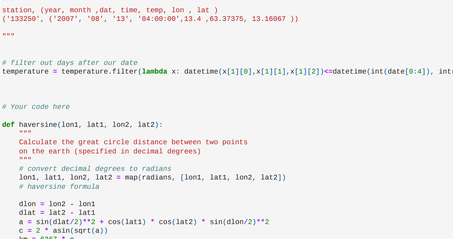
Result

The line plots below show the two kernels created for hourly temperatures of the point and time of interest. Both predictions show similar with the temperatures rising from morning until afternoon and then decreasing, however there is a large discrepancy in the temperature measured.

The product kernel considers the joint probability of all kernels, which means the product can still be significant even if a single kernel has low values.

Summing kernels emphasizes regions where kernels overap significantly instead, which makes those particular areas more significant.

The reason in differing temperatures is because multiplication of values is sensitive to extreme values that can disproportionally affect the product that dont differ alot before multiplying i.e. 1+0.1 vs 1*0.1. The product kernel is higher than the sum kernel probably due to all kernel values having high weight and thus the product is high.



Code

In [ ]:

```
#!/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")

# variables
h_distance = 1000 # as 1000 km away from this point should have about the same temperature
h_date = 10 # ten days diff should affect temperature at the current day
h_time = 4 # 4 hours diff for the current hour
a = 62.2857 # middle of sweden
b = 15.3735 # middle of sweden ish
date = "2013-06-24" # Midsommer that year

stations = sc.textFile("BDA/input/stations.csv")
temps = sc.textFile("BDA/input/temperature-readings.csv")

lines_temp = temps.map(lambda line: line.split(";")) # splitting the data by ;

lines_stat = stations.map(lambda line: line.split(";"))

# station(key) long lat
station = lines_stat.map(lambda x: (x[0],(float(x[3]), float(x[4]))))

# broadcasting the stations to all nodes so we dont have to join with temp as its inefficient.
station = station.collectAsMap()

stations_bc = sc.broadcast(station)

# mapping the data with lon and lat from stations_bc, using x[0] as index to get the correct values from stations rdd
temperature = lines_temp.map(lambda x: (x[0],(int(x[1][0:4]),int(x[1][5:7]),int(x[1][8:10]),int(x[2][0:2]),float(x[3]),stations_bc.value[x[0]][0],stations_bc.value[x[0]][1])))


"""
station, (year, month ,dat, time, temp, lon , lat )
('133250', ('2007', '08', '13', '04:00:00',13.4 ,63.37375, 13.16067 ))

"""


# filter out days after our date
temperature = temperature.filter(lambda x: datetime(x[1][0],x[1][1],x[1][2])<=datetime(int(date[0:4]), int(date[5:7]), int(date[8:10])))



# Your code here

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
# day distance kernel

def day_dist(year,month,day,h):
    """Function that calculates distance in days from midsommer 2013"""

    # difference between the days
    daydiff = abs((datetime(int(date[0:4]), int(date[5:7]), int(date[8:10])) - datetime(year,month,day)).days)

    # returning the gaussian kernel
    return (exp(-(daydiff/h)**2))


def time_dist(time_test,time_mes,h):
    """ Function that measure the time difference in hours from given time-vector to previous hours"""
    time_diff = abs(time_test - time_mes)

    if time_diff >=13: # if previous cal is over 12 we switch the calculation so eg. 04:00-24:00 only is a 4 hour diff and not 20.
                time_diff = 24 - time_diff

    return (exp(-(time_diff/h)**2))


def physical_dist(obs_lon,obs_lat,new_lon,new_lat,h):

    """Function that calculate the physical distance """

    dist_phy = haversine(obs_lon,obs_lat,new_lon,new_lat)

    return (exp(-(dist_phy)/h)**2)

# station, time, temp, gaus_sum, gaus_prod
daily_physical = temperature.map(lambda x: (x[0],(x[1][3],x[1][4],
                physical_dist(a,b,x[1][4],x[1][5],h_distance) * day_dist(x[1][0],x[1][1],x[1][2],h_date),
                physical_dist(a,b,x[1][4],x[1][5],h_distance) * day_dist(x[1][0],x[1][1],x[1][2],h_date))))

"""
station, (time, temp, distance+day, distance*day)
"""
# saving the data so we dont have to re-do everything again
daily_physical.persist()

# Your code here
kernel_sum = list()
kernel_prod = list()
for time in [24,22,20,18,16,14,12,10,8,6,4]:

        # removing every "future" measured time.
        if time <=24:
                filt_time = daily_physical.filter(lambda x : x[1][0] <= time)
        else:
                filt_time = daily_physical

        # mapping and adding time_dist to our kernels
        filt_time = filt_time.map(lambda x:(time,(x[1][2]+time_dist(time,x[1][0],h_time),
                                                  x[1][2]+time_dist(time,x[1][0],h_time)*x[1][1],
                                                  x[1][3]*time_dist(time,x[1][0],h_time),
                                                  x[1][3]*time_dist(time,x[1][0],h_time)*x[1][1]
                                                  )))

        """
        Time, (temp, kernel_sum,ks*temp, kernel_prod,ks*temp)
        """
        # summing the results
        kernel_res =filt_time.reduceByKey(lambda x,y:(x[0]+y[0],x[1]+y[1],x[2]+y[2],x[3]+y[3]))

    # mapping the sum(kernel*temp)/sum(kernel)
        # appending the values in the list as (time, predicted temp)
        kernel_sum.append(kernel_res.map(lambda x:(x[0],x[1][1]/x[1][0])).collect())
        kernel_prod.append(kernel_res.map(lambda x:(x[0],x[1][3]/x[1][2])).collect())


kernel_sum = sc.parallelize(kernel_sum) # parrallize the list to save it
kernel_sum.saveAsTextFile("BDA/output/sum")

kernel_prod = sc.parallelize(kernel_prod)
kernel_prod.saveAsTextFile("BDA/output/prod")


# Visualizing kernels
import matplotlib.pyplot as plt

def kernel_plot(data, h, xlabel, figure_num):
    u = [d / h for d in data]
    res = [exp(-x**2) for x in u]
    plt.figure(figure_num)
    plt.plot(data, res)
    plt.xlabel(xlabel)
    plt.ylabel('kernel')

dist = list(range(0, 5000))
days = list(range(0, 50))
hours = list(range(0, 24))

h_distance = 1000
h_date = 10
h_time = 4

kernel_plot(dist, h_distance, 'distance', 0)
kernel_plot(days, h_date, 'days', 1)
kernel_plot(hours, h_time, 'hours', 2)


# Visualizing temperatures
times = [24,22,20,18,16,14,12,10,8,6,4]

prods = [13.510553054043463, 14.558531449832272, 16.358096565020862, 17.750289954487805, 18.379959028634054,
       18.300031152459393, 17.651113755802 4, 16.443244301074085, 15.048160588889353, 13.786652957936294,12.723081450446688]

sums = [3.011097416665485,3.6492685950797696,4.664154989855703,5.425272421652926,6.384644815671309,6.402708300048972,
        5.882371750729638,4.409092728155342,3.344292940765084 8,2.806482342133761,2.434104067210284 5]

plt.plot(times, prods)
plt.plot(times, sums)
```