732A54 Big Data Analytics

LAB EXERCISE 3: MACHINE LEARNING

Group G6:

Hoda Fakharzadehjahromy(hodfa840)
Seyda Aqsa Iftekhar(syeif776)

RESOURCES

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:

- The first to account for the distance from a station to the point of interest.
- The second to account for the distance between the day a temperature measurement was made and the day of interest.
- 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.

Questions

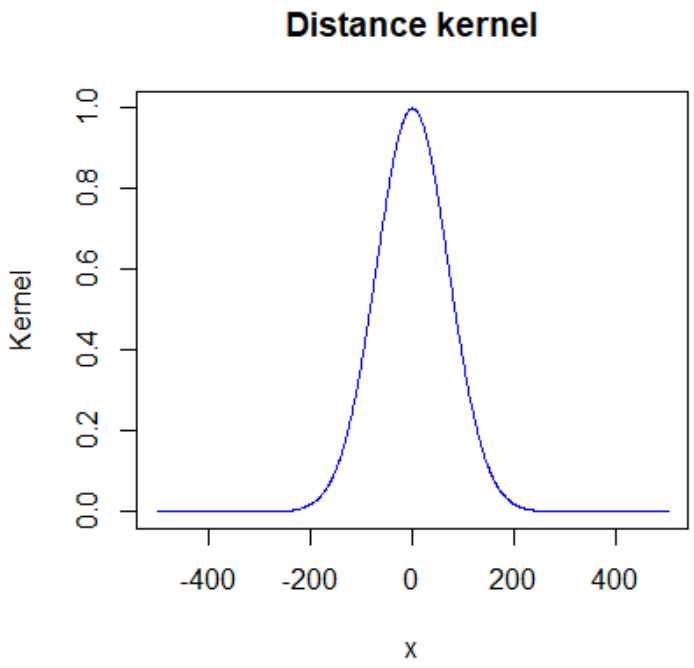
- Show that your choice for the kernels' width is sensible, i.e. it gives more weight to closer points. Discuss why your definition of closeness is reasonable.

We chose the following values for h (the smoothing factor):

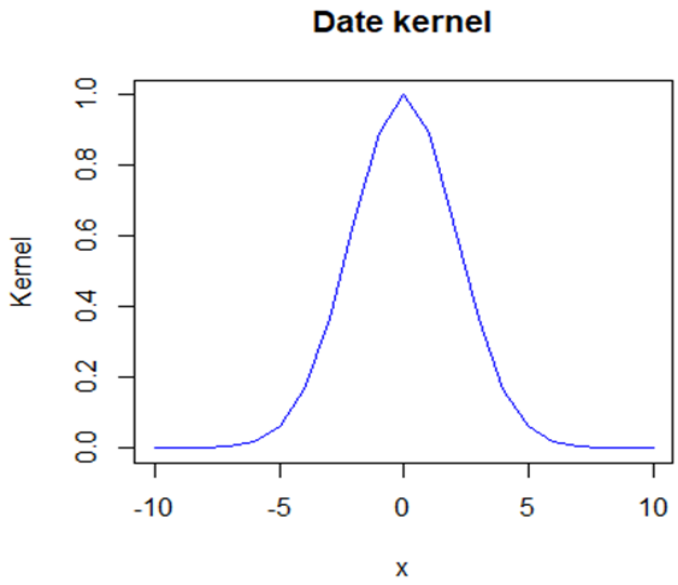
```
h_distance = 100
h_date = 3
h_time = 2*60*60
```

as we expect a temperature in the specific day be close to the temperature of the pints that are in 100 km radius in the 2-hours interval for the past 3 days.

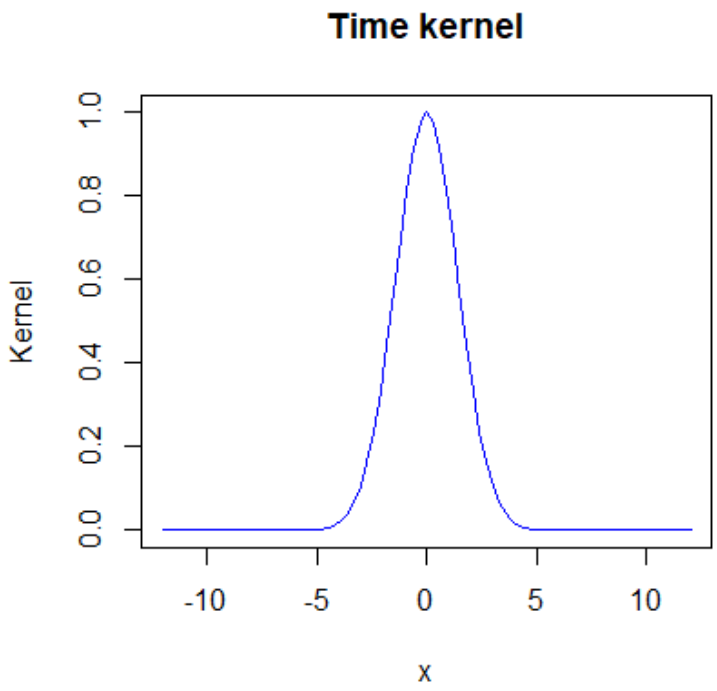
We wanted the points that are closer to the point we want to predict have higher effect. The Sum kernel, however, makes the results of closeness independent of one another and if the point is close in distance but its date and time are far away from the prediction point the kernel might still results in high weigh value for this point as we are summing all the kernels. The sum kernel might not take into account the seasonal time.



We can see that for 200 km above the kernel is close to zero. We decided to choose 100 km as the definition of closeness in distance kernel.



we can see for value of days above 5 days the kernel is close to zero, hence we selected 3 the definition of closeness in date kernel



we can see for value of days above 4 days the kernel is close to zero, hence we selected 2 the definition of closeness in time kernel

Output:

For the Sum kernels the Outputs are:

```
#time      #temperature

24:00:00 4.956963840632157
22:00:00 4.390270895626881
20:00:00 4.73056622323275
18:00:00 5.225133831666042
16:00:00 5.90808655965257
14:00:00 6.447249257762833
12:00:00 6.459955723343102
10:00:00 5.911026104187175
08:00:00 4.787300017744497
06:00:00 3.811075352261145
04:00:00 3.524423264627072
```

we can notice that the lower temperatures in the morning and evening and a higher temperatures in the afternoon.

- Repeat the exercise using a kernel that is the product of the three Gaussian kernels above. Compare the results with those obtained for the additive kernel. If they differ, explain why.

  - Using the product of the kernels rather than their sum causes the kernels to be more dependent to one another, since, if one kernel is close to zero (ie. is far from the prediction point), the whole kernel will also become close to zero.The results of the product kernel for prediction seems more reasonable as one expect the temperature be higher in June. So, the preferred kernel is the product of kernels rather than sum.

For the product Kernel the Output are:

```
 #time     #temperature

24:00:00 12.97414991694815
22:00:00 13.60255390985128
20:00:00 15.091835167497688
18:00:00 16.82984952770218
16:00:00 17.891465240443626
14:00:00 18.230131420493368
12:00:00 18.077017906121554
10:00:00 17.350201552289146
08:00:00 15.837802114915416
06:00:00 14.042095285551977
04:00:00 12.299308338036251
```

Same as the sum Kernel, we can notice that the lower temperatures in the morning and evening and a higher temperatures in the afternoon.

APPENDIX

Code:

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


def convert_to_days(a, b):
    return (datetime.strptime(a, "%Y-%m-%d") - datetime.strptime(b, "%Y-%m-%d")).days

def convert_to_secs(time):
    return ((int(time.split(":")[0]) * 3600)+ int(time.split(":")[1]*60)+
int(time.split(":")[2]*1))

def gauss_kernel(distance, days, time):
    dist_weight = exp(-(distance/h_distance)**2)
    date_weight = exp(-(days/h_date)**2)
    time_weight = exp(-(time/h_time)**2)
   # return dist_weight + date_weight + time_weight
    return dist_weight * date_weight * time_weight


h_distance = 100# Up to you
```

```
h_date = 3# Up to you
h_time = 2*60*60 # Up to you
a = 58.4274 # Up to you
b = 14.826 # Up to you
date = "2013-06-04" # Up to you

stations = sc.textFile("data/stations.csv").map(lambda x: x.split(';'))
# map stattion to station_haversinedistance
stations = stations.map(
   lambda x: (x[0], haversine(float(x[3]), float(x[4]), a, b))).collectAsMap()

temperature = sc.textFile("data/temperature-readings.csv")

#temperature = temperature.sample(False,0.01)

temperature = temperature.map(lambda x: x.split(';'))

#filtering the date

temperature = temperature.filter(lambda x: x[1] < date)
temperature = temperature.map(lambda x: (
                        stations[x[0]],
                        convert_to_days(x[1], date),
                        convert_to_secs(x[2]),
                        float(x[3])
                    )).cache()

stdoutOrigin=sys.stdout
sys.stdout = open("sum_result2h.txt", "w")
#sys.stdout = open("product_result.txt", "w")
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"]:
   secs = convert_to_secs(time)
   pred = temperature.map(lambda x: (x[3], gauss_kernel(x[0], x[1], secs - x[2])))
   pred = pred.map(lambda x: (x[0] * x[1], x[1]))
   pred = pred.reduce(lambda x, y: (x[0] + y[0], x[1] + y[1]))
   pred = pred[0] / pred[1]


   print(time, pred)

sys.stdout.close()
sys.stdout=stdoutOrigin
```