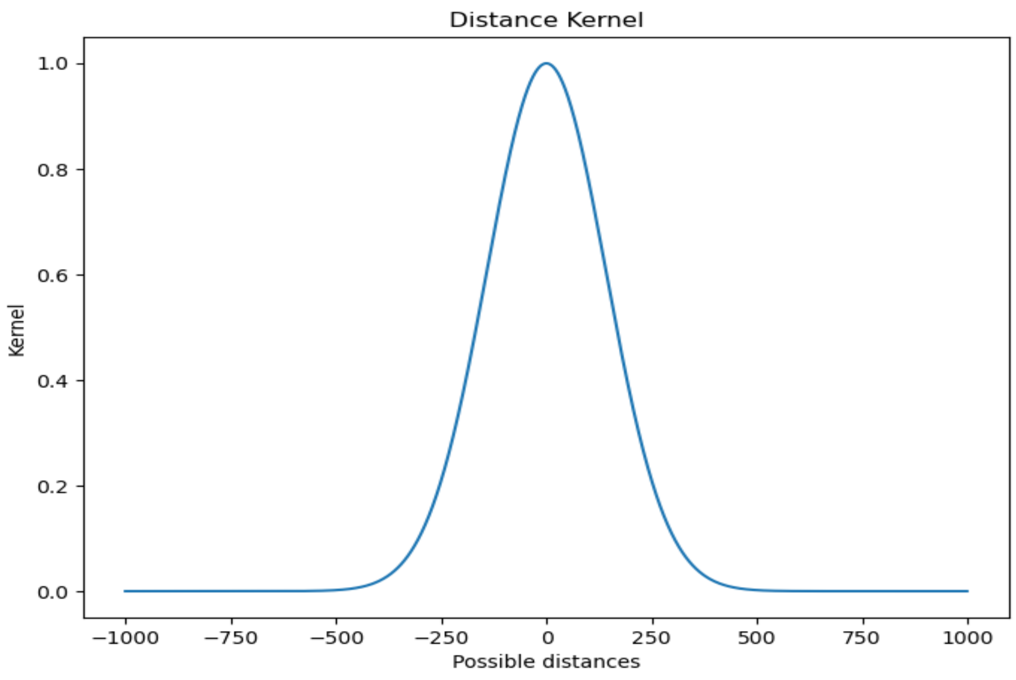
TDDE31 – Big data Analytics, Lab3

Priyansh Gupta – prigu857

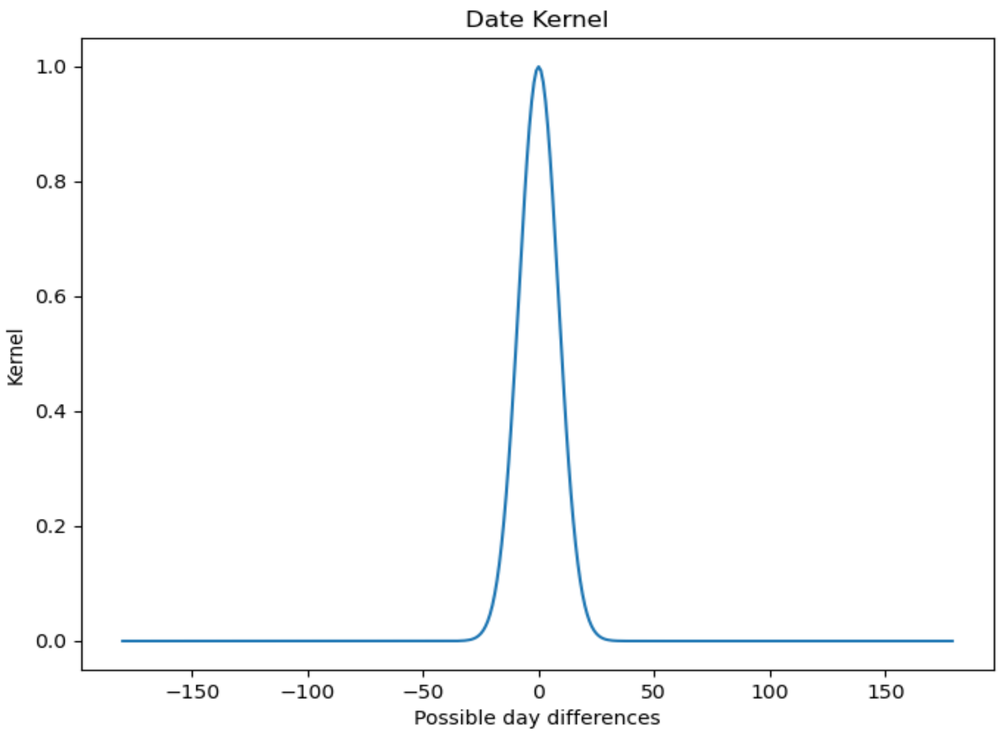
Lukas Floberg – lukfl677

Assignment:

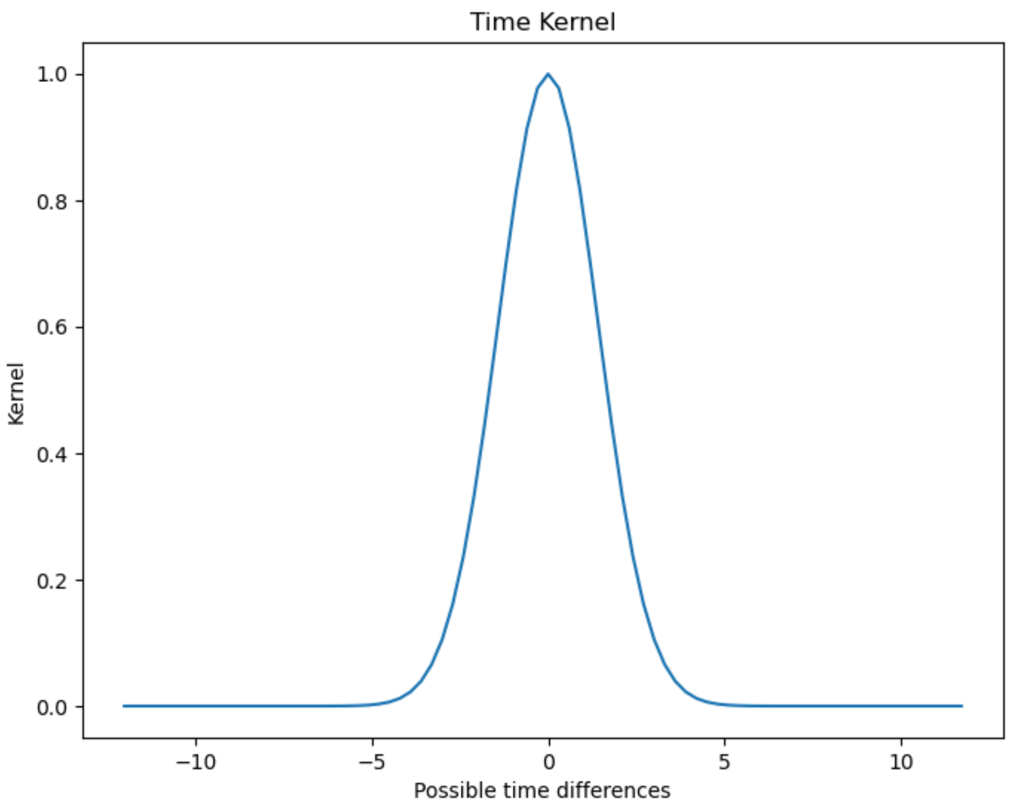
Distance Kernels:



Date Kernel:



Time Kernel:



H-parameters for different Kernels were choosen to make sure that the values which are far apart from the target value have a smaller effect on the prediction for temperature. Above plots represents the three different kernels which are used in prdiction.

Here are the other variables which are set to make the prediction:
h_distance = 200
h_date = 12
h_time = 2
a = 58.4274
b = 14.826
date = "2013-07-04

Output with sum of above three Kernel for prediction:
(Hour, Temperature)
(24, 6.629317081411721)
(22, 6.896759576473815)
(20, 7.0073417298704435)
(18, 7.0244350241361975)
(16, 7.585499284892816)

(14, 7.889051068724912)
(12, 7.771560521086171)
(10, 7.596757923279711)
(8, 6.968270988263838)
(6, 6.183511901554222)
(4, 6.32158773804262)

Multiplication of above three Kernels for prediction:
Multiplication will have an effect on the prediction that if even one kernel is close to zero, the multiplication of all three Kernels will be zero, thus eliminating the whole value. Thus, for prediction multiplication of kernel is more preferable rather than sum of the three kernels.

Output with multiplication of the above Kernels for prediction is as follow:
(Hour, Temperature)
(24, 12.75353953404971)
(22, 13.966654289084293)
(20, 16.13262132937899)
(18, 17.603629169797543)
(16, 18.559176282868666)
(14, 19.20987932585848)
(12, 19.102421285410877)
(10, 18.316669563462405)
(8, 16.39104943110443)
(6, 14.787139335036786)
(4, 13.3368692306381)

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")

#filter days
def filter_days(date, RDD):
    original_date = datetime(int(date[0:4]), int(date[5:7]), int(date[8:10]))#yyyy-mm-dd
    return(RDD.filter(lambda x: (datetime(int(x[0][1][0:4]),int(x[0][1][5:7]),
int(x[0][1][8:10]))<original_date)))

## COUNT Hours
def diff_hours(time1, time2):
    time_diff = abs(time1 - time2)
    if (time_diff > 12):
        return 24 - time_diff
    else:
        return time_diff

## Difference in days
def diff_days(date1, date2):
    d1 = datetime(int(date1[0:4]), int(date1[5:7]), int(date1[8:10]))#yyyy-mm-dd
    d2 = datetime(int(date2[0:4]), int(date2[5:7]), int(date2[8:10]))#yyyy-mm-dd
    diff = (d1 - d2)
    diff = diff.days % 365
    if diff > 182:
        return 365-diff
    else:
        return diff

def haversine(lon1, lat1, lon2, lat2):
    # 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 gKernel(diff, h):
    return(exp(-(diff/h)**2))

h_distance = 200
h_date = 12
h_time = 2
a = 58.4274 # Latitude
b = 14.826 # Longitude
date = "2013-07-04" # Date for predicting temperatures

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

line_temp = temps.map(lambda line: line.split(";"))
line_stations = stations.map(lambda line: line.split(";"))

#(station, la, lo)
stations = line_stations.map(lambda x: (x[0], (float(x[3]), float(x[4]))))

#Broadcast stations
data_station = stations.collectAsMap()
broadcast_stations = sc.broadcast(data_station)

#temp values (key = (stationid, date, time),  value = (temp,lo + la))
```

```
temp = line_temp.map(lambda x: ( (x[0], x[1], int(x[2][0:2])), (float(x[3]),
broadcast_stations.value.get(x[0])) ))

#filter data till the date
temp = filter_days(date, temp)
temp.cache()

predTemp_sum = []
predTemp_product = []

# Predict temperatures for each specified time
for time in [24, 22, 20, 18, 16, 14, 12, 10, 8, 6, 4]:
    #three gaussian kernel for each data point
    #(key,value) = ((stationid, date, time) , (distanceKernel, daysKernel, hours Kernel, temp))
    gaussianKernels = temp.map(lambda x : (x[0], (gKernel( haversine(a,b, x[1][1][0],
x[1][1][1]), h_distance), gKernel(diff_days(date, x[0][1]), h_date), gKernel(diff_hours(time,
x[0][2]), h_time), x[1][0])))

    #combining all three kernels to get the weighted average for sum and multiply
    kernels = gaussianKernels.map(lambda x: (1, (( x[1][0] + x[1][1] + x[1][2]) * x[1][3],
x[1][0] + x[1][1] + x[1][2], x[1][0] * x[1][1] * x[1][2] * x[1][3], x[1][0] * x[1][1] * x[1][2]
)))
    kernels = kernels.reduceByKey( lambda x1, x2:(x1[0]+x2[0], x1[1]+x2[1], x1[2]+x2[2],
x1[3]+x2[3]))
    kernels = kernels.mapValues(lambda x: (x[0]/x[1], x[2]/x[3]))

    #seperating results of sum and multiplication of kernel methods
    sum_kernel = kernels.collectAsMap().get(1)[0]
    prod_kernel = kernels.collectAsMap().get(1)[1]

    predTemp_sum.append((time, sum_kernel))
    predTemp_product.append((time, prod_kernel))

print('Pred from sum kernel')
print(predTemp_sum)
print('Pred from prod kernel')
print(predTemp_product)
```