732A54 Big Data Analytics

LAB EXERCISE 3: MACHINE LEARNING

Group G6:

Hoda Fakharzadehjahromy(hodfa840)
Seyda Aqsa Iftekhar(syeif776)

Spark and Python. **Do not use Spark SQL or any other programming languages**.

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:

Y The first to account for the distance from a station to the point of interest.

Y The second to account for the distance between the day a temperature measurement was made and the day of interest.

Y 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.

Results:

```
[('04:00:00', 14.70261442434642), ('06:00:00', 16.813432854608553),
('08:00:00', 18.671774525802373), ('10:00:00', 22.15937530110451),
('12:00:00', 21.256292838613014), ('14:00:00', 22.6788734661957), ('16:00:00',
22.79005560997927), ('18:00:00', 19.67894097490446), ('20:00:00',
16.001156638313443), ('22:00:00', 14.797512402873513), ('00:00:00',
15.819477451732176)]
```

The prediction shows higher value in the afternoon than in the morning. This seems reasonable. However, the estimated value seems to be actually higher since we expect the weather to be colder in august.

Code:

```
from __future__ import division
from math import radians, cos, sin, asin, sqrt, exp
from datetime import datetime
from pyspark import SparkContext
import os
import sys

os.environ['PYSPARK_PYTHON'] = sys.executable
os.environ['PYSPARK_DRIVER_PYTHON'] = sys.executable
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 #km
h_date = 4 # Days
h_time = 2*60*60 # Seconds
a = 58.4274 # Up to you
b = 14.826 # Up to you
date = "2013-08-04" # Up to you
times = ["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"]
temp = [0]*len(times)
res = [0] * len(times)
temp_mul = [0]*len(times)

stations_file = sc.textFile("data/stations.csv")
temps_file = sc.textFile("data/temperature-readings.csv")
#temps_file = temps_file.sample(False,0.1)

stationLines = stations_file.map(lambda line: line.split(";"))
stations = stationLines.map(lambda x: (int(x[0]), float(x[3]),
        float(x[4])))

tempLines = temps_file.map(lambda line: line.split(";"))
temps = tempLines.map(lambda x: (int(x[0]),x[1],x[2],float(x[3])))

def gaussian_Kernel_dist(data, coords, h):
```

```
        u = data.map(lambda x:
(x[0],haversine(x[2],x[1],coords[0],coords[1])/h))
        k = u.map(lambda x: (x[0],exp(-(x[1]**2))))
        #print k.collect()
        return k

def gaussian_Kernel_date(x, date, h):
        diff_date       = (datetime(int(x[0:4]),int(x[5:7]),int(x[8:10]))
                -
datetime(int(date[0:4]),int(date[5:7]),int(date[8:10]))).days / h
        k = exp(-(diff_date**2))
        #print(k.collect())
        return k

def gaussian_Kernel_time(x, time, h):
        diff_time = (datetime(2000,1,1,int(x[0:2]),int(x[3:5]),int(x[6:8]))
                -
datetime(2000,1,1,int(time[0:2]),int(time[3:5]),int(time[6:8]))).seconds / h
        k = exp(-(diff_time**2))
        #print k.collect()
        return k


def predict():
        k_dist = gaussian_Kernel_dist(stations, [b, a], h_distance)
        k_dist_broadcast = k_dist.collectAsMap()
        stations_dist = sc.broadcast(k_dist_broadcast)

        #Filter on date
        filtered_dates = temps.filter(lambda x:
                (datetime(int(x[1][0:4]),int(x[1][5:7]),int(x[1][8:10]))
                <= datetime(int(date[0:4]),int(date[5:7]),int(date[8:10]))))
        filtered_dates.cache()

        for time in times:
                #Filter on time
                filtered_times = filtered_dates.filter(lambda x:

        ((datetime(int(x[1][0:4]),int(x[1][5:7]),int(x[1][8:10]))
                       ==
datetime(int(date[0:4]),int(date[5:7]),int(date[8:10]))) and

        (datetime(2000,1,1,int(x[2][0:2]),int(x[2][3:5]),int(x[2][6:8]))
                       <=
datetime(2000,1,1,int(time[0:2]),int(time[3:5]),int(time[6:8]))))


                kernel = filtered_times.map(lambda x:
(stations_dist.value[x[0]],

           gaussian_Kernel_date(x[1], date, h_date),

           gaussian_Kernel_time(x[2], time, h_time), x[3]))

                k_sum = kernel.map(lambda x: (x[0] * x[1] * x[2],x[3]))
                k_sum = k_sum.map(lambda x: (x[0]*x[1],x[0]))
```

```
                k_sum = k_sum.reduce(lambda x,y: (x[0]+y[0],x[1]+y[1]))
                res[times.index(time)] = (time, k_sum[0] / k_sum[1])
                return (res)


print(predict())
```