732A54 - Big Data Analytics

BDA3 - Machine Learning with Spark - Exercises

Ashraf Sarhan (ashsa762)

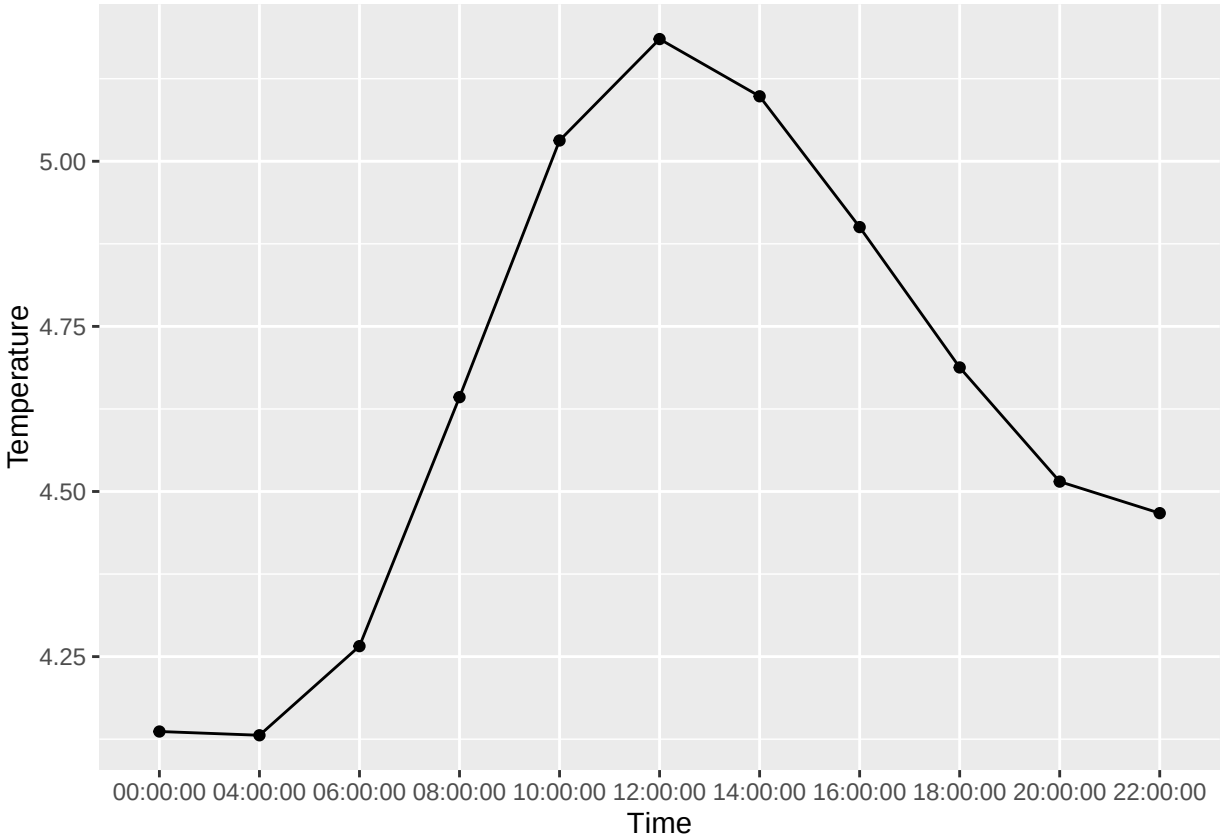
Introduction:

In this lab we were supposed to implement a kernel to predict temperatures. The kernel is a linear combination of three Gaussian kernels (date, time of day and geographical distance)

The estimated temperatures below were predicted in Linköping 2013-01-25. As we can notice that the lower temperatures in the morning and evening with a higher temperatures in the afternoon. However, the values of the temperatures doesn't reflects the common temperatures during the winter time which normlly below zero, This indicates that the kernel does not consider seasonal time.

Plot:

```
temps <- read.csv('temp.csv', header = TRUE, sep = ',')
library(ggplot2)
ggplot(temps, aes(time, temp, group = 1)) +
         geom_point() +
         geom_line() +
         labs(x = "Time", y = "Temperature")
```



kernels:

Date kernel:

For the date kernel, we set the width to 7 days as we assumed that the temperature of a particular day is close to what it has been the 7 previous days.

Time kernel:

For the time kernel, We set the width to be 2 hours since the last few hours should have a a closer temperatures as the current time.

Time kernel:

For the distance kernel we set the width of 100 to transform the kilometers to 10 Swedish miles which we assumed that is a resanable measure based on the size of Sweden as the location affects the temperature quite much where the further to the north the colder.

Output:

```
temps <- read.csv('temp.csv', header = TRUE, sep = ',')
head(temps, 10)
```

```
##        time     temp
## 1  04:00:00 4.130918
## 2  06:00:00 4.265844
## 3  08:00:00 4.642866
## 4  10:00:00 5.031491
## 5  12:00:00 5.185055
## 6  14:00:00 5.098405
## 7  16:00:00 4.900331
## 8  18:00:00 4.687833
## 9  20:00:00 4.514975
## 10 22:00:00 4.467204
```

Conclusion:

We though that main problem with the currant kernel is that the three kernels are independent of each other. The date kernel is the only kernel that can detect seasonal trends but it cannot contribute much to the prediction as it's independet. Had the kernels been multiplied together the predictions would probably have had been much better.

Code:

kernel_function.py

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Kernel Function

@author: ashraf
"""

from pyspark import SparkContext

sc = SparkContext(appName = "kernel")

sc.addFile("/nfshome/x_asmoh/py_scripts/help_functions.py")

import help_functions as hfunc

# Station, lat, long
stations = sc.textFile("data/stations.csv") \
                .map(lambda line: line.split(";")) \
                .map(lambda obs: (obs[0], (float(obs[3]),float(obs[4])))) \
                .collect()

stations_dict = {}
for s in stations:
    stations_dict[s[0]] = s[1]

#Broadcast stations_dict
stations_bc = sc.broadcast(stations_dict)

# (station, (date, time, temp))
temperatures = sc.textFile("data/temperature-readings.csv") \
                    .sample(False, .0001, 12345) \
                    .map(lambda line: line.split(";")) \
                    .map(lambda l: \
                        (l[0], (str(l[1]), str(l[2]), float(l[3]))))

def mergeVal(x):
    sVals = list(stations_bc.value[x[0]])
    vals = list(x[1])
    vals.extend(sVals)
    return (x[0],tuple(vals))

def kernelFunc(pred, data, dist):
    import datetime
    result = list()
    for p in pred:
        temp = data \
            .filter(lambda x: \
                datetime.datetime.strptime(x[1][0], '%Y-%m-%d') < \
```

```
                datetime.datetime.strptime(p[1], '%Y-%m-%d')) \
            .map(lambda x: \
                (x[1][2], ( \
                        hfunc.deltaHours(p[0],x[1][1]), \
                        hfunc.deltaDays(p[1], x[1][0]), \
                        hfunc.haversine(lon1 = p[2], \
                                        lat1 = p[3], \
                                        lon2 = x[1][4], \
                                        lat2 = x[1][3])))) \
            .map(lambda (temp, (distTime, distDays, distKM)): \
                (temp,(hfunc.gaussian(distTime, h = dist[0]), \
                        hfunc.gaussian(distDays, h = dist[1]), \
                        hfunc.gaussian(distKM, h = dist[2])))) \
            .map(lambda (temp, (ker1, ker2, ker3)): \
                (temp,ker1 + ker2 + ker3)) \
            .map(lambda (temp, kerSum): \
                (temp, (kerSum, temp*kerSum))) \
            .map(lambda (temp, (kerSum, tkSum)): \
                (None, (kerSum, tkSum))) \
            .reduceByKey(lambda (kerSum1, tkSum1), (kerSum2, tkSum2): \
                (kerSum1 + kerSum2, tkSum1 + tkSum2)) \
            .map(lambda (key,(sumKerSum, sumTkSum)): \
                (float(sumTkSum)/float(sumKerSum)))
        result.append((p[0], temp.collect()))
    return result

# Test the kernelFunc
# (station, (date, time, temp, lat, long))
train = temperatures.map(lambda l: mergeVal(l))
pred = (('04:00:00', '2013-01-25',float(15.62), float(58.41)),
        ('06:00:00', '2013-01-25',float(15.62), float(58.41)),
        ('08:00:00', '2013-01-25',float(15.62), float(58.41)),
        ('10:00:00', '2013-01-25',float(15.62), float(58.41)),
        ('12:00:00', '2013-01-25',float(15.62), float(58.41)),
        ('14:00:00', '2013-01-25',float(15.62), float(58.41)),
        ('16:00:00', '2013-01-25',float(15.62), float(58.41)),
        ('18:00:00', '2013-01-25',float(15.62), float(58.41)),
        ('20:00:00', '2013-01-25',float(15.62), float(58.41)),
        ('22:00:00', '2013-01-25',float(15.62), float(58.41)),
        ('00:00:00', '2013-01-25',float(15.62), float(58.41)))
dist = (2, 7, 100)
rsltPred = kernelFunc(pred, train, dist)
rsltPred_rdd = sc.parallelize(rsltPred).repartition(1)
rsltPred_rdd.take(10)
```

help_functions.py

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Help Function

@author: ashraf
"""
def gaussian(dist, h):
    import math, collections
    if isinstance(dist, collections.Iterable):
        res = []
        for x in dist:
            res.append(math.exp(float(-(x**2))/float((2*(h**2)))))
    else:
        res = math.exp(float(-(dist**2))/float((2*(h**2))))
    return res

def haversine(lon1, lat1, lon2,lat2, radians = 6371):
    import math
    # Convert decimal degrees to radians
    lon1, lat1, lon2,lat2 = map(math.radians, [lon1, lat1, lon2,lat2])
    dlon = lon2 - lon1
    dlat = lat2 - lat2
    a = math.sin(dlat/2)**2 + math.cos(lat1) * \
        math.cos(lat2) * math.sin(dlon/2)**2
    c = 2 * math.asin(math.sqrt(a))
    return c * radians

def timeCorr(time):
    import math
    result = []
    if hasattr(time, '__iter__'):
        for x in time:
            if x <= -12:
                result.append(24 + x)
            else:
                result.append(math.fabs(x))
    else:
        if time <= -12:
            result = 24 + time
        else:
            result = math.fabs(time)
    return result

def deltaHours(time1, time2):
    import datetime
    hDelta = datetime.datetime.strptime(time1, '%H:%M:%S')-
      datetime.datetime.strptime(time2, '%H:%M:%S')
    tDiff = hDelta.total_seconds()/3600
    tCorr = timeCorr(tDiff)
    return tCorr
```

```
def deltaDays(day1, day2):
    import datetime
    dDelta = datetime.datetime.strptime(day1, '%Y-%m-%d')-
      datetime.datetime.strptime(day2, '%Y-%m-%d')
    return dDelta.days
```