BDA 3 – ML with SPARK

732A54 – BIG DATA ANALYTICS

Thijs Quast (thiqu264)
Alexander Karlsson (aleka769)

1. Show that your choice for the kernels' width is sensible, i.e. it gives more weight to closer points. Discuss why your definition of closeness is reasonable.

As kernel widths we take the following values:

Distance: 150
Date: 30
Time: 3

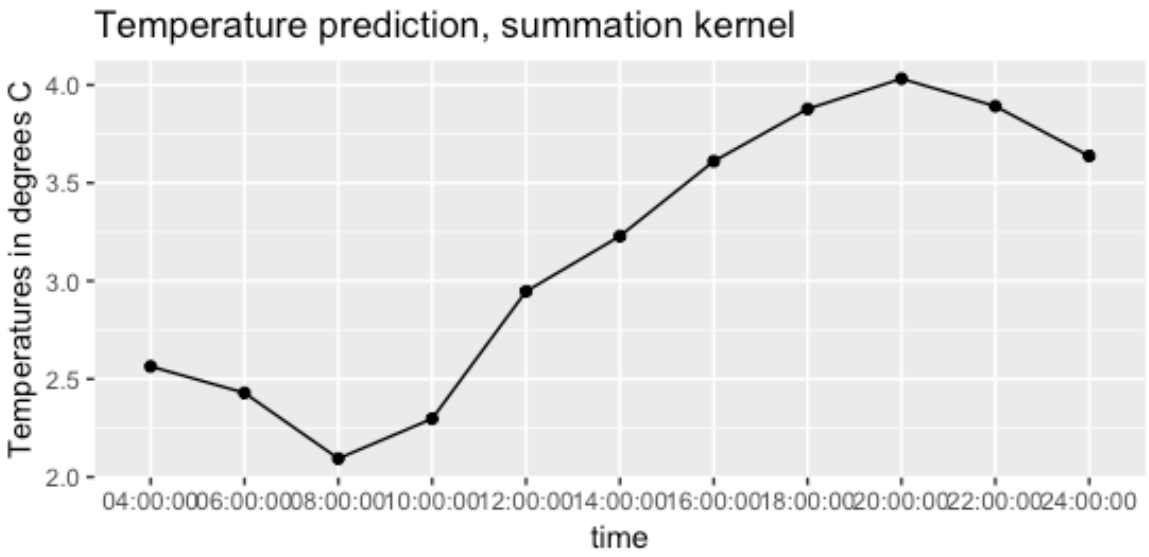
We think these values are reasonable, as we believe the weather within 150 distance units is still to some extent related. Temperatures are somewhat related within a month, and a timeframe of 3 hours is reasonable, as for instance, the temperature at 9.00 in the morning is to some slight extent still related to the temperature at 12.00

Predicted location:
(57.7236, 12.9641)

Predicted day:
2010-01-21

Predicted temperatures:

Time	**Temperature**
04:00:00	2.563905592304216
06:00:00	2.4289710840486594
08:00:00	2.0938878495350055
10:00:00	2.2972411443145675
12:00:00	2.946794691923887
14:00:00	3.228671342110132
16:00:00	3.6106886768120763
18:00:00	3.877643890122466,
20:00:00	4.031673237654158
22:00:00	3.8917132069077445
24:00:00	3.637445913192351



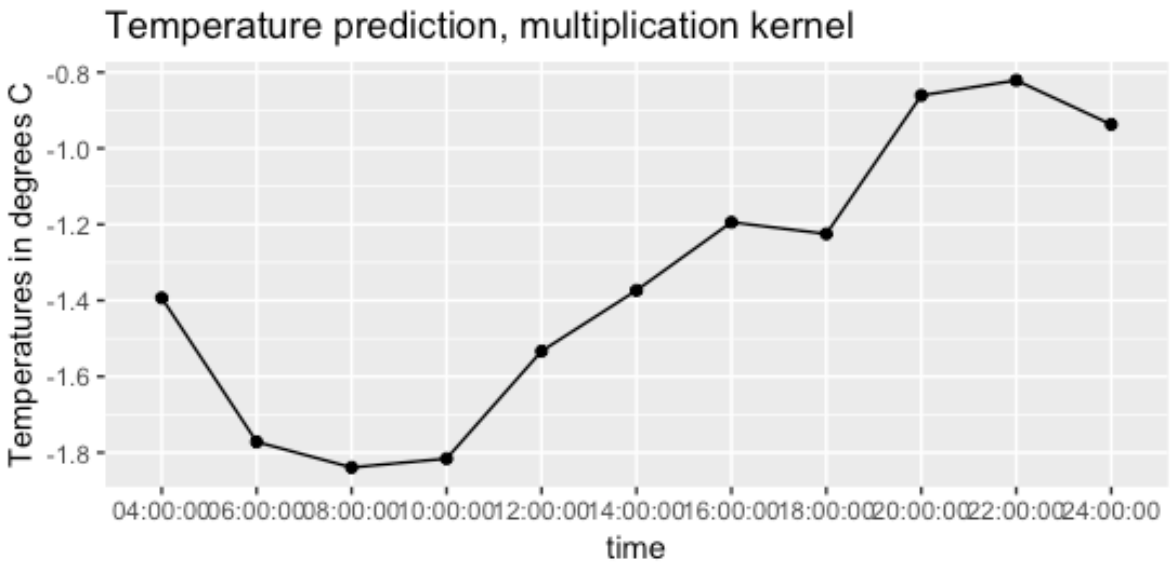
Temperature predictions for this day seem to follow a bit of a cycle which is a bit too much shifted towards the end of the day. Normally one would expect the highest temperatures to be around 14:00:00 or 15:00:00 in the day. We believe the somewhat distorted predictions might come from the chosen kernel widths, which might be a bit too inaccurate. Relevant code for question 1 can be found in the appendix.

2. It is quite likely that the predicted temperatures do not differ much from one another. Do you think that the reason may be that the three Gaussian kernels are independent one of another?

If so, propose an improved kernel, e.g. propose an alternative way of combining the three Gaussian kernels describedabove.

As an alternative we propose a multiplicatie kernel method. Results are the following:

Time	Temperature
04:00:00	−1.393287490069491
06:00:00	−1.7719255977161696
08:00:00	−1.8391576285244362
10:00:00	−1.8159203936222441
12:00:00	−1.5333861509157185
14:00:00	−1.3735338269981499
16:00:00	−1.194267592639943
18:00:00	−1.2249885679058476
20:00:00	−0.8608471290658204
22:00:00	−0.8211958284076682
24:00:00	−0.9371973735325981



When using the multiplicative kernel methods, the predicted temperatures are lower than with the summation kernel. The pattern in the predictions however seems rather similar, towards the end of the day we expect the warmest temperatures. Which is not so realistic, however we believe this could be due to the chosen kernel widths. Relevant code for question 2 can be found in the appendix.

Appendix

Question 1

```
from __future__ import division
from math import radians, cos, sin, asin, sqrt, exp
from datetime import datetime
from operator import truediv

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 kernel(diff, kernel_width):
    weight = exp(-(diff/kernel_width)**2)
    return weight

h_distance = 150 # Up to you
h_date = 30 # Up to you
h_time = 3 # Up to you
a = 57.7236 # Up to you
b = 12.9641 # Up to you
date = "2010-01-21"# Up to you
time = "12:00:00"

import pyspark
sc = pyspark.SparkContext(appName="Q1_")

temp_path = "/user/x_thiqu/data/temperature-readings.csv"
temperature_file = sc.textFile(temp_path)

stat_path = "/user/x_thiqu/data/stations.csv"
stations_file = sc.textFile(stat_path)

def as_date(date):
    date_format = "%Y-%m-%d"
    a = datetime.strptime(date, date_format)
    return a

lines = temperature_file.map(lambda line: line.split(";"))
lines2 = stations_file.map(lambda line: line.split(";"))

#lines = lines.sample(False, 0.01)
lines = lines.filter(lambda x: as_date(x[1]) < as_date(date))

# datediff function (In days)
def datediff(date1,date2,h_date):
    date_format = "%Y-%m-%d"
```

```
    a = datetime.strptime(date1, date_format)
    b = datetime.strptime(date2, date_format)
    delta = b - a
    dd = delta.days % 365
    if dd > (365/2):
        c = 365 - dd
    else:
        c = dd
    return kernel(c, h_date)

# time difference function
def timediff(time1,time2,h_time):
    date_format = "%H:%M:%S"
    a = datetime.strptime(time1, date_format)
    b = datetime.strptime(time2, date_format)
    delta = b - a
    dt = delta.seconds/3600

    if dt > 12:
        c = 24 - (dt/3600)
    else:
        c = dt/3600

    return kernel(c, h_time)

year_temperature = lines.map(lambda x: ((int(x[0]),
                                x[1],
                                x[2],
                                (float(x[3])
                                ))))

year_temperature = year_temperature.map(lambda x: (x[0],  #stationNr
                                    (datediff(x[1], date, h_date),
                                    timediff(x[2], "04:00:00", h_time),
                                    timediff(x[2], "06:00:00", h_time),
                                    timediff(x[2], "08:00:00", h_time),
                                    timediff(x[2], "10:00:00", h_time),
                                    timediff(x[2], "12:00:00", h_time),
                                    timediff(x[2], "14:00:00", h_time),
                                    timediff(x[2], "16:00:00", h_time),
                                    timediff(x[2], "18:00:00", h_time),
                                    timediff(x[2], "20:00:00", h_time),
                                    timediff(x[2], "22:00:00", h_time),
                                    timediff(x[2], "00:00:00", h_time),
                                    x[3])))


stations = lines2.map(lambda x: ((int(x[0]),
                            float(x[3]),
                            float(x[4]),
                           )))

stations = stations.map(lambda x: (x[0], x[1], x[2], haversine(x[1], x[2], a, b)))
stations= stations.map(lambda x: (x[0], kernel(x[3], h_distance)))
stations = stations.collectAsMap()
bc = sc.broadcast(stations)
rdd = year_temperature.map(lambda x: (x[0], x[1], bc.value.get(x[0])))
```

```
def sum_kernels(a,b,c):
   d = a+b+c
   return d

rdd = rdd.map(lambda x: (x[0],
                  sum_kernels(x[1][0], x[1][1], x[2]),
                  sum_kernels(x[1][0], x[1][2], x[2]),
                  sum_kernels(x[1][0], x[1][3], x[2]),
                  sum_kernels(x[1][0], x[1][4], x[2]),
                  sum_kernels(x[1][0], x[1][5], x[2]),
                  sum_kernels(x[1][0], x[1][6], x[2]),
                  sum_kernels(x[1][0], x[1][7], x[2]),
                  sum_kernels(x[1][0], x[1][8], x[2]),
                  sum_kernels(x[1][0], x[1][9], x[2]),
                  sum_kernels(x[1][0], x[1][10], x[2]),
                  sum_kernels(x[1][0], x[1][11], x[2]),
                  x[1][12]))

#rdd = rdd.sortBy(ascending=False, keyfunc = lambda k: k[0], numPartitions=1)

rdd = rdd.map(lambda x: ( [x[1]*x[-1],
                   x[2]*x[-1],
                   x[3]*x[-1],
                   x[4]*x[-1],
                   x[5]*x[-1],
                   x[6]*x[-1],
                   x[7]*x[-1],
                   x[8]*x[-1],
                   x[9]*x[-1],
                   x[10]*x[-1],
                   x[11]*x[-1]],

                   [x[1],
                   x[2],
                   x[3],
                   x[4],
                   x[5],
                   x[6],
                   x[7],
                   x[8],
                   x[9],
                   x[10],
                   x[11]]
                  ))

rdd = rdd.reduce(lambda x, y: ([x[0][0] + y[0][0],
                       x[0][1] + y[0][1],
                       x[0][2] + y[0][2],
                       x[0][3] + y[0][3],
                       x[0][4] + y[0][4],
                       x[0][5] + y[0][5],
                       x[0][6] + y[0][6],
                       x[0][7] + y[0][7],
                       x[0][8] + y[0][8],
                       x[0][9] + y[0][9],
                       x[0][10] + y[0][10]],
```

```
                    [x[1][0]+y[1][0],
                    x[1][1] + y[1][1],
                    x[1][2] + y[1][2],
                    x[1][3] + y[1][3],
                    x[1][4] + y[1][4],
                    x[1][5] + y[1][5],
                    x[1][6] + y[1][6],
                    x[1][7] + y[1][7],
                    x[1][8] + y[1][8],
                    x[1][9] + y[1][9],
                    x[1][10] + y[1][10]]))

rdd = map(truediv, rdd[0], rdd[1])
rdd = list(rdd)
print(rdd)
```

Question 2

```
from __future__ import division
from math import radians, cos, sin, asin, sqrt, exp
from datetime import datetime
from operator import truediv

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 kernel(diff, kernel_width):
    weight = exp(-(diff/kernel_width)**2)
    return weight

h_distance = 150 # Up to you
h_date = 30 # Up to you
h_time = 3 # Up to you
a = 57.7236 # Up to you
b = 12.9641 # Up to you
date = "2010-01-21"# Up to you
time = "12:00:00"

import pyspark
sc = pyspark.SparkContext(appName="Q1_")

temp_path = "/user/x_thiqu/data/temperature-readings.csv"
temperature_file = sc.textFile(temp_path)

stat_path = "/user/x_thiqu/data/stations.csv"
stations_file = sc.textFile(stat_path)

def as_date(date):
    date_format = "%Y-%m-%d"
    a = datetime.strptime(date, date_format)
    return a

lines = temperature_file.map(lambda line: line.split(";"))
lines2 = stations_file.map(lambda line: line.split(";"))

#lines = lines.sample(False, 0.01)
lines = lines.filter(lambda x: as_date(x[1]) < as_date(date))

# datediff function (In days)
def datediff(date1,date2,h_date):
    date_format = "%Y-%m-%d"
    a = datetime.strptime(date1, date_format)
```

```
    b = datetime.strptime(date2, date_format)
    delta = b - a
    dd = delta.days % 365
    if dd > (365/2):
        c = 365 - dd
    else:
        c = dd
    return kernel(c, h_date)

# time difference function
def timediff(time1,time2,h_time):
    date_format = "%H:%M:%S"
    a = datetime.strptime(time1, date_format)
    b = datetime.strptime(time2, date_format)
    delta = b - a
    dt = delta.seconds/3600

    if dt > 12:
        c = 24 - (dt/3600)
    else:
        c = dt/3600

    return kernel(c, h_time)

year_temperature = lines.map(lambda x: ((int(x[0]),
                                x[1],
                                x[2],
                                (float(x[3])
                                ))))

year_temperature = year_temperature.map(lambda x: (x[0],  #stationNr
                                    (datediff(x[1], date, h_date),
                                    timediff(x[2], "04:00:00", h_time),
                                    timediff(x[2], "06:00:00", h_time),
                                    timediff(x[2], "08:00:00", h_time),
                                    timediff(x[2], "10:00:00", h_time),
                                    timediff(x[2], "12:00:00", h_time),
                                    timediff(x[2], "14:00:00", h_time),
                                    timediff(x[2], "16:00:00", h_time),
                                    timediff(x[2], "18:00:00", h_time),
                                    timediff(x[2], "20:00:00", h_time),
                                    timediff(x[2], "22:00:00", h_time),
                                    timediff(x[2], "00:00:00", h_time),
                                    x[3])))


stations = lines2.map(lambda x: ((int(x[0]),
                            float(x[3]),
                            float(x[4]),
                            )))

stations = stations.map(lambda x: (x[0], x[1], x[2], haversine(x[1], x[2], a, b)))
stations= stations.map(lambda x: (x[0], kernel(x[3], h_distance)))
stations = stations.collectAsMap()
bc = sc.broadcast(stations)
rdd = year_temperature.map(lambda x: (x[0], x[1], bc.value.get(x[0])))
```

```
def sum_kernels(a,b,c):
   d = a+b+c
   return d

def multiply_kernels(a,b,c):
            d = a*b*c
            return d

rdd = rdd.map(lambda x: (x[0],
                    multiply_kernels(x[1][0], x[1][1], x[2]),
                    multiply_kernels(x[1][0], x[1][2], x[2]),
                    multiply_kernels(x[1][0], x[1][3], x[2]),
                    multiply_kernels(x[1][0], x[1][4], x[2]),
                    multiply_kernels(x[1][0], x[1][5], x[2]),
                    multiply_kernels(x[1][0], x[1][6], x[2]),
                    multiply_kernels(x[1][0], x[1][7], x[2]),
                    multiply_kernels(x[1][0], x[1][8], x[2]),
                    multiply_kernels(x[1][0], x[1][9], x[2]),
                    multiply_kernels(x[1][0], x[1][10], x[2]),
                    multiply_kernels(x[1][0], x[1][11], x[2]),
                    x[1][12]))

#rdd = rdd.sortBy(ascending=False, keyfunc = lambda k: k[0], numPartitions=1)

rdd = rdd.map(lambda x: ( [x[1]*x[-1],
                     x[2]*x[-1],
                     x[3]*x[-1],
                     x[4]*x[-1],
                     x[5]*x[-1],
                     x[6]*x[-1],
                     x[7]*x[-1],
                     x[8]*x[-1],
                     x[9]*x[-1],
                     x[10]*x[-1],
                     x[11]*x[-1]],

                     [x[1],
                     x[2],
                     x[3],
                     x[4],
                     x[5],
                     x[6],
                     x[7],
                     x[8],
                     x[9],
                     x[10],
                     x[11]]
                    ))

rdd = rdd.reduce(lambda x, y: ([x[0][0] + y[0][0],
                         x[0][1] + y[0][1],
                         x[0][2] + y[0][2],
                         x[0][3] + y[0][3],
                         x[0][4] + y[0][4],
                         x[0][5] + y[0][5],
                         x[0][6] + y[0][6],
                         x[0][7] + y[0][7],
```

```
                    x[0][8] + y[0][8],
                    x[0][9] + y[0][9],
                    x[0][10] + y[0][10]],

                    [x[1][0]+y[1][0],
                    x[1][1] + y[1][1],
                    x[1][2] + y[1][2],
                    x[1][3] + y[1][3],
                    x[1][4] + y[1][4],
                    x[1][5] + y[1][5],
                    x[1][6] + y[1][6],
                    x[1][7] + y[1][7],
                    x[1][8] + y[1][8],
                    x[1][9] + y[1][9],
                    x[1][10] + y[1][10]]))

rdd = map(truediv, rdd[0], rdd[1])
rdd = list(rdd)
print(rdd)
```