BDA3 - Machine Learning with Spark

Roshni Sundaramurthy (rossu809) and Yusur Al-Mter (yusal621)

May 18, 2019

Contents

1 Exercise

Implement in Spark1 (PySpark) a kernel method 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. Specifically, the forecast should consist of the predicted temperatures from 4 am (04:00) to 12 am (00:00) in an interval of 2 hours for a date and place in Sweden. Use a kernel that is the sum of three Gaussian kernels:

1. The first to account for the distance from a station to the point of interest.
2. The second to account for the distance between the day a temperature measurement was made and the day of interest.
3. 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.

2 Result

The three gaussian kernels used are:

Distance kernel, time kernel, and day kernel

Choosing values for the three distance widths matters since we care about the closest to get a good prediction, so only the points that are closer to our interest gets higher weights. Setting smaller values for widths, we overfit the temperature. While choosing higher values, it gives the average of all the temperature. An appropriate smoothing coefficient or width for each of the three kernels is necessary as it gives more weight to closer points.

The choosen smoothing coefficient / width

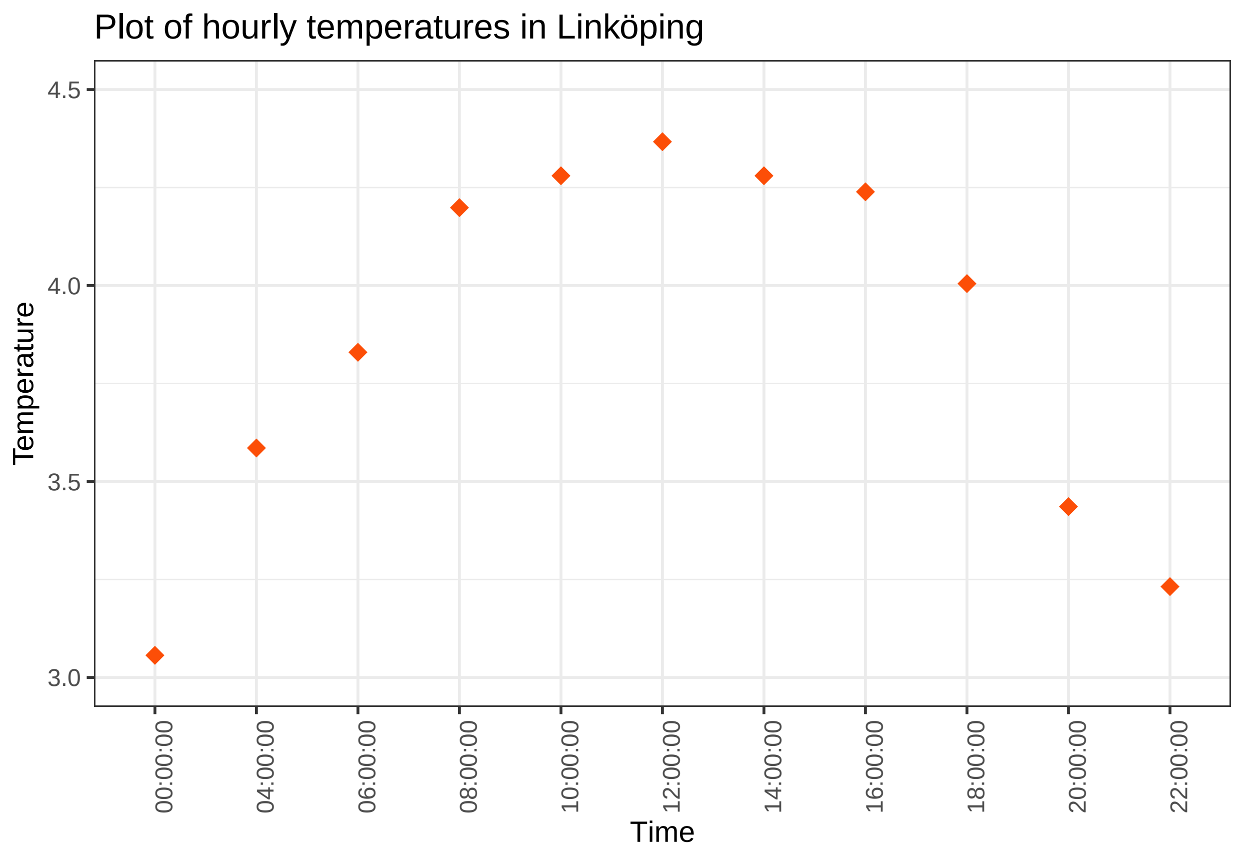
h_distance = 100 kms
h_date = 2 (temperature close to 2 days)
h_time = 2 (temperature close to 2 hours)

The appropriate smoothing coefficient ensures that the predictions have a higher accuracy. Taking the above h values, it was found that the kernels produced better predictions.

```r
library(ggplot2)

temp = read.csv("hourly_temp.csv")
```

```
# plot
ggplot(temp) + geom_point(aes(x = Time, y = Tempp), shape = 18, color = "#FC4E07", size = 3) +
              ylab("Temperature")+ scale_y_continuous(limits=c(3,4.5))+
              ggtitle("Plot of hourly temperatures in Linköping")+ theme_bw()+
              theme(axis.text.x = element_text(angle = 90))
```



Above is the plot for additive kernels. Both the additive and multiplicative kernels have different effects on predicting the temperature, since if we have one value for width closer or equal to zero, it will affect the total weight for both summing and multiplying the kernels in a different approach and will end up in different results. Using multiplicative kernel can be better than this approach.

Code

```
# Imports
from __future__ import division
from math import radians, cos, sin, asin, sqrt, exp
from datetime import datetime, timedelta, date
from pyspark import SparkContext
sc = SparkContext(appName="lab_kernel")
```

```
# distance between two points
def haversine(lon1, lat1, lon2, lat2):
    # 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 # radius of earth (6371 km)
    return km

# reasonable smoothing coefficient / width
h_distance = 100
h_date = 2
h_time = 2

a = 58.4274  # latitude
b = 14.826  # longitude
predict_date = "2014-02-05"
predict_time = [datetime.strptime("04:00:00","%H:%M:%S") +
                  timedelta(hours=2*i) for i in range(0, 11)]
pred_t = len(predict_time)

station = sc.textFile("/user/x_rossu/bdlab1/stations.csv")
station = station.map(lambda x: x.split(";"))
station = station.map(lambda x: (x[0],(x[3],x[4]))) # stn, a, b
stations = station.collectAsMap()
stations = sc.broadcast(stations) # broadcasting all nodes in a cluster

temp_file = sc.textFile("/user/x_rossu/bdlab1/temperature-readings.csv")
lines = temp_file.map(lambda line: line.split(";"))
temps = lines.map(lambda x:(x[0],(x[1],x[2],float(x[3]))))
# filtering temps file based on date and time
temps = temps.filter(lambda x : (x[1][0]<=predict_date)).filter(lambda x: x[1][1] < "04:00:00")
temps.cache()
#temps.take(5)


# temp + station file combine
temps = temps.map(lambda x: (x[0], (stations.value.get(x[0],), x[1])))
temps.take(5)

# stn, latitude, longitude, day, time, temp
temps1 = temps.map(lambda x: (x[0], (x[1][0][0], x[1][0][1], x[1][1][0],
                                      x[1][1][1], x[1][1][2])))
temps1.take(5)

# functions
def calc_days(date1,date2):
    d0 = datetime.strptime(date1, '%Y-%m-%d')
    d1 = datetime.strptime(date2,'%Y-%m-%d')
    diff = (d0 - d1).days
    return abs(diff)

def gaus_kernel(dist, h):
    u = dist/h
    k = exp(-u**2)
    return k
```

```
#def calc_hour(pred_time, st_time):
#    diff = datetime.strptime(st_time, '%H:%M:%S') - datetime.strptime(pred_time,'%H:%M:%S')
#    diff = diff.total_seconds() / diff.seconds
#    diff = diff/3600 # seconds to hour
#    return diff

# dict to save outputs
from collections import OrderedDict
predicttemp = OrderedDict()

# calc for every 2 hours time # kernel sum
for time in range(pred_t):
    temp1 = temps1.map(lambda x: (x[0], (haversine(b, a,float(x[1][1]), float(x[1][0])),
    calc_days(x[1][2],predict_date),(datetime.strptime(x[1][3], '%H:%M:%S') -
                                       predict_time[time]).seconds/3600,x[1][4])))
    temp1 = temp1.map(lambda x: (0, (gaus_kernel(x[1][0],h_distance),
                                     gaus_kernel(x[1][1], h_date),
                                     gaus_kernel(x[1][2], h_time), x[1][3])))
    temp1 = temp1.map(lambda x: (x[0], (x[1][0] * x[1][3] + x[1][1]* x[1][3] +
                                        x[1][2] * x[1][3],
                                        x[1][0] + x[1][1]+ x[1][2])))
    temp1 = temp1.reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))
    temp1 = temp1.map(lambda x: x[1][0] / x[1][1])
    predicttemp[predict_time[time].strftime("%H:%M:%S")] = temp1.take(1)


# save output
out = []
for time,temp in predicttemp.items() :
    out.append((time,temp[0]))

print(out) # print time and temperature
```