Big Data Analytics - BDA3 Report

Mim Kemal Tekin (mimte666) & Andreas Stasinakis (andst745)

5/18/2019

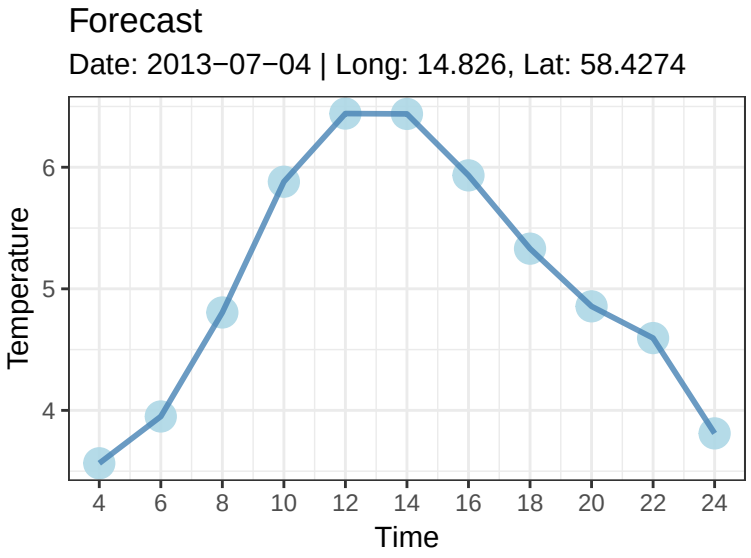
Contents

Main Task: Forecast via Sum of Guassian Kernels

In this assignment we implement a forecast predictor via kernel methods in Spark (PySpark). The Prediction is made for a close area to Linkoping and on 2013-07-04 for 4am-6am-...-12am(00:00). We can see the results below. Results are listed in order from 4am to 12 am.

```
3.56483243078
3.9499499458
4.80536592456
5.88101315891
6.44178931489
6.43875106854
5.93315743423
5.33036195768
4.85586461241
4.59473322143
3.81087715334
```

And we can plot this result as following in order to see visually:



The python script that we use to implement kernel methods is below. We have functions to calculate the distances between locations, dates and times.

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

# function calculates the great circle distance between two points as km
def haversine(lon1, lat1, lon2, lat2):
    # convert decimal degrees to radians
    lon1, lat1, lon2, lat2 = \
      map(radians, [float(lon1), float(lat1), float(lon2), float(lat2)])
    dlon = lon2 - lon1 # haversine formula
    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

# function calculates the day difference as day count
def dates_diff(d1, d2):
    date_format = "%Y-%m-%d"
    d1 = datetime.strptime(d1, date_format)
    d2 = datetime.strptime(d2, date_format)
    delta = d1 - d2
    return abs(delta.days)

# function calculates the time difference as hour
def time_diff(t1, t2):
    time_format = "%H:%M:%S"
    t1 = datetime.strptime(t1, time_format)
    t2 = datetime.strptime(t2, time_format)
    delta = t1 - t2
    return abs(delta.total_seconds()/3600)

# function calculates the gaussian kernel
def gaussian_kernel(h,dist):
    var = 2*(h**2)
    dist = dist**2
    kernel = exp(-dist/var)
    return kernel


### CREATE THE SPARK CONTEXT
sc = SparkContext(appName="lab_kernel")

### IMPORT THE DATA
temps = sc.textFile("/user/x_mimte/data/temperature-readings.csv")
stations = sc.textFile("/user/x_mimte/data//stations.csv")

### fix/map the data for semicolons
stations = stations. \
    map(lambda line: line.split(";"))
```

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

### MAP THE DATA
# map it as
# (station_no, (lat, long))
stations = stations.map(lambda x: (x[0],(x[3], x[4])))

# map it as
# (station_no, (date, time, temp))
temps = temps.map(lambda x: (x[0], (x[1], x[2], float(x[3]))))

### CREATE A BROADCAST TO GET VALUES BY KEY
station_loc = stations.collectAsMap()
bc = sc.broadcast(station_loc)

# map the data (we joined stations with location info)
# (station_no, (date, time, temp), (lat, long))
rdd = temps.map(lambda x: (x[0], x[1], bc.value.get(x[0])))

### SET THE INPUT DAY
lat = 58.4274 # Up to you
long = 14.826 # Up to you
date = "2013-07-04" # Up to you
times = tuple(["{:02d}:00:00".format(i%24) for i in range(4,26,2)])
# lat1, long1 = ('60.1538', '13.8019')

### SETTINGS FOR KERNEL
h_distance = 50
h_date = 30
h_time = 2

# filter the rdd to get only the insterested day
rdd = rdd.filter(lambda x: x[1][0]<date)

### Calculate the constant kernels for the day.
# We do this in order to gain a small performance when we calculate kernels for
# different times on the day
dist_data_kernels = rdd.\
    map(lambda x: (\
        gaussian_kernel(h_date, dates_diff(date, x[1][0])) +\
        gaussian_kernel(h_distance, haversine(long, lat, x[2][1], x[2][0])),\
        x[1][1],\
        float(x[1][2])))

# we keep the readings of this data in the cache
dist_data_kernels = dist_data_kernels.cache()

### LOOP FOR CALCULATE KERNELS AND MAKE PREDICTIONS
predictions = []
for time in times:
    # calculate gaussian kernel for time differences and obtain the sum of kernel
    # (1st map) map the final kernel as (final_kernel, temperature)
```

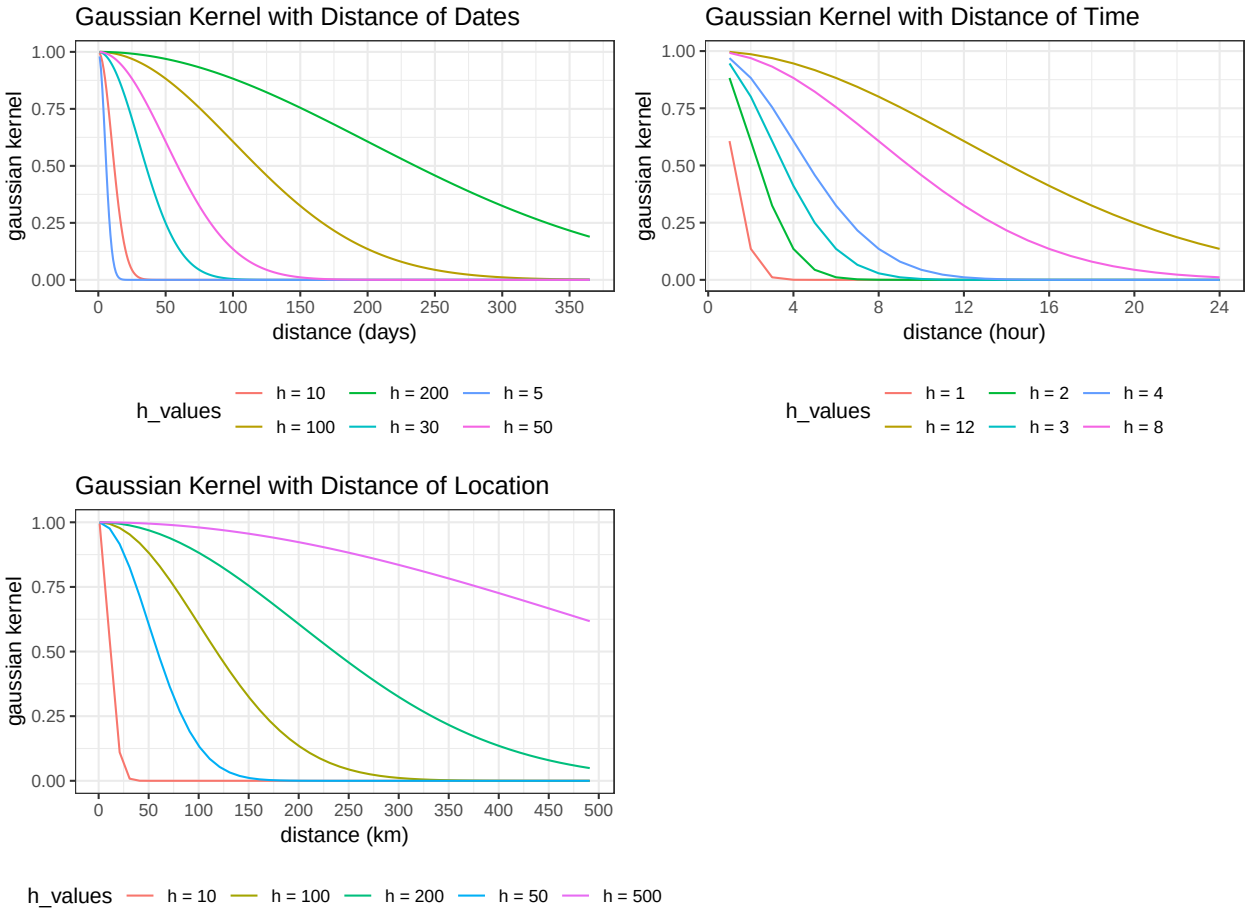
```
    # (2nd map) map them as (final_kernel*temperature, final_kernel)
    # reduce the columns by summing them. This results will give us
    # numerator and denominator.
    num, den = dist_data_kernels. \
        map(lambda x: (x[0] + gaussian_kernel(h_time, time_diff(time, x[1])), x[2])). \
        map(lambda x: (x[0]*x[1], x[0])). \
        reduce(lambda x,y: (x[0]+y[0], x[1]+y[1]))
    # divide them and get the prediction for that time,day,location.
    # store them in the array
    predictions.append(num/den)

# parallelize the array in order to create RDD from it and save the results
# to HDFS directory as text file
forecast = sc.parallelize(predictions)
forecast.saveAsTextFile("bda3_res/forecast")
```

Questions

Question 1

We select the optimal h for the kernels by ploting the possible differences and the effect of h to the result. We are interested with distances of km, date and time. When we think it we need to give the importance to close observations when we make predictions. For instance if we are predicting for "02:00:00", we do not want to affect this prediction with temperature of "14:00:00". Because different times on the day are supposed to be different, we should consider day time and night time. Same thing is valid for locations, we do not want to count in the observations in the north when we make a prediction for south. Also date is important, every year some events affect climate and even the same dates are not similar each other.

In order to see the affect of h parameter to this distances we create some x-axis grids that are sensible for the specific distances. We can see the specific plots below:



All of the plots show that when we select a small h value, gaussian kernel reaches 0 so much faster. This means we care about closer observations only. If we talk about this on Location plot, we can say that when h=10, we eliminate the effect of the observations which are located more than ~20,25 km. Same logic can apply for other plots. In the light of this, we can say when we increase h, we decrease the sensitivity of that specific distance. In the forecast domain, it is better to care about mostly closer observations. Let us discuss about our h settings. We selected:

- h_location = 50
- h_dates = 30
- h_time = 2

h for location that we select, mostly counts the observations captured between 0-100km. But of course as we can see from the plot, gaussian kernel provides us that set lower weights to these further observations. h for dates, we select quite big h, because we think that it is okay to make a prediction by looking the observations in a season distance which is aroun 0-60 days. We can see h=30 gives the importance to max 100 days. For the dates, we select a small h. If we select h=1 instead h=2, it would be too strict again. It will just count only 3 hours further observations. With h=2 we capture a good range between 0-6 hours which is a good period duration on a day if we consider a day as morning, afternoon, evening and night.
We did not do any further accuracy performance test while we select h parameter. We tried to select some sensible values by considering the importance of far observations.

Question 2

We can observe the results are not that different each other and it does not have a good accuracy. During July we expect much higher temperatures than 4-6 degrees. This causes from how we obtain the final kernel from the gaussian kernels. We get the sum of three different reasonable kernel, but there is a problem with this. If we think an observation which is located at 300 km more north and if somehow other kernels (date and time) have a short distance from the point that we want to predict, we will have 0 for the location kernel but other kernels will have a value close to 1. In the final kernel calculation phase, we get sum of them and we still use this observation to get information, despite the fact that it is too north than our location, and this causes lower temperatures after prediction.
To prevent this problem, we can simply multiply each kernel to obtain final kernel. In this case we will eliminate this observation when it is not close enough to get information from it. By this way we will not have the problem that we discussed before.

Appendix: Plot Scripts

```
##########################################################################
######################### Plot for Forecast ##############################
##########################################################################

forecast = c(3.5648, 3.9499, 4.8053, 5.8810, 6.4417, 6.4387, 5.9331, 5.3303,
             4.8558, 4.5947, 3.8108)
df_plot = data.frame(time = 2:12*2,
                     temp = forecast)

plot = ggplot(df_plot, aes(x = time, y = temp)) +
  geom_point(color="lightblue", alpha=0.9, size=5) +
  geom_line(color="steelblue", alpha=0.8, size=1) +
  labs(title="Forecast",
       subtitle = "Date: 2013-07-04 | Long: 14.826, Lat: 58.4274",
       x = "Time", y = "Temperature") +
  scale_x_continuous(breaks = 2:12*2) +
  theme_bw()
plot
##########################################################################
######################### Kernel Plot Script #############################
##########################################################################
library(ggplot2)
library(gridExtra)

gaussian_kernel = function(norm, weight){
  return(exp(-norm^2/(2*weight^2)))
}

###################### Different values for locations ################

x_distance = seq(1,500,10)
h_dist = c(10,50,100,200,500)

df_distance = data.frame(x = x_distance)
for(h in h_dist){
  df_distance[[paste("y",h,sep = "")]] = gaussian_kernel(x_distance, h)
}

df_loc = df_distance
colnames(df_loc) = c("iteration","h = 10", "h = 50","h = 100",
                         "h = 200","h = 500")

df_loc = tidyr::gather(df_loc, key = "h_values",
                           value ="kernel_values", -iteration)

plot_loc = ggplot(df_loc) +
  geom_line(aes(x=iteration, y = kernel_values, color=h_values)) +
  labs(x = "distance (km)", y = "gaussian kernel",
       title = "Gaussian Kernel with Distance of Location") +
  scale_x_continuous(breaks = c(0, 1:10*50)) +
  theme_bw() +
```

```
  theme(legend.position = "bottom")


##################### Different values for days #################

x_days = seq(1,365,1)
h_days = c(5, 10,30,50,100, 200)

df_distance = data.frame(x = x_days)
for(h in h_days){
  df_distance[[paste("y",h,sep = "")]] = gaussian_kernel(x_days, h)
}


df_date = df_distance
colnames(df_date) = c("iteration", "h = 5", "h = 10", "h = 30","h = 50",
                          "h = 100","h = 200")

df_date = tidyr::gather(df_date, key = "h_values",
                            value ="kernel_values", -iteration)

plot_date = ggplot(df_date) +
  geom_line(aes(x=iteration, y = kernel_values, color=h_values)) +
  labs(x = "distance (days)", y = "gaussian kernel",
       title = "Gaussian Kernel with Distance of Dates") +
  scale_x_continuous(breaks = c(0, 1:7*50)) +
  theme_bw() +
  theme(legend.position = "bottom")


##################### Different values for time(hour) #################

x_time = seq(1,24,1)
h_time = c(1,2,3,4,8,12)

df_distance = data.frame(x = x_time)
for(h in h_time){
  df_distance[[paste("y",h,sep = "")]] = gaussian_kernel(x_time, h)
}


df_time = df_distance
colnames(df_time) = c("iteration","h = 1", "h = 2", "h = 3", "h = 4","h = 8",
                          "h = 12")

df_time = tidyr::gather(df_time, key = "h_values",
                            value ="kernel_values", -iteration)

plot_time = ggplot(df_time) +
  geom_line(aes(x=iteration, y = kernel_values, color=h_values)) +
  labs(x = "distance (hour)", y = "gaussian kernel",
       title = "Gaussian Kernel with Distance of Time") +
  scale_x_continuous(breaks = c(0, 1:8*4)) +
```

```
  theme_bw() +
  theme(legend.position = "bottom")


grid.arrange(grobs = list(plot_date, plot_time, plot_loc), ncol=2)
```