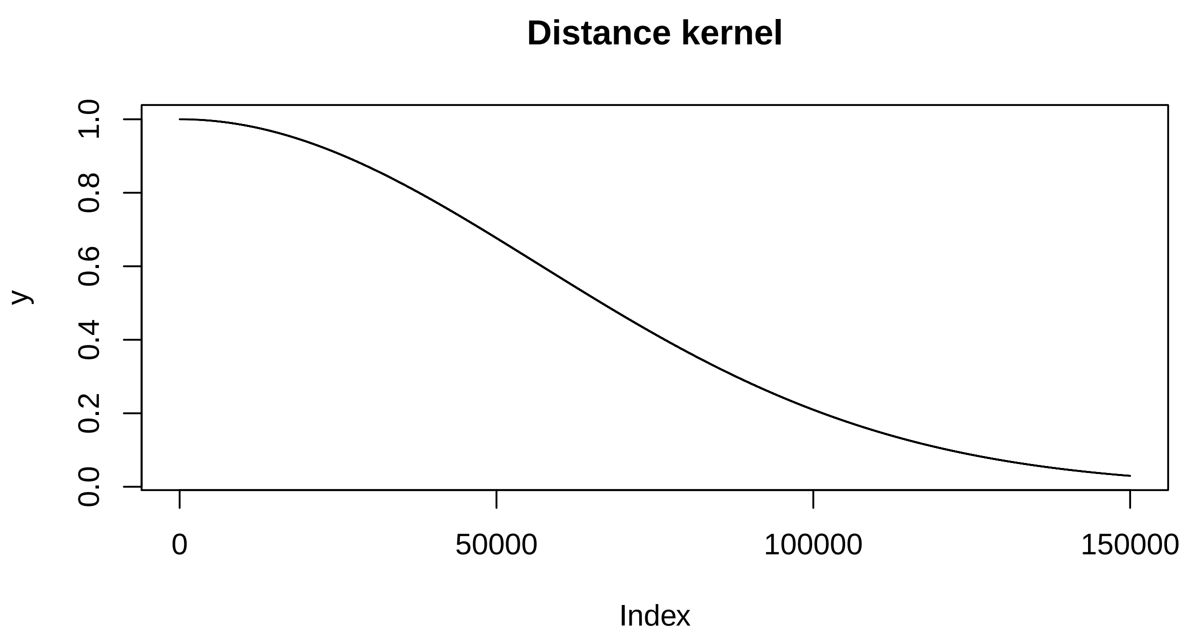
lab3_kernel_2

Zhixuan_Duan(zhidu838) and zahra jalilpour(zahja096)

```
# load package
library(ggplot2)
```

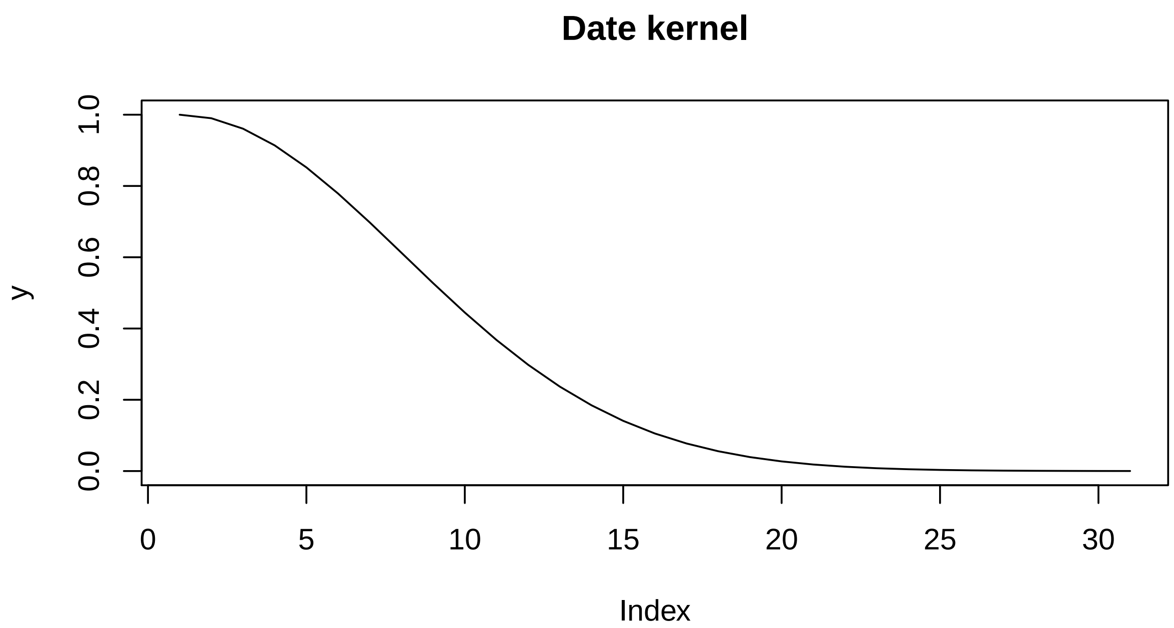
Test h_distance, h_date and h_time.

```
h_distance <- 80000
h_date <- 10
h_time <- 6

dist = seq(0,150000, 1)
y = exp(-(dist/h_distance)^2)
plot(y, type="l", main = "Distance kernel")
```

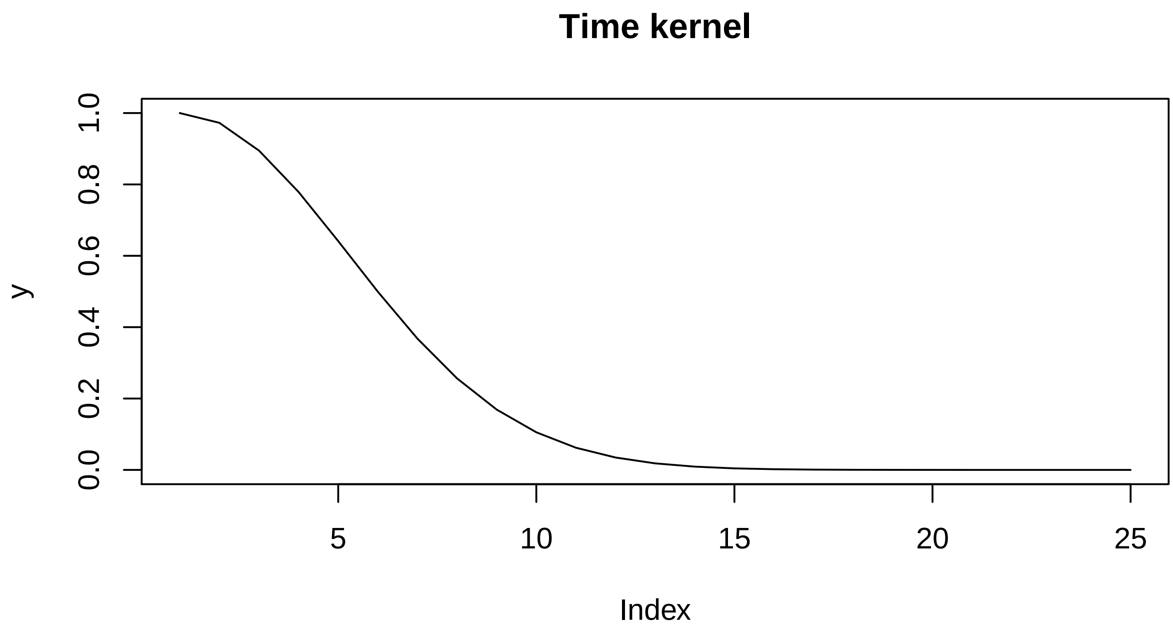



```
# the haversine function outputs kms, so we define h_distance = 80

dat = seq(0,30, 1)
```

```
y = exp(-(dat/h_date)^2)
plot(y, type="l", main = "Date kernel")
```




```
# h_date = 10

tim = seq(0, 24, 1)
y = exp(-(tim/h_time)^2)
plot(y, type="l", main = "Time kernel")
```




```
# h_time = 6
```

pyspark code

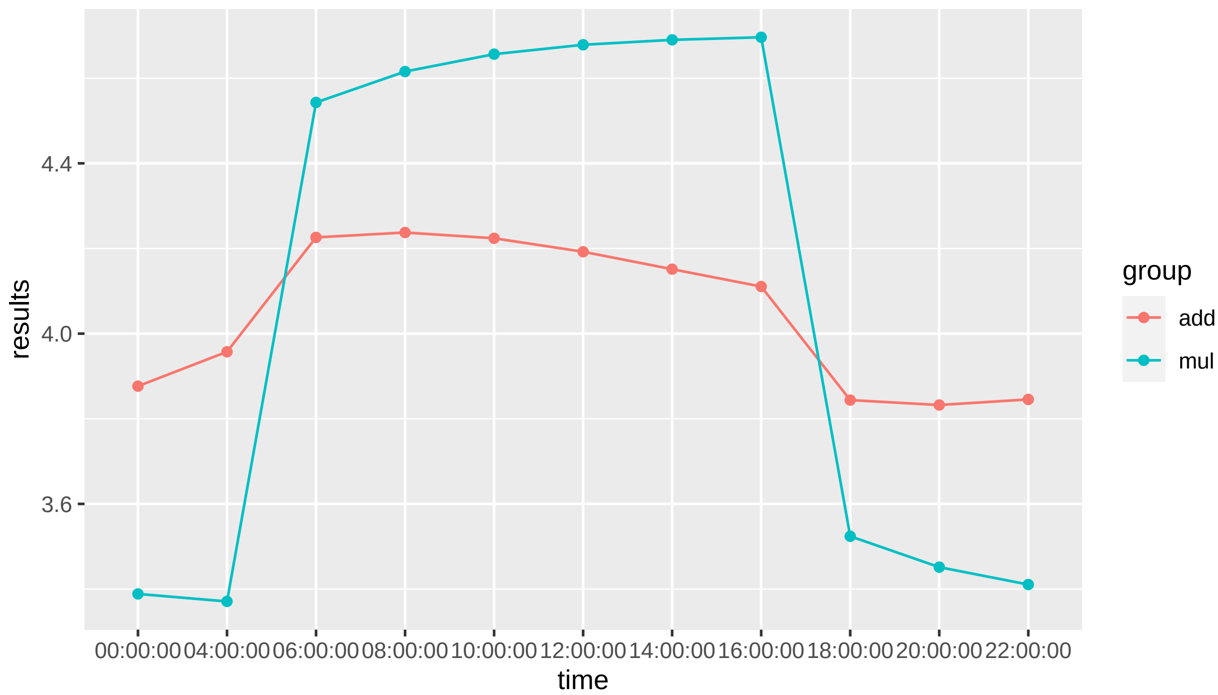
```
from __future__ import division
from math import radians, cos, sin, asin, sqrt, exp
from datetime import datetime
from pyspark import SparkContext

sc = SparkContext(appName="lab_kernel_2")

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 = 80
h_date = 10
```

```python
h_time = 6
a = 58.40 # latitude
b = 15.62 # longitude
date = "2013-11-04"


## date
def d_date(date1, date2):
    date1 = datetime.strptime(date1, "%Y-%m-%d")
    date2 = datetime.strptime(date2, "%Y-%m-%d")
    output = abs((date2 - date1).days)
    return output

## time
def d_time(time1,time2):
    time1 = datetime.strptime(time1, '%H:%M:%S')
    time2 = datetime.strptime(time2, '%H:%M:%S')
    output = abs((time1-time2))
    return output

def add_kernels(d1, d2, d3):
    k1 = exp(- d1**2 / (2*(h_distance**2)))
    k2 = exp(- d2**2 / (2*(h_date**2)))
    k3 = exp(- d3**2 / (2*(h_time**2)))
    output = k1 + k2 + k3
    return output

def mul_kernels(d1, d2, d3):
    k1 = exp(- d1**2 / (2*(h_distance**2)))
    k2 = exp(- d2**2 / (2*(h_date**2)))
    k3 = exp(- d3**2 / (2*(h_time**2)))
    output = k1 * k2 * k3
    return output


stations = sc.textFile("/Users/darin/Desktop/stations.csv")
temp = sc.textFile("/Users/darin/Desktop/temperature-readings-small.csv")

###########
stations = stations.map(lambda x: x.split(";"))
stations = stations.map(lambda x: (x[0],haversine(b,a,float(x[4]),float(x[3]))))

m=sc.parallelize(stations.collect()).collectAsMap()
stations=sc.broadcast(m)


###########
temp = temp.map(lambda x: x.split(";"))
temp = temp.map(lambda x: (stations.value[str(x[0])],x[1],x[2],float(x[3])))

filter_temp = temp.filter(lambda x: x[1] <= date)

filter_temp = filter_temp.map(lambda x: (x[0], d_date(date,x[1]), d_time(time,x[2]), x[3]))
```

```
##########
predicted_add=[]
predicted_mul=[]


for time in ["00:00:00", "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"]:
    add_allkernels = filter_temp.map(lambda x:(add_kernels(x[0],x[1],x[2]),x[3])).map(lambda x:(x[0],x[
    predicted_add.append(add_allkernels[1] / add_allkernels[0])

    mul_allkernels = filter_temp.map(lambda x:(mul_kernels(x[0],x[1],x[2]),x[3])).map(lambda x:(x[0],x[
    predicted_mul.append(mul_allkernels[1] / mul_allkernels[0])

print(predicted_add)
print('\n')
print(predicted_mul)
sc.parallelize(predicted_add).saveAsTextFile("add_kernel_predictions")
sc.parallelize(predicted_mul).saveAsTextFile("mul_kernel_predictions")


sc.stop()
```

Prediction

```
add = c(3.876649015227585, 3.845461804411638, 3.832316480947907, 3.8439126628440006, 4.110669123798261,

mul = c(3.3885900519940253, 3.410613572232551, 3.45145724020793, 3.5240266060201395, 4.696178708661572,

time = c("00:00:00", "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")

add_df = data.frame(results = add, time = time, group = 'add')

mul_df = data.frame(results = mul, time = time, group = 'mul')

final = rbind(add_df, mul_df)

ggplot(final, aes(x = time, y = results, group = group)) +
  geom_line(aes(color = group)) +
  geom_point(aes(color = group))
```



Analysis

The predicted temperature is not much different in the additive model, but the multiplicative model shows a clear upward and downward trend.

We think this is because when the three kernels are all very small and close to 0, the multiplication model will be smaller(approaching 0), and the addition model will be relatively large. And when the three kernels are large, the multiplication model will have a greater impact on the predicted value compared to additive model.