Lab 3 Report

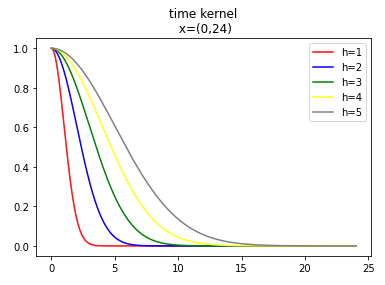
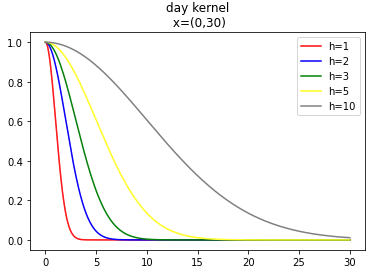
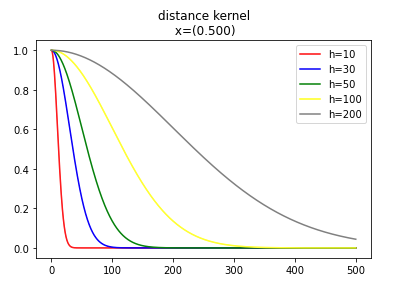
Group member:

Hong Zhang(honzh073)

Bing Rui Yao(binya518)

● 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.

Following graphs shows the curves of Gaussian kernels with different widths. The kernel values are approaching 0 for distance over 150, difference of day over 5, and difference of hours over 5. We define h\_distance=100, h\_day=3, and h\_hour=2 as optimal kernel width so that the kernel values for close points and far points are distinguishable.



● Use a kernel that is the sum of three Gaussian kernels.

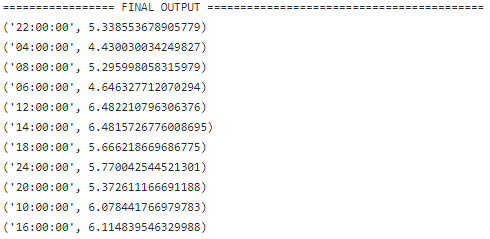
Input:

a = 58.4274

b = 14.826

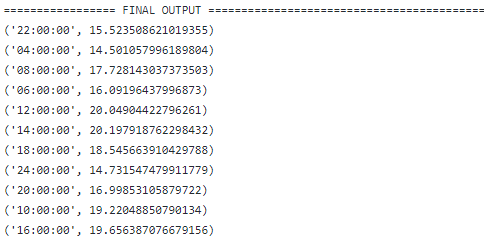
date = "2013-07-04"

Predictions:



● 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:



The reason for the different predictions is that the sum kernel gives mores weights to the points which meet at least one conditions, so the points with high weight in one of the three kernels and low in others can also gain high weights. But the product kernel only gives high weights to the points which meet all the conditions of ‘close points’, so the points which are strictly close to the input point contribute more in prediction.

Code Appendix:

from \_\_future\_\_ import division

from math import radians, cos, sin, asin, sqrt, exp

from datetime import datetime, timedelta

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

h\_distance = 100# Up to you

h\_date = 3# Up to you

h\_time = 2# Up to you

a = 58.4274 # Up to you

b = 14.826 # Up to you

date = "2013-07-04" # Up to you

stations = sc.textFile("BDA/input/stations.csv")

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

# Your code here

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

temp\_lines = temps.map(lambda line: line.split(";"))

#(station\_num,distance)

distance\_stations = stations\_lines.map(lambda x: (x[0],haversine(b,a,float(x[4]),float(x[3]))))

#broadcast distance data

broadcast\_distance = sc.broadcast(dict(distance\_stations.collect()))

# gaussian kernel

def gk(x,h):

return exp(-x\*\*2/(2\*h\*\*2))

# distance kernel

def kernel\_distance(station\_num):

return gk(broadcast\_distance.value[station\_num],h\_distance)

def kernel\_day(day\_1,day\_2):

return gk(abs((day\_1-day\_2).days)%365+round(abs((day\_1-day\_2).days/365)/4),h\_date)

def kernel\_hour(time\_1,time\_2):

time\_2 = time\_2.replace(year=time\_1.year,month=time\_1.month,day=time\_1.day)

diff=abs((time\_1-time\_2).total\_seconds())/3600

return gk(diff,h\_time)

#preprocess

#(station\_num,datetime,temperature)

temp\_data = temp\_lines.map(lambda x: (x[0],datetime(int(x[1][0:4]),int(x[1][5:7]),int(x[1][8:10]),int(x[2][0:2]),int(x[2][3:5]),int(x[2][6:8])),float(x[3])))

temp\_data = temp\_data.filter(lambda x: x[1]<=(datetime(int(date[0:4]),int(date[5:7]),int(date[8:10]),0,0,0)+timedelta(days=1))).cache()

# prediction

prediction\_temp={}

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"]:

#for time in ["20:00:00"]:

if time=="24:00:00":

datetime\_interest = datetime(int(date[0:4]),int(date[5:7]),int(date[8:10]),23,59,59)

else:

datetime\_interest = datetime(int(date[0:4]),int(date[5:7]),int(date[8:10]),int(time[0:2]),int(time[3:5]),int(time[6:8]))

#filter the dates which are posterior

#(station\_num,datetime,temperature)

temp\_filter = temp\_data.filter(lambda x: x[1]<datetime\_interest)

#(kernel\_all,temperatures)

kernel\_all = temp\_filter.map(lambda x: (kernel\_distance(x[0])+kernel\_day(datetime\_interest,x[1])+kernel\_hour(datetime\_interest,x[1]),x[2]))

#kernel\_all = temp\_filter.map(lambda x: (kernel\_distance(x[0])\*kernel\_day(datetime\_interest,x[1])\*kernel\_hour(datetime\_interest,x[1]),x[2]))

#(sum:kernel,sum:kernel \* temperature)

kernel\_sum = kernel\_all.map(lambda x: (x[0],x[0]\*x[1])).reduce(lambda a,b: (a[0]+b[0],a[1]+b[1]))

prediction\_temp[time]=kernel\_sum[1]/kernel\_sum[0]

sc.parallelize(prediction\_temp.items()).saveAsTextFile("BDA/output")