lab3

Zhixuan_Duan(zhidu838)

2020_07_05

Load packages

```
library(ggplot2)
```

pyspark code

```
from __future__ import division
from math import radians, cos, sin, asin, sqrt, exp
from datetime import datetime
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 = 1000
h_date = 30
h_time = 6
a = 58.4274
b = 14.826
date = "2013-11-04"

# pre-setting
stations = sc.textFile("/Users/darin/Desktop/Big_Data/data/stations.csv")
temp = sc.textFile("/Users/darin/Desktop/Big_Data/data/temperature-readings.csv")
result3 = "/Users/darin/Desktop/Big_Data/data/lab3"

stations = stations.map(lambda line: line.split(";"))
stations = stations.map(lambda x: (int(x[0]), float(x[3]),float(x[4])))

temp = temp.map(lambda line: line.split(";"))
```

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

times = ["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"]

output = [0]*len(times)

# compute 3 kernels based on formula
def K_dist(data,coords, h):
    u = data.map(lambda x: (x[0],haversine(x[2],x[1],coords[0],coords[1])/h))
    k = u.map(lambda x: (x[0],exp(-(x[1]**2))))
    return k

def K_date(x,date,h):
    diff_date = (datetime(int(x[0:4]),int(x[5:7]),int(x[8:10]))
                 - datetime(int(date[0:4]),int(date[5:7]),int(date[8:10]))).days / h
    k = exp(-(diff_date ** 2))
    return k

def K_time(x,time,h):
    diff_time = (datetime(2000,1,1,int(x[0:2]),int(x[3:5]),int(x[6:8]))
                 - datetime(2000,1,1,int(time[0:2]),int(time[3:5]),int(time[6:8]))).seconds / h
    k = exp(-(diff_time**2))
    return k

# use add and mul models to predict temp
def predict():
    k_dist = K_dist(stations,[b,a],h_distance)
    k_dist_broadcast = k_dist.collectAsMap()
    stations_dist = sc.broadcast(k_dist_broadcast)

    # choose date
    filtered_dates = temp.filter(lambda x: (datetime(int(x[1][0:4]),int(x[1][5:7]),
                                                     int(x[1][8:10])) <= datetime(int(date[0:4]),
                                                                                  int(date[5:7]),
                                                                                  int(date[8:10]))))
    filtered_dates.cache()

    # choose time
    for time in times:
        filtered_times = filtered_dates.filter(lambda x: ((datetime(int(x[1][0:4]),
                                                                    int(x[1][5:7]),
                                                                    int(x[1][8:10]))== datetime(int(dat
                                                                                                int(dat
                                                                                                int(dat

                                                          )
                                                         ) and
                                                         (datetime(2000, 1, 1,
                                                                   int(x[2][0:2]),
                                                                   int(x[2][3:5]),
                                                                   int(x[2][6:8])) <= datetime(2000, 1,
                                                                                               int(time[
                                                                                               int(time[
```

```
                                                    )
                                                )
        # get the weighted kernel
        kernel = filtered_times.map(lambda x: (stations_dist.value[x[0]],
                                               K_date(x[1], date, h_date),
                                               K_time(x[2], time, h_time),
                                               x[3]))

        # define two kernels
        k_sum = kernel.map(lambda x: (x[0] + x[1] + x[2], x[3]))
        # k_mul = kernel.map(lambda x: (x[0] * x[1] * x[2], x[3]))

        k_sum = k_sum.map(lambda x: (x[0] * x[1], x[0]))
        # k_mul = k_mul.map(lambda x: (x[0] * x[1], x[0]))

        k_sum = k_sum.reduce(lambda x, y: (x[0] + y[0], x[1] + y[1]))
        # k_mul = k_mul.reduce(lambda x, y: (x[0] + y[0], x[1] + y[1]))
        output[times.index(time)] = (time, k_sum[0] / k_sum[1])
        #output[times.index(time)] = (time, k_mul[0] / k_mul[1])

predict()
output.saveAsTextFile(result3)
```

Kernels

the distance kernel

The first to account for the distance from a station to the point of interest.

the date kernel

The second to account for the distance between the day a temperature measurement was made and the day of interest.

the time kernel

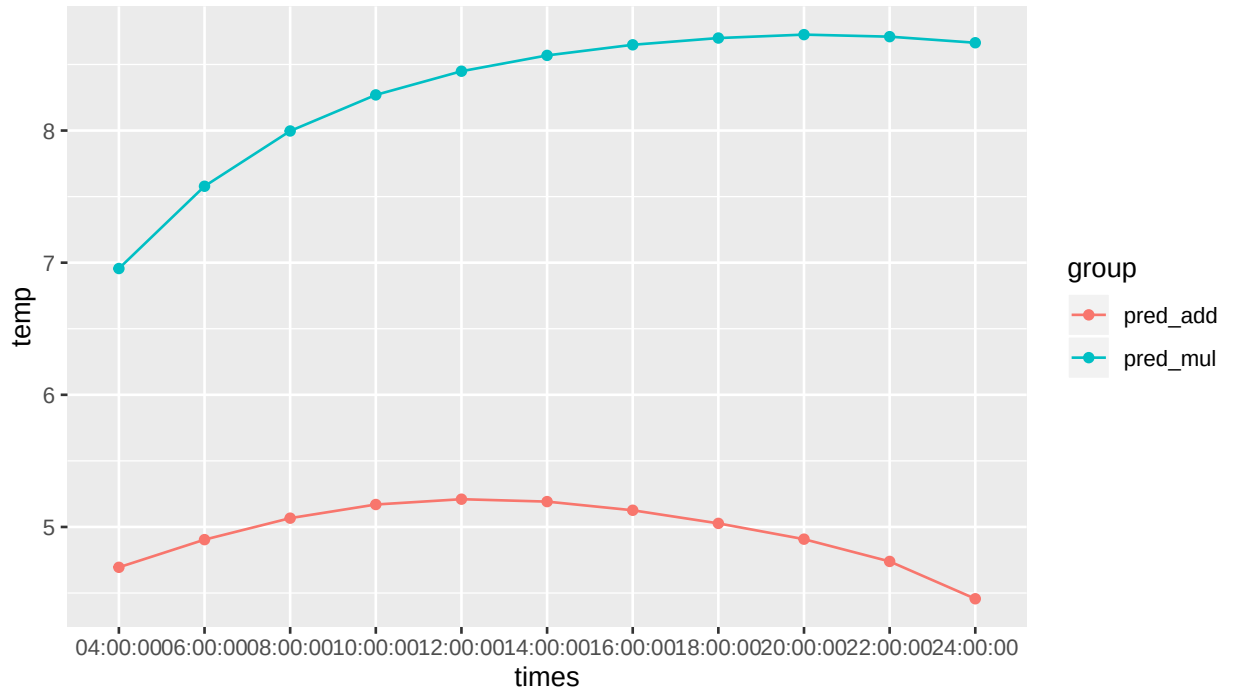
The third to account for the distance between the hour of the day a temperature measurement was made and the hour of interest.

Compare the kernel widths

```
h_distance <- 1000
h_date <- 100
h_time <- 10

# result1
data1 <- read.csv("/Users/darin/Desktop/result1")
```

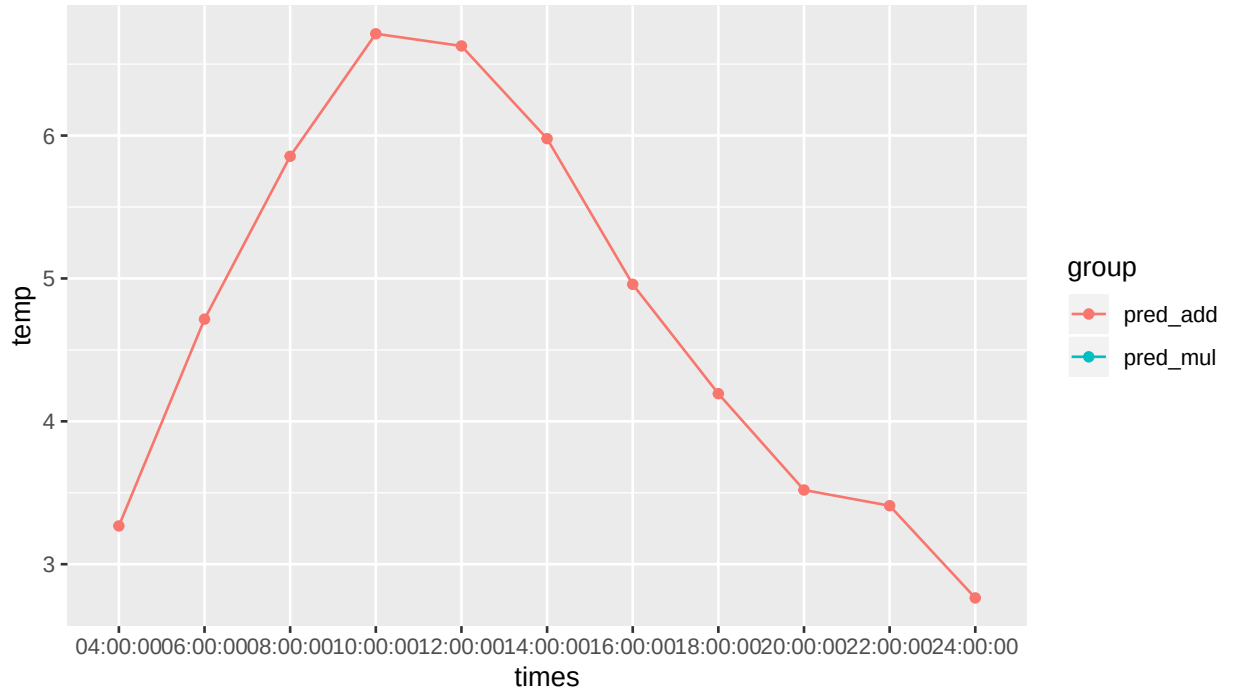
```
ggplot(data1, aes(x = times, y = temp, group = group)) + geom_line(aes(color = group)) +
  geom_point(aes(color = group))
```




```
h_distance <- 100
h_date <- 10
h_time <- 1

# result2
data2 <- read.csv("/Users/darin/Desktop/result2")

ggplot(data2, aes(x = times, y = temp, group = group)) + geom_line(aes(color = group)) +
  geom_point(aes(color = group))
```



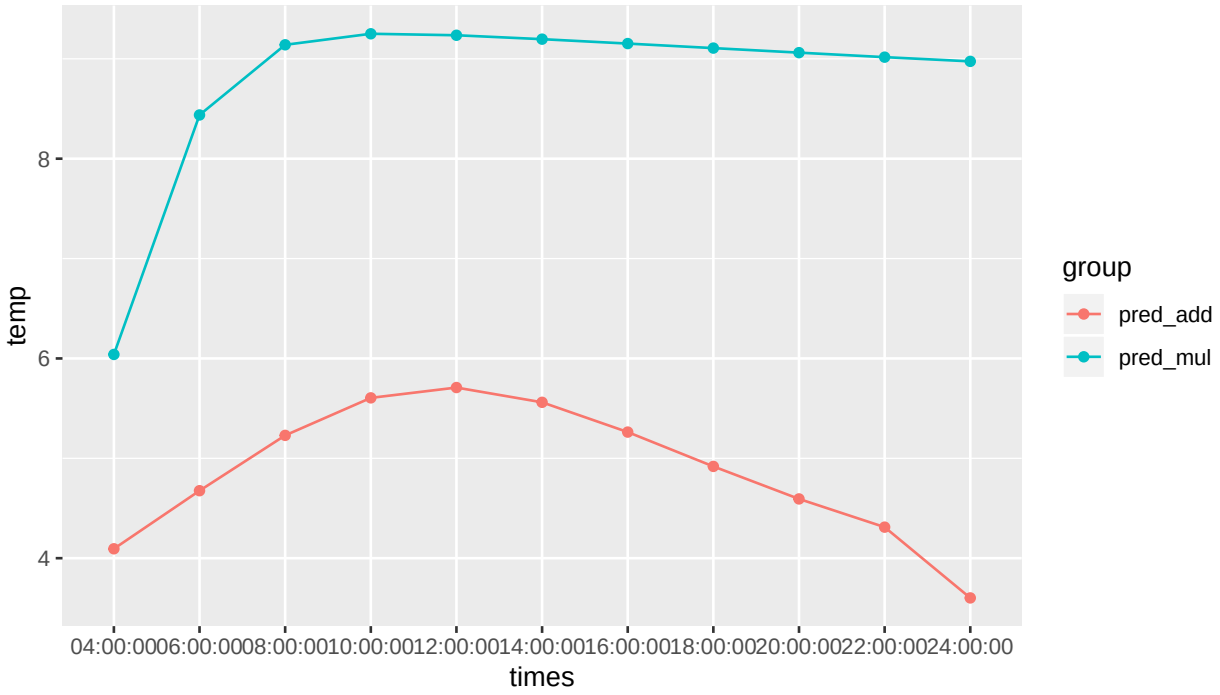
Analysis: We select two groups of kernel width to predict the temperature, and from these results, we choose h_distance as 1000 considering the Sweden land area. The h_date is 30, and the h_time is 6, since the temperature of the whole year is relatively smooth, but the temperature during the day may change significantly, these two values would give closer values more weight.

Compare the add and mul kernel models

```
h_distance <- 1000
h_date <- 30
h_time <- 6

# result3
data3 <- read.csv("/Users/darin/Desktop/result3")

ggplot(data3, aes(x = times, y = temp, group = group)) + geom_line(aes(color = group)) +
  geom_point(aes(color = group))
```



Analysis: From the plot, we found that the cumulative model has greater volatility, while the cumulative model is gentler. The reason for this phenomenon may be that the multiplicative model is more sensitive to outliers than the cumulative model, resulting in a greater influence in the final model.

Appendix

```
knitr::opts_chunk$set(echo = TRUE,
                      warning = FALSE,
                      message = FALSE,
                      fig.width = 7,
                      fig.height = 4,
                      fig.align = 'center')
library(ggplot2)
```

```
from __future__ import division
from math import radians, cos, sin, asin, sqrt, exp
from datetime import datetime
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 = 1000
h_date = 30
h_time = 6
a = 58.4274
b = 14.826
date = "2013-11-04"

# pre-setting
stations = sc.textFile("/Users/darin/Desktop/Big_Data/data/stations.csv")
temp = sc.textFile("/Users/darin/Desktop/Big_Data/data/temperature-readings.csv")
result3 = "/Users/darin/Desktop/Big_Data/data/lab3"

stations = stations.map(lambda line: line.split(";"))
stations = stations.map(lambda x: (int(x[0]), float(x[3]),float(x[4])))

temp = temp.map(lambda line: line.split(";"))
temp = temp.map(lambda x: (int(x[0]),x[1],x[2],float(x[3])))

times = ["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"]

output = [0]*len(times)
```

```
# compute 3 kernels based on formula
def K_dist(data,coords, h):
    u = data.map(lambda x: (x[0],haversine(x[2],x[1],coords[0],coords[1])/h))
    k = u.map(lambda x: (x[0],exp(-(x[1]**2))))
    return k

def K_date(x,date,h):
    diff_date = (datetime(int(x[0:4]),int(x[5:7]),int(x[8:10]))
                 - datetime(int(date[0:4]),int(date[5:7]),int(date[8:10]))).days / h
    k = exp(-(diff_date ** 2))
    return k

def K_time(x,time,h):
    diff_time = (datetime(2000,1,1,int(x[0:2]),int(x[3:5]),int(x[6:8]))
                 - datetime(2000,1,1,int(time[0:2]),int(time[3:5]),int(time[6:8]))).seconds / h
    k = exp(-(diff_time**2))
    return k

# use add and mul models to predict temp
def predict():
    k_dist = K_dist(stations,[b,a],h_distance)
    k_dist_broadcast = k_dist.collectAsMap()
    stations_dist = sc.broadcast(k_dist_broadcast)

    # choose date
    filtered_dates = temp.filter(lambda x: (datetime(int(x[1][0:4]),int(x[1][5:7]),
                                                     int(x[1][8:10])) <= datetime(int(date[0:4]),
                                                                                  int(date[5:7]),
                                                                                  int(date[8:10]))))
    filtered_dates.cache()

    # choose time
    for time in times:
        filtered_times = filtered_dates.filter(lambda x: ((datetime(int(x[1][0:4]),
                                                                    int(x[1][5:7]),
                                                                    int(x[1][8:10]))== datetime(int(dat
                                                                                                int(dat
                                                                                                int(dat
                                                           )
                                                          ) and
                                                          (datetime(2000, 1, 1,
                                                                    int(x[2][0:2]),
                                                                    int(x[2][3:5]),
                                                                    int(x[2][6:8])) <= datetime(2000, 1,
                                                                                                int(time[
                                                                                                int(time[
                                                                                                int(time[
                                                           )
                                                 )
        # get the weighted kernel
        kernel = filtered_times.map(lambda x: (stations_dist.value[x[0]],
                                               K_date(x[1], date, h_date),
                                               K_time(x[2], time, h_time),
```

```
                                            x[3]))

        # define two kernels
        k_sum = kernel.map(lambda x: (x[0] + x[1] + x[2], x[3]))
        # k_mul = kernel.map(lambda x: (x[0] * x[1] * x[2], x[3]))

        k_sum = k_sum.map(lambda x: (x[0] * x[1], x[0]))
        # k_mul = k_mul.map(lambda x: (x[0] * x[1], x[0]))

        k_sum = k_sum.reduce(lambda x, y: (x[0] + y[0], x[1] + y[1]))
        # k_mul = k_mul.reduce(lambda x, y: (x[0] + y[0], x[1] + y[1]))
        output[times.index(time)] = (time, k_sum[0] / k_sum[1])
        #output[times.index(time)] = (time, k_mul[0] / k_mul[1])

predict()
output.saveAsTextFile(result3)
h_distance <- 1000
h_date <- 100
h_time <- 10

# result1
data1 <- read.csv("/Users/darin/Desktop/result1")

ggplot(data1, aes(x = times, y = temp, group = group)) + geom_line(aes(color = group)) +
  geom_point(aes(color = group))
h_distance <- 100
h_date <- 10
h_time <- 1

# result2
data2 <- read.csv("/Users/darin/Desktop/result2")

ggplot(data2, aes(x = times, y = temp, group = group)) + geom_line(aes(color = group)) +
  geom_point(aes(color = group))
h_distance <- 1000
h_date <- 30
h_time <- 6

# result3
data3 <- read.csv("/Users/darin/Desktop/result3")

ggplot(data3, aes(x = times, y = temp, group = group)) + geom_line(aes(color = group)) +
  geom_point(aes(color = group))
```