BDA3 – Machine Learning

Alexander Bois (alebo256) & Daniel Bissessar (danbi675)

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")

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

def timediff(time1,time2):
        t1 = int(time1[0:2])
        t2 = int(time2[0:2])
        diff = abs(t1-t2)
        if (diff > 12):
                return 24-diff
        return diff

def datediff(date1,date2):
        d1 = datetime(int(date1[0:4]),int(date1[5:7]),int(date1[8:10]))
        d2 = datetime(int(date2[0:4]),int(date2[5:7]),int(date2[8:10]))
        diff = d1-d2
        diff = diff.days
        diff = diff%365
        if (diff>182):
                return 365-diff
        return diff

h_distance =50000
h_date = 15
h_time = 4
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

def kernel(diff, h):
        return (exp(-(diff/h)**2))

st_lines = stations.map(lambda line: line.split(";"))
te_lines = temps.map(lambda line: line.split(";"))

stat = st_lines.map(lambda x: (x[0],(float(x[3]),float(x[4])))).collectAsMap()
station_list = sc.broadcast(stat)

temp = te_lines.map(lambda x:
((x[0],x[1],x[2]),(float(x[3]),station_list.value.get(x[0]))))
frent = datetime(int(date[0:4]),int(date[5:7]),int(date[8:10]))
temp_filter = temp.filter(lambda x:
datetime(int(x[0][1][0:4]),int(x[0][1][5:7]),int(x[0][1][8:10]))<frent)
temp_list = temp_filter.cache()

pred_sum = []
pred_mult = []
```

```
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"]:

        kernels = temp_list.map(lambda x:
(kernel(datediff(date,x[0][1]),h_date),kernel(timediff(time,x[0][2]),h_time),kernel
(haversine(a,b,x[1][1][0],x[1][1][1]),h_distance),x[1][0]))
        kern_sum = kernels.map(lambda x:
(1,(x[3]*(x[0]+x[1]+x[2]),x[0]+x[1]+x[2],x[3]*x[0]*x[1]*x[2],x[0]*x[1]*x[2])))
        kern_summ = kern_sum.reduceByKey(lambda a,b: (a[0]+b[0], a[1]+b[1],
a[2]+b[2], a[3]+b[3]))
        predSum=kern_summ.mapValues(lambda x: (x[0]/x[1],x[2]/x[3])).collectAsMap()
        pred_sum.append((time, predSum.get(1)[0]))
        pred_mult.append((time, predSum.get(1)[1]))

print(pred_sum)
print(pred_mult)
# Your code here
```

Result:

```
Sum of kernels
('24:00:00', 4.81538920860463)
('22:00:00', 4.91585909525204)
('20:00:00', 5.04322717365766)
('18:00:00', 5.18470335385016)
('16:00:00', 5.32406618099935)
('14:00:00', 5.41949307220866)
('12:00:00', 5.41092191144642)
('10:00:00', 5.27191273224620)
('08:00:00', 5.05199285829180)
('06:00:00', 4.85751427434033)
('04:00:00', 4.75990655461475)

Multiplied kernels
('24:00:00', 12.73176940568272)
('22:00:00', 13.75826210809814)
('20:00:00', 15.11449761264393)
('18:00:00', 16.23422040859560)
('16:00:00', 17.02898982894894)
('14:00:00', 17.46085884881963)
('12:00:00', 17.36548410700745)
('10:00:00', 16.65336362272281)
('08:00:00', 15.48364638306218)
('06:00:00', 14.27195053106827)
('04:00:00', 13.24647007565829)
```

Questions:

1. We choose the h values for the different kernels after running the program on a smaller data set in Rstudio and plotting the values for the contribution of different data points for each kernel. With the help of these plots we reasoned about how many points should have large vs small contribution to each kernel. We found that the h-values in our code was suitable, result wise it only worked well for the multiplied kernels.
2. The summed kernels and the multiplied kernels differ since the weighting becomes very different for “extreme” values for different kernels. In the summed kernels, each kernel is somewhat independent in their contribution, while in the multiplied case they are not. For example, if the distance kernel is very small (implying far away), but the day and time kernel weights are close to one, the resulting kernel for that point in the summed kernels will be bigger, since the time and day kernels are not affected by the distance kernel. This is not the case in the multiplied kernels where the total kernel will be still very small in any kernel is very small.