732A54 - Big Data Analytics

BDA3 Lab

Sridhar Adhikarla (sriad858), Obaid Ur Rehman (obaur539)

Introduction:

The temperature below were predicted for Linkoping area on the date 2013-01-25. The month of January is usually cold in Linkoping and the temperatue is around 0. The addition model, which is a linear combination of the kernels, gives too high temperatues for the day. This clearly shows that it is not able to capture the seasonal trend. We have made some efforts in this assignment to come up with a better implementation that could capture the seasonal trend by adding some dependence on date of the year to the prediction.

Outputs:

Addition model outputs

```r
temps_add <- read.csv('outputs/add.csv', header = TRUE, sep = ',')
temps_add
```

```
##        time       h1       h2       h3       h4       h5       h6
## 1  04:00:00 3.395501 3.753910 4.258891 4.134047 4.606060 4.826301
## 2  06:00:00 3.812175 4.091880 4.499197 4.469413 4.860225 5.042039
## 3  08:00:00 4.711138 4.902008 5.179051 5.041074 5.234712 5.365916
## 4  10:00:00 5.755894 5.828434 5.938870 5.657237 5.617249 5.699383
## 5  12:00:00 6.259156 6.272850 6.300911 6.045626 5.880867 5.929725
## 6  14:00:00 6.264103 6.277893 6.306150 6.087803 5.955869 5.995501
## 7  16:00:00 5.816690 5.881716 5.981794 5.849600 5.863187 5.915568
## 8  18:00:00 5.228191 5.354968 5.543798 5.505569 5.690420 5.768589
## 9  20:00:00 4.714490 4.925444 5.221781 5.219044 5.537444 5.644983
## 10 22:00:00 4.381235 4.719249 5.141322 5.105839 5.482517 5.613584
## 11 00:00:00 3.547491 3.946212 4.478320 4.241471 4.644345 4.868918
```

Multiplication model outputs

```r
temps_mul <- read.csv('outputs/mul.csv', header = TRUE, sep = ',')
temps_mul
```

```
##        time          h1          h2          h3          h4         h5
## 1  04:00:00 -0.07705943 -0.07683965 -0.10014620 -0.11304652 -0.1240935
## 2  06:00:00 -0.12522279 -0.12370583 -0.15043273 -0.13818508 -0.1313857
## 3  08:00:00 -0.12099088 -0.12779394 -0.15622560 -0.14877405 -0.1374583
## 4  10:00:00 -0.11007640 -0.11363345 -0.13134080 -0.14246630 -0.1409973
## 5  12:00:00 -0.10891861 -0.11228854 -0.12384842 -0.13580460 -0.1439371
## 6  14:00:00 -0.10927254 -0.11195458 -0.12212357 -0.13350926 -0.1484636
## 7  16:00:00 -0.10673836 -0.10833659 -0.12205943 -0.13684229 -0.1564836
```

```
## 8  18:00:00 -0.11242187 -0.11129251 -0.14947277 -0.15282159 -0.1688748
## 9  20:00:00 -0.18093373 -0.18019273 -0.19633990 -0.18387905 -0.1832388
## 10 22:00:00 -0.19217177 -0.19557352 -0.19836203 -0.19998309 -0.1939031
## 11 00:00:00 -0.06979352 -0.06272680 -0.08276777 -0.09817603 -0.1218290
##            h6
## 1  -0.1264274
## 2  -0.1311233
## 3  -0.1370032
## 4  -0.1432301
## 5  -0.1508243
## 6  -0.1607675
## 7  -0.1734553
## 8  -0.1868485
## 9  -0.1960719
## 10 -0.1987291
## 11 -0.1306597
```

Plot:

Different distances considered

```
distances <-  data.frame(name=colnames(temps_add)[-1],
                         time_days_kms=c("(2, 5, 50)", "(2, 5, 70)", "(2, 7, 100)",
                                      "(3, 8, 100)", "(4, 9, 150)", "(4, 10, 200)"))
distances
```

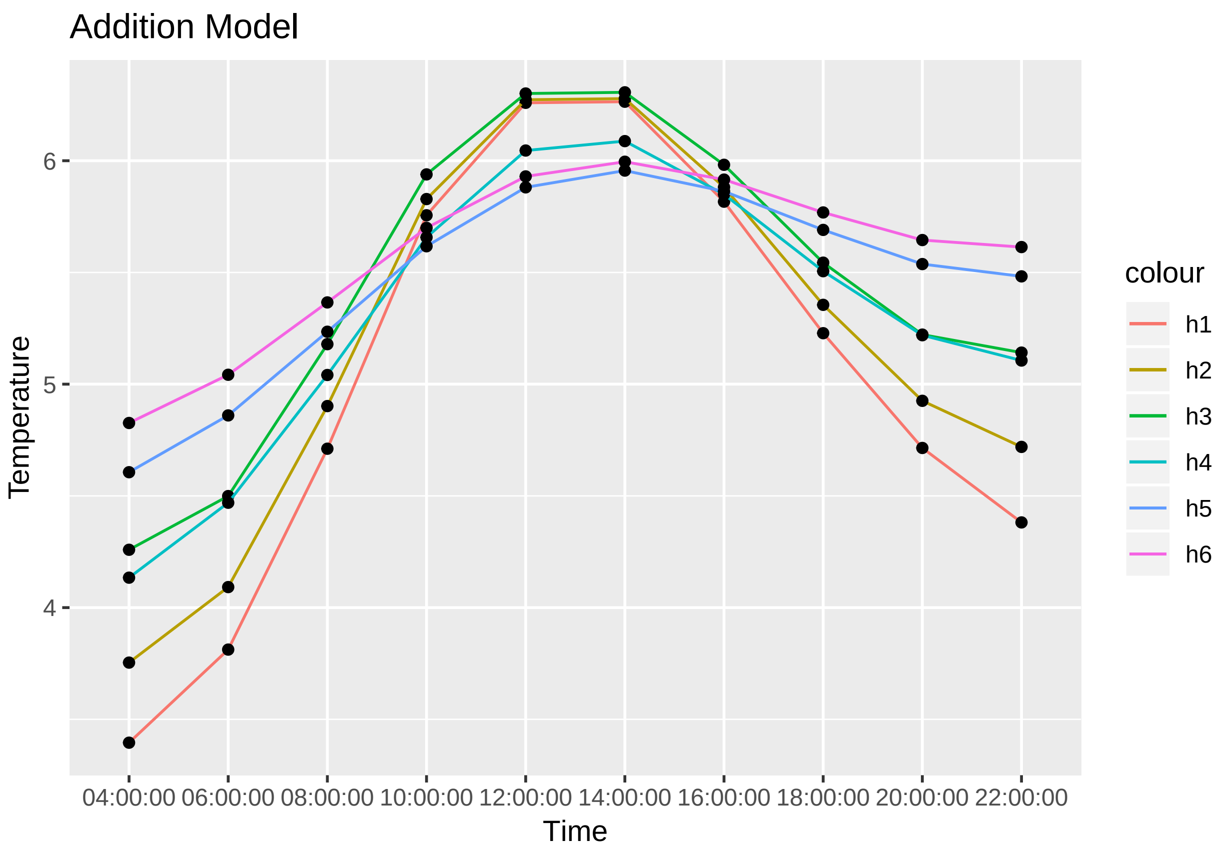
```
##   name time_days_kms
## 1   h1    (2, 5, 50)
## 2   h2    (2, 5, 70)
## 3   h3   (2, 7, 100)
## 4   h4   (3, 8, 100)
## 5   h5   (4, 9, 150)
## 6   h6  (4, 10, 200)
```

Kernel Add prediction

```
library(ggplot2)

ggplot(temps_add[1:10,]) +
  geom_line(aes(x = time, y = h1, group=1, col="h1")) +
  geom_point(aes(x = time, y = h1, group=1)) +
  geom_line(aes(x = time, y = h2, group=1, col="h2")) +
  geom_point(aes(x = time, y = h2, group=1)) +
  geom_line(aes(x = time, y = h3, group=1, col="h3")) +
  geom_point(aes(x = time, y = h3, group=1))+
  geom_line(aes(x = time, y = h4, group=1, col="h4")) +
  geom_point(aes(x = time, y = h4, group=1)) +
  geom_line(aes(x = time, y = h5, group=1, col="h5")) +
  geom_point(aes(x = time, y = h5, group=1)) +
  geom_line(aes(x = time, y = h6, group=1, col="h6")) +
```

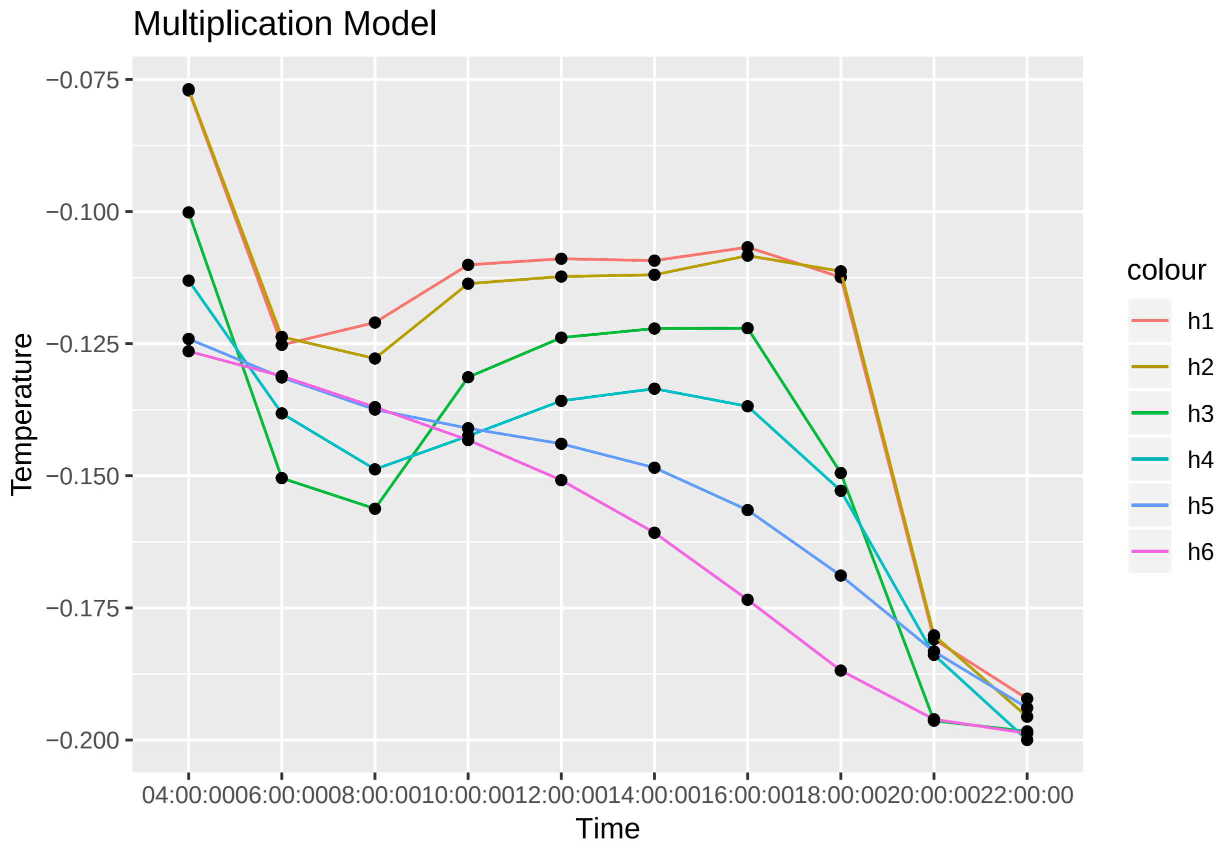
```
  geom_point(aes(x = time, y = h6)) +
  labs(x = "Time", y = "Temperature", title = "Addition Model")
```



Kernel Multiply prediction

```
library(ggplot2)

ggplot(temps_mul[1:10,]) +
  geom_line(aes(x = time, y = h1, group=1, col="h1")) +
  geom_point(aes(x = time, y = h1, group=1)) +
  geom_line(aes(x = time, y = h2, group=1, col="h2")) +
  geom_point(aes(x = time, y = h2, group=1)) +
  geom_line(aes(x = time, y = h3, group=1, col="h3")) +
  geom_point(aes(x = time, y = h3, group=1))+
  geom_line(aes(x = time, y = h4, group=1, col="h4")) +
  geom_point(aes(x = time, y = h4, group=1)) +
  geom_line(aes(x = time, y = h5, group=1, col="h5")) +
  geom_point(aes(x = time, y = h5, group=1)) +
  geom_line(aes(x = time, y = h6, group=1, col="h6")) +
  geom_point(aes(x = time, y = h6)) +
  labs(x = "Time", y = "Temperature", title = "Multiplication Model")
```



From these plots we can see that as we increase the distances the prediction for the day keeps getting flatter. This is because it is using more data than needed for the prediction of temperature of that place. Selecting a distance somewhere in the middle like h2 (2 hours, 5 days, 70 kms). This covers most of the required dependence of the nearby area and close days.

Date kernel:

We think 5 days is reasonable, as the temperature will stop having much dependence on the temperature more than 5 days ago.

Distance kernel:

Distance of 70 kms is selected in the final model. This gives a reasonable radius of areas to cover, to predict the temperature.

Time kernel:

The time kernel has higher dependence on up to 2 hours before temperatures. This gives the model a reasonable estimate to predict temperature according to time to day.

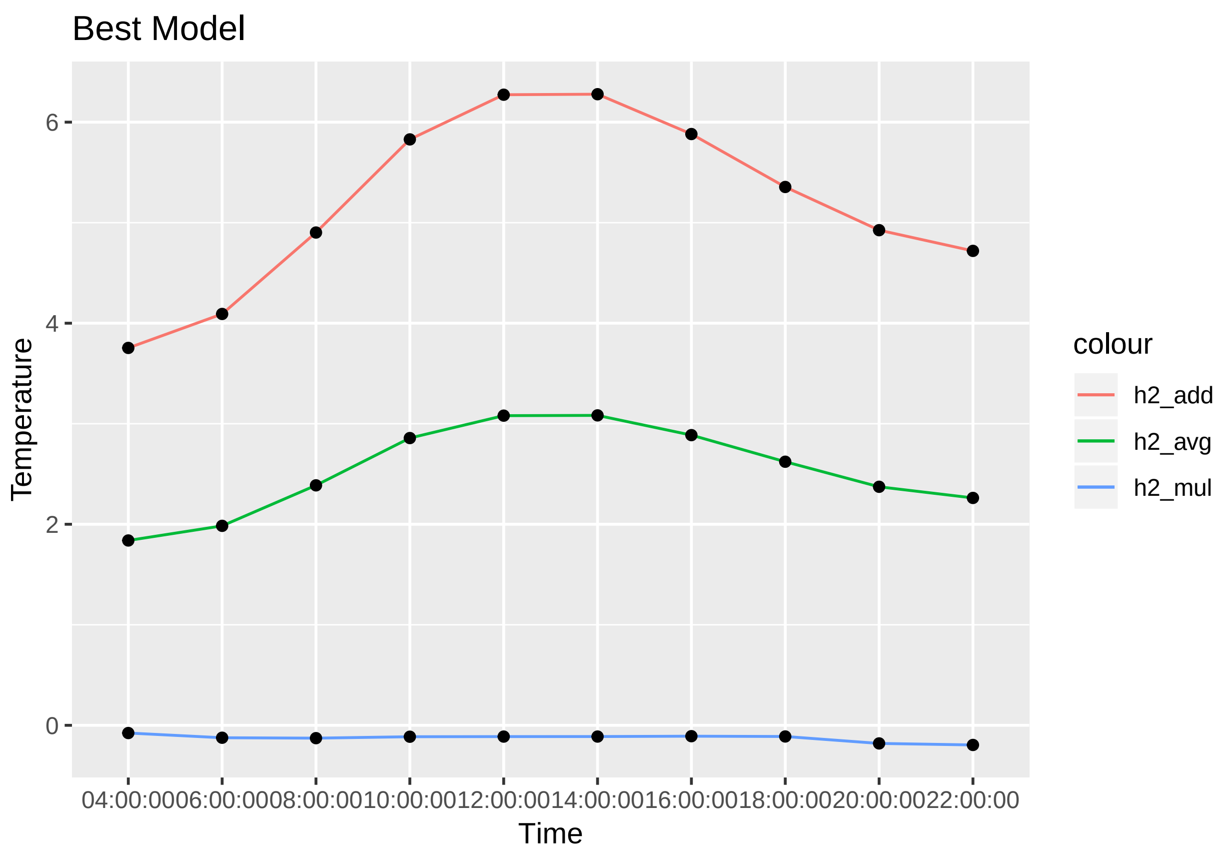
Best Selected (H2)

The final model selected uses distances = (2 hours, 5 days, 70 kms)

```r
library(ggplot2)

temps_add$avg = (temps_add$h2 + temps_mul$h2)/2

ggplot() +
  geom_line(data = temps_mul[1:10,], aes(x = time, y = h2, group=1, col="h2_mul")) +
  geom_point(data = temps_mul[1:10,], aes(x = time, y = h2)) +
  geom_line(data = temps_add[1:10,], aes(x = time, y = h2, group=1, col="h2_add")) +
  geom_point(data = temps_add[1:10,], aes(x = time, y = h2)) +
  geom_line(data = temps_add[1:10,], aes(x = time, y = avg, group=1, col="h2_avg")) +
  geom_point(data = temps_add[1:10,], aes(x = time, y = avg)) +
  labs(x = "Time", y = "Temperature", title = "Best Model")
```



Conclusion:

We thought that the kernels in the addition model are predicting independently. The date kerneal is the only kernel that can detect some seasonal trend, and making a dependent prediction would be better. We ended up multiplying the three kernels to get that model.

We found that this multiplication model is like a lower bound to the predicted temperature and the addition model is like an upperbound to the predicted temperature. On selecting an optimal distance from the plots above we ended up averaging the addition model prediction and the multiplication model prediction, and this average prediction looks like a more acurate prediction for a day in the month of january.

To conclude, we think that the average of the multiplication model and the addition model is a more acurate model for prediction of temperatre as it is not completely an independent model.

Code:

kernel_function.py

```python
#!/usr/bin/env python2
# -*- coding: utf-8 -*-

from pyspark import SparkContext
from datetime import datetime
import collections
from math import radians, cos, sin, asin, sqrt, exp, fabs
import csv

def gaussian(dist, h):
    if isinstance(dist, collections.Iterable):
        res = []
        for x in dist:
            res.append(exp(float(-(x**2))/float((2*(h**2)))))
    else:
        res = exp(float(-(dist**2))/float((2*(h**2))))
    return res

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
    return km

def timeCorr(time):
    result = []
    if hasattr(time, '__iter__'):
        for x in time:
            if x <= -12:
                result.append(24 + x)
            else:
                result.append(fabs(x))
    else:
        if time <= -12:
            result = 24 + time
        else:
            result = fabs(time)
    return result

def h_hours(time1, time2):
    hDelta = datetime.strptime(time1, '%H:%M:%S') - datetime.strptime(time2, '%H:%M:%S')
    tDiff = hDelta.total_seconds()/3600
    tCorr = timeCorr(tDiff)
    return tCorr
```

```
def h_days(day1, day2):
    dDelta = datetime.strptime(day1, '%Y-%m-%d') - datetime.strptime(day2, '%Y-%m-%d')
    return dDelta.days

def mergeVal(x):
    sVals = list(stations_bc.value[x[0]])
    vals = list(x[1])
    vals.extend(sVals)
    return (x[0],tuple(vals))


def kernelFunc(pred, data, dist):
    result = list()

    date = pred["date"]
    lat = pred["lat"]
    lon = pred["lon"]
    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']

    data = data.filter(lambda x: datetime.strptime(x[1][0], '%Y-%m-%d') < datetime.strptime(date, '%Y-%

    for time in times:
        temp = data.map(lambda x: (x[1][2],( h_hours(time, x[1][1]),
                                             h_days(date, x[1][0]),
                                             haversine(lon1=lon,
                                                       lat1=lat, l
                                                       on2=x[1][4],
                                                       lat2=x[1][3])))) \
            .map(lambda (temp, (distTime, distDays, distKM)): (temp,(gaussian(distTime, h=dist[0]),
                                                                      gaussian(distDays, h=dist[1]),
                                                                      gaussian(distKM, h=dist[2])))) \
            .map(lambda (temp, (ker1, ker2, ker3)): (temp,
                                                     ker1 + ker2 + ker3,
                                                     float(ker1) * float(ker2) * float(ker3))) \
            .map(lambda (temp, kerSum, kerProd): (temp,
                                                  (kerSum,
                                                   temp*kerSum,
                                                   kerProd,
                                                   temp*kerProd))) \
            .map(lambda (temp, (kerSum, tkSum, kerProd, tkProd)): (None,
                                                                   (kerSum, tkSum,
                                                                    kerProd, tkProd))) \
            .reduceByKey(lambda (kerSum1, tkSum1, kerProd1, tkProd1),
                         (kerSum2, tkSum2, kerProd2, tkProd2): \
                                    (kerSum1+kerSum2,
                                     tkSum1+tkSum2,
                                     kerProd1+kerProd2,
                                     tkProd1+tkProd2)) \
            .map(lambda (key,(sumKer, sumTk, prodKer, prodTk)): (float(sumTk)/float(sumKer),
                                                                  float(prodKer)/float(prodTk)))
        result.append([time, temp.collect()[0]])
    return result
```

```
sc = SparkContext(appName = "BDA3_Spark_Kernel_Job")

# Station, lat, long
stations = sc.textFile("data/stations.csv").map(lambda line: line.split(";")) \
                .map(lambda obs: (obs[0], (float(obs[3]), float(obs[4])))).collect()

stations_dict = {}
for s in stations:
    stations_dict[s[0]] = s[1]

#Broadcast stations_dict
stations_bc = sc.broadcast(stations_dict)

# (station, (date, time, temp))
temperatures = sc.textFile("data/temperature-readings.csv") \
                    .sample(False, .001, 12345).map(lambda line: line.split(";")) \
                    .map(lambda l: (l[0], (str(l[1]), str(l[2]), float(l[3]))))


# Test the kernelFunc
# (station, (date, time, temp, lat, long))
train = temperatures.map(lambda l: mergeVal(l))

pred = {}
pred["date"] = '2013-01-25'
pred["lat"] = 58.4274
pred["lon"] = 14.826

results = {'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': [[], []]}

dists = [(2, 5, 50), (2, 5, 70), (2, 7, 100), (3, 8, 100), (4, 9, 150), (4, 10, 200)]

for dist in dists:
    predictions = kernelFunc(pred, train, dist)
    for p in predictions:
        results[p[0]][0].append(p[1][0])
        results[p[0]][1].append(p[1][1])

print(results)

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']

with open('outputs/add.csv', 'wb') as out:
    csv_out=csv.writer(out)
    csv_out.writerow(['time', 'h1', 'h2', 'h3', 'h4', 'h5', 'h6'])
    for time in times:
        row = [time] + results[time][0]
```

```
        csv_out.writerow(row)

with open('outputs/mul.csv', 'wb') as out:
    csv_out=csv.writer(out)
    csv_out.writerow(['time', 'h1', 'h2', 'h3', 'h4', 'h5', 'h6'])
    for time in times:
        row = [time] + results[time][1]
        csv_out.writerow(row)

# predictions_rdd = sc.parallelize(predictions).repartition(1)
# print(predictions_rdd.collect())
```