732A54: Lab 3
Big Data Analytics

Carles Sans Fuentes

March 28, 2017

Exercise 1

In this section a kernel method must be implemented to predict the hourly temperatures for a date and a place in Sweden using the stations.csv and temperature-readings.csv files. These files contain information about weather stations and temperature measurements for the different stations at different days and times in Sweden. With that, it must be given a temperature forecast for a date and place in Sweden, using temperatures from 4 am to 24 pm in an interval of 2 hours.

To do so, three Gaussian kernel functions have been implemented:

- A kernel to account for the distance from a station to the point of interest. This has been implemented using the distHaversine function from the geosphere package.
- A kernel that accounts for the distance between the day a temperature measurement was made and the day of interest.
- A kernel that accounts for the distance between the hour of the day a temperature measurement was made and the hour of interest.

The "h" provided by the gaussian formula is 50 for km, 15 days, and 18000 seconds (5 hours). The formula is provided below:

$$K(x, x') = e^{-\frac{|x-x'|^2}{2h^2}}$$

This is acceptable due to the fact that I want to try not to overestimate the results for the data, such that by taking these h I am trying that only a month will be taken into account for prediction, just 50 km for checking the nearby stations, and then also just 5 hours up and down on a day to perform the prediction.

Results

A plot prediction for the day 2012-12-04 has been shown in the following figure (figure 1 below). The code has lasted 1:47:27.326531 hours to output the result.

It can be seen that the prediction is quite a straight line throughout the 0 degrees. throughout the day. It could make sense that the temperature could be quite stable throughout the day even though the what inductively should happen is that the highest temperature of a day should be around 11- 16, and that is what happens. For that reason and for the fact that the temperature looks quite high for December though possible, I think that my prediction is quite not really accurate for this case. However, other days should be evaluated to know how good our prediction is.

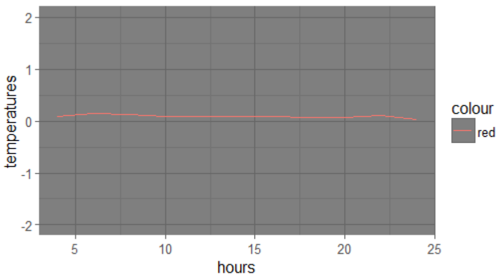

Figure 1: Prediction temperatures plot for the day 2013-11-04

Below the detailed average temperatures are given:

```
[[0.0965644688194552], [0.14551540492916287], [0.13637480199167554], [0.08606147967202618],
    [0.09239058705458468], [0.09179091198975678], [0.08841534779501738], [0.0647447218727851 7],
     [0.06279345442117194], [0.09921507167837842], [0.03027204326630707]]
```

Nevertheless, considering the three kernel variables as independent does not make much sense in our case and it bias our results down. The intuition behind this fact is that by taking time and day separately, all temperatures from that day or that time in Sweden affects equally our predictor. Another way to say this is that by considering date and time separated to the station, dates and times that are far from weather stations will have the same value as those near, and thus it will affect negatively the results. For that, those who are nearby and hence more valuable for our prediction are not used. A better prediction for it could be weighting each variable according to cross-validation: for instance, giving more importance to stations that are nearby and less to time, assuming different Kernel models (e.g. Cauchy or Epanechnikov kernel), or creating dependence between the variables and so by the kernels.

Appendix

Assignment 1

```

###Kernel Gaussian distance
def kDistGaussian(dist, h):
    import math, collections
    if isinstance(dist, collections.Iterable):
        k_dist = [math.exp(float(-abs((x)**2))/(2*h**2)) for x in dist]
    else:
        k_dist = math.exp(float(-abs((dist)**2))/(2*h**2))

    return k_dist

#####km Distance
def haversine(lon1, lat1, lon2, lat2):
    from math import radians, cos, sin, asin, sqrt, exp
    from datetime import datetime
    """
    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

####Date distance

def dist_Date(date1, date2):
    import datetime
    date1 = datetime.datetime.strptime(date1, "%Y-%m-%d")
    date2 = datetime.datetime.strptime(date2, "%Y-%m-%d")
    k_date = date1-date2
    return  k_date.days

####Seconds distance


def dist_Time(Time1, Time2):
    import datetime

    Time1 = datetime.datetime.strptime(Time1, "%H:%M:%S")
    Time2 = datetime.datetime.strptime(Time2, "%H:%M:%S")
    k_time = (Time1 -Time2).total_seconds()

    return k_time
```

```

#importing my functions

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")
sc.addFile("/nfshome/x_carsa/Desktop/data/kernel_functions.py")

import kernel_functions as func

#x[0] = station number x[1] date, x[2] time, x[3] temperature

temp_date_file = sc.textFile("/user/x_carsa/data/temperature-readings.csv")\
            .map(lambda line: line.split(";")) \
            .map(lambda obs: (obs[0], (str(obs[1]), str(obs[2]), float(obs[3]))))



#x[0] = station number, x[3] latitude, x[4] longitude
location_file = sc.textFile("/user/x_carsa/data/stations.csv")\
                .map(lambda line: line.split(";")) \
                .map(lambda obs: (obs[0], (float(obs[3]),float(obs[4])))) \
                .collect()

location_file_broad = sc.broadcast(location_file).value
```

```

location_file_broad = {station: pos for (station, pos) in location_file_broad} # first part (
    month: temp, = structure type, for = old structure type)
new_location_file_broad= location_file_broad
###(station, (date, time, temp, lon, lat))

def mymerge(x):
    merge1= temp_date_file.map(lambda (station, pos): \
                    (station,(pos[0], pos[1], pos[2], \
                     x[station][1], \
                     x[station][0] \
                     )))
    return merge1

Merge_data = mymerge(new_location_file_broad)

def GaussianKernel(date, time, lon, lat, data):
    ###data = list((station, (date, time, temp, lon, lat)),
        #+(...))
    import datetime
    start_time = datetime.datetime.now()
    result = []

    data_filter = data.filter(lambda x: (datetime.datetime.strptime(x[1][0], "%Y-%m-%d") <
        datetime.datetime.strptime(date, "%Y-%m-%d"))) \
                                    .cache()


    for times in time:
        distances = data_filter.map(lambda x: (x[1][2],(func.haversine(lon1 = lon, lat1 = lat,
            lon2 =x[1][3],lat2=x[1][4]), \
                                                func.dist_Date(date1= date, date2=x[1][0]),\
                                                func.dist_Time(Time1= times, Time2=x[1][1]))))


        Gaussian =  distances.map(lambda (temp, (dkm, dDate,dTime)): (temp,\
                                    (func.kDistGaussian(dkm, h = 50),\
                                    func.kDistGaussian(dDate, h = 15),\
                                    func.kDistGaussian(dTime, h = 18000) \
                                    )))

        finalGauss = Gaussian.map(lambda (temp, (k1,k2,k3)): (temp,( k1+k2+k3)))\
                                .map(lambda (temp, (ksum)): (temp, (ksum, ksum*temp)))\
                                .map(lambda (temp, (ksum,krowmult)): ("together",(ksum,
                                    krowmult)))\
                                .reduceByKey(lambda (ksum1,ksum2),(krowmult1,krowmult2): \
                                                (ksum1+ksum2, krowmult1+krowmult2)) \
                                .map(lambda (together,( totalksum, totalkrowmult)): float(
                                    totalkrowmult)/float(totalksum))

        result.append(finalGauss.collect())

    print str(datetime.datetime.now() - start_time)
    print result
    return result

a = 58.4274 # latitude
b = 14.826  #longitude
onedate = "2012-12-04"
mytime = ["04:00:00", "06:00:00", "08:00:00","10:00:00","12:00:00", "14:00:00","16:00:00", "
    18:00:00","20:00:00", "22:00:00","00:00:00"]
myprediction =  GaussianKernel(date= onedate, time= mytime,\
                                lon= b, lat = a, data = Merge_data)
myprediction = sc.parallelize(myprediction).repartition(1)
print myprediction
```