# BDAML

Mohammed Ali{mohal954} and Adesijibomi Aderinto{adead268}

5/26/2022

## Initial Kernel Width

kernel widths we take the following values:

Initial kernel values of 150, 30 and 3 was chosen as kernel values for distance,date and hours respectively, but the predicted values somewhat did not look realistic, for example temperatures values for early hours where higher than noon times of the day.

```
================================================================================
(['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'], [5.421764324715561, 4.974977849872319, 5.764412575828927,
 6.382163805097108, 6.738896194526725, 6.900783590920305, 6.777654954618235, 6.081489705907598, 6.100421511487955, 5.897572676966061, 5.497683627457746])
(serviceOption=None,
 services=List(),
 started=false)
```

Figure 1: Kernel Sum

```
================================================================================
(['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'], [7.03247655243638, 7.071701415735579, 7.077037002858471,
7.064586206365586, 7.028053605383858 5, 7.000663096710501, 6.967709088373254, 6.948010297101549, 6.923319915365797, 6.9339254428493655, 6.955834452906066])
(serviceOption=None,
 services=List(),
 started=false)
```

Figure 2: Kernel Product

## Chosen Kernel Width

**Distance: 100.**

**Date: 10days**

**Time: 1hr**

**Distance: (57.7236,12.9641).**

**date = 2013-11-02**

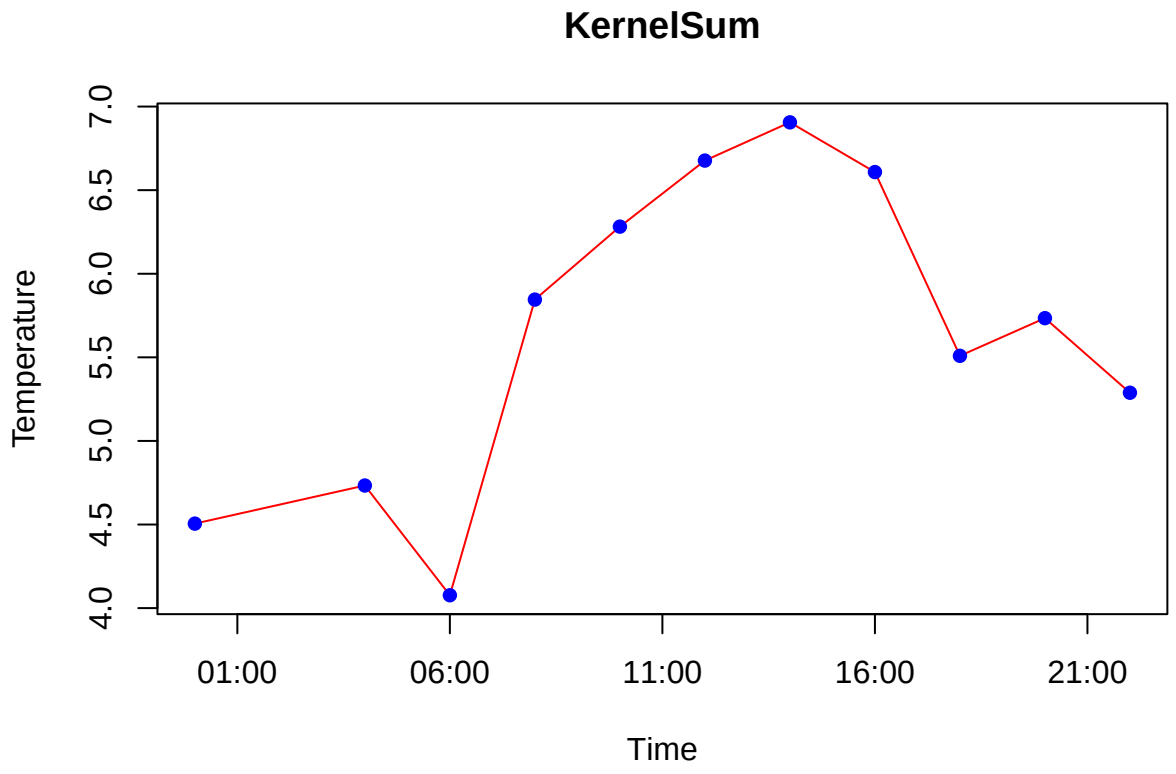
```
================================================================================
(['04:00:00', '06:00:00', '08:00:00', '10:00:00', '12:00:00', '14:00:00', '16:00:00', '18:00:00', '20:00:00', '22:00:00', '00:00:00'], [4.733285424094405, 4.076995491109113, 5.845333615138174,
 6.28180101971014, 6.67694761038637, 6.905568469560453, 6.608083518542613, 5.508976105500491, 5.734415821025383, 5.28823548226495, 4.50532804549835])
(serviceOption=None,
 services=List(),
 started=false)
```

Figure 3: Kernel Sum

```
data_sum <- data.frame(times =c("00:00:00","04:00:00","06:00:00","08:00:00",
    predictions = c(4.50532804549835,4.733285424094405, 4.076995491109113,
                    5.845333615138174, 6.28180101971014, 6.67694761038637,
                    6.905568469560453, 6.608083518542613, 5.508976105500491,
                    5.734415821025383, 5.28823548226495))
data_sum
```

```
##       times predictions
## 1  00:00:00    4.505328
## 2  04:00:00    4.733285
## 3  06:00:00    4.076995
## 4  08:00:00    5.845334
## 5  10:00:00    6.281801
## 6  12:00:00    6.676948
## 7  14:00:00    6.905568
## 8  16:00:00    6.608084
## 9  18:00:00    5.508976
## 10 20:00:00    5.734416
## 11 22:00:00    5.288235
```

```
plot(as.POSIXct(data_sum$times, format = "%H:%M:%OS"), data_sum$predictions,
     type = "l",xlab = "Time", ylab = "Temperature",ylim = c(min(data_sum[,2]),
                                max(data_sum[,2])),col = "red",main = "KernelSum")
points(as.POSIXct(data_sum$times, format = "%H:%M:%OS"),
       data_sum$predictions,pch = 16, col = "Blue")
```



**comments** Kernel sum shows a bit realistic expectations, although not as good as expected, this may be due to the chosen kernel width

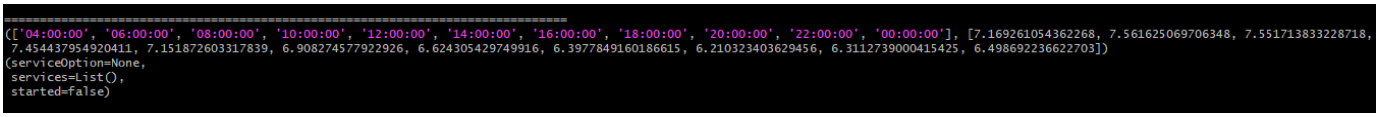
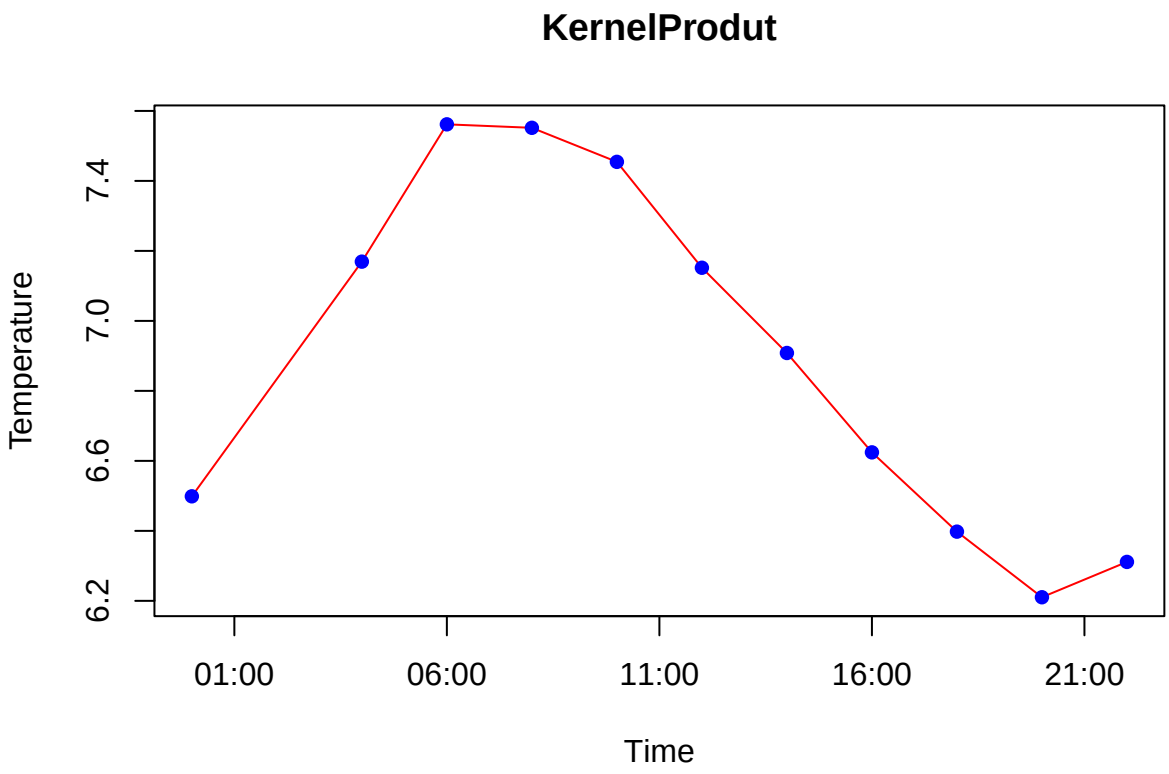



Figure 4: Kernel Product

```
data_prod <- data.frame(times =c("00:00:00","04:00:00","06:00:00","08:00:00",
        predictions = c(6.498692236622703,7.169261054362268, 7.561625069706348,
                        7.551713833228718, 7.454437954920411, 7.151872603317839,
                        6.908274577922926, 6.624305429749916, 6.3977849160186615,
                        6.210323403629456, 6.3112739000415425))
data_prod
```

```
##       times predictions
## 1  00:00:00    6.498692
## 2  04:00:00    7.169261
## 3  06:00:00    7.561625
## 4  08:00:00    7.551714
## 5  10:00:00    7.454438
## 6  12:00:00    7.151873
## 7  14:00:00    6.908275
## 8  16:00:00    6.624305
## 9  18:00:00    6.397785
## 10 20:00:00    6.210323
## 11 22:00:00    6.311274
```

```
plot(as.POSIXct(data_prod$times, format = "%H:%M:%OS"), data_prod$predictions,
     type = "l",xlab = "Time", ylab = "Temperature",ylim = c(min(data_prod[,2]),
                            max(data_prod[,2])),col ="red",main = "KernelProdut")
points(as.POSIXct(data_prod$times, format = "%H:%M:%OS"), data_prod$predictions,
       pch = 16, col = "Blue")
```




**comments** Kernel product shows a better realistic expectations, temperatures at noon are higher, also the kernel looks better that the kernel sum

# Appendix

kernel sum

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

h_distance = 100 # Up to you
h_date = 10 # Up to you
h_time = 1 # Up to you
a = 57.7236 # Up to you
b = 12.9641 # 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"]

date = "2013-11-02"# Up to you

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

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

def diff_date(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
    return dd

def diff_time(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
    return dt

temperature_file = sc.textFile("BDA/input/temperature-readings.csv")
stations_file = sc.textFile("BDA/input/stations.csv")

temps = temperature_file.map(lambda line: line.split(";"))
station = stations_file.map(lambda line: line.split(";"))

lines = temps.filter(lambda x: as_date(x[1]) < as_date(date)).cache()

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

year_temps = year_temps.map(lambda x: (x[0],(diff_date(x[1], date, h_date),
                                       diff_time(x[2], "04:00:00", h_time),
                                       diff_time(x[2], "06:00:00", h_time),
                                       diff_time(x[2], "08:00:00", h_time),
```

```
                                        diff_time(x[2], "10:00:00", h_time),
                                        diff_time(x[2], "12:00:00", h_time),
                                        diff_time(x[2], "14:00:00", h_time),
                                        diff_time(x[2], "16:00:00", h_time),
                                        diff_time(x[2], "18:00:00", h_time),
                                        diff_time(x[2], "20:00:00", h_time),
                                        diff_time(x[2], "22:00:00", h_time),
                                        diff_time(x[2], "00:00:00", h_time),
                                        x[3]))).cache()

year_temps_k = year_temps.map(lambda x: (x[0],(kernel(x[1][0],h_date),
                                                kernel(x[1][1],h_time),
                                                kernel(x[1][3],h_time),
                                                kernel(x[1][4],h_time),
                                                kernel(x[1][5],h_time),
                                                kernel(x[1][6],h_time),
                                                kernel(x[1][8],h_time),
                                                kernel(x[1][9],h_time),
                                                kernel(x[1][11],h_time),
                                                x[1][12]))).cache()

stations = station.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)
joined = year_temps_k.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 = joined.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])).cache()

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(times,rdd)
```

kernel product

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

h_distance = 100 # Up to you
h_date = 10 # Up to you
h_time = 1 # Up to you
a = 57.7236 # Up to you
b = 12.9641 # 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"]

date = "2013-11-02"# Up to you

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

def as_date(date):
```

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

def diff_date(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
    return dd

def diff_time(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
    return dt

temperature_file = sc.textFile("BDA/input/temperature-readings.csv")
stations_file = sc.textFile("BDA/input/stations.csv")

temps = temperature_file.map(lambda line: line.split(";"))
station = stations_file.map(lambda line: line.split(";"))

lines = temps.filter(lambda x: as_date(x[1]) < as_date(date)).cache()

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

year_temps = year_temps.map(lambda x: (x[0],(diff_date(x[1], date, h_date),
                                        diff_time(x[2], "04:00:00", h_time),
                                        diff_time(x[2], "06:00:00", h_time),
                                        diff_time(x[2], "08:00:00", h_time),
                                        diff_time(x[2], "10:00:00", h_time),
                                        diff_time(x[2], "12:00:00", h_time),
                                        diff_time(x[2], "14:00:00", h_time),
                                        diff_time(x[2], "16:00:00", h_time),
                                        diff_time(x[2], "18:00:00", h_time),
                                        diff_time(x[2], "20:00:00", h_time),
                                        diff_time(x[2], "22:00:00", h_time),
                                        diff_time(x[2], "00:00:00", h_time),
                                        x[3]))).cache()

year_temps_k = year_temps.map(lambda x: (x[0],(kernel(x[1][0],h_date),
                                               kernel(x[1][1],h_time),
                                               kernel(x[1][3],h_time),
                                               kernel(x[1][4],h_time),
                                               kernel(x[1][5],h_time),
                                               kernel(x[1][6],h_time),
                                               kernel(x[1][8],h_time),
                                               kernel(x[1][9],h_time),
                                               kernel(x[1][11],h_time),
```

k

```
                                                x[1][12]))).cache()

stations = station.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)
joined = year_temps_k.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 = joined.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])).cache()

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(times,rdd)
```