Erik Nordström erino349
Cajsa Schöld cajsc235

TDDE31 Lab Exercise 3: Machine Learning

Results:

Results predicting the temperature for 2013-07-04 lon: 58.4274 14.826 using the additive kernel:
[(24, 9.311915042394283), (22, 8.985519087733714), (20, 8.651774872968065), (18, 9.476836490632897), (16, 8.89755746590985), (14, 8.31581338544655), (12, 7.892527698870256), (10, 6.448596082355736), (8, 4.729686944477248), (6, 4.2051858663890505), (4, 2.8034940997291113)]

Results predicting the temperature for 2013-07-04 lon: 58.4274 14.826 using the multiplication kernel:
[(24, 13.7476287630208), (22, 15.534271875562615), (20, 17.704482780072652), (18, 19.00869220203396), (16, 19.337561936825885), (14, 19.203513458281712), (12, 18.242601721497163), (10, 16.837490225276994), (8, 14.746011214650395), (6, 12.777741581621438), (4, 11.401412494322047)]

Questions:

Show that your choice for the kernels' width is sensible, i.e. it gives more weight to closer points. Discuss why your definition of closeness is reasonable.

When determining the kernels' widths we plotted the kernel function and experimented with different values until we found widths that assigned, in our opinion, appropriate weights for the different data points, regarding the physical, date and time distance to the data point we aimed to predict. We decided to go with the following values:

```
h_distance = 150 # km
h_date = 7 # days
h_time = 150 # min
```

Repeat the exercise using a kernel that is the product of the three Gaussian kernels above. Compare the results with those obtained for the additive kernel. If they differ, explain why.
When using the additive kernel all of the three gaussian kernels contribute to the combined kernel by summing its values. This results in that when predicting the weather in July for a certain location, data from December for the same location can still significantly influence the final prediction, even though the temperature is significantly different in December compared to July for the same place.

Erik Nordström erino349
Cajsa Schöld cajsc235

When instead using a kernel that is the product of the three gaussian kernels it will address this problem, resulting in that a data point in december for a certain location will not influence the prediction for the temperature in July as much, even though they are from the same location. This will for most cases result in better predictions.

Code:

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

sc = SparkContext(appName="lab_kernel")
stations = sc.textFile("BDA/input/stations.csv")
temps = sc.textFile("BDA/input/temperature-readings.csv")

date = "2013-07-04"
h_distance = 150
h_date = 7
h_time = 150
lon = 58.4274
lat = 14.826

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"]

days_in_month = 31
minutes_per_hour = 60

def haversine(lon1, lat1, lon2, lat2):
   """
   Calculate the great circle distance between two points
   on the earth (specified in decimal degrees)
   """
   lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
   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

# kernel for physical distance
def compute_dist_kern(c_lon, c_lat):
    eukl_dist_km = haversine(lon, lat, c_lon, c_lat)
    kernel = exp(-(eukl_dist_km/h_distance)**2)
    return kernel

# kernel for day distance
def compute_day_kern(c_month, c_day):
    month = int(date[5:7])
    day = int(date[8:10])
    elapsed_days_in_year = month * days_in_month + day
    c_elapsed_days_in_year = c_month * days_in_month + c_day
    day_delta = abs(elapsed_days_in_year - c_elapsed_days_in_year)
    kernel = exp(-(day_delta/h_date)**2)
    return kernel

# kernel for minute distance
def compute_time_kern(time, c_time):
    hour, c_hour = int(time[0:2]), int(c_time[0:2])
    minute, c_minute = int(time[3:5]), int(c_time[3:5])
    elapsed_minutes_in_day = hour * minutes_per_hour + minute
    c_elapsed_minutes_in_day = c_hour * minutes_per_hour + c_minute
    minute_delta = abs(elapsed_minutes_in_day - c_elapsed_minutes_in_day)
    kernel = exp(-(minute_delta/h_time)**2)
    return kernel

# create and broadcast a RDD containing station data to all nodes
station_lines = stations.map(lambda line: line.split(";"))
# (station, (lon, lat))
subset_station_lines = station_lines.map(lambda x: (int(x[0]),
(float(x[3]), float(x[4]))))
station_data = subset_station_lines.collectAsMap()
b_station_data = sc.broadcast(station_data)

temp_lines = temps.map(lambda line: line.split(";"))
# ((year, month, day, time), (temp, lon + lat))
```

Erik Nordström erino349
Cajsa Schöld cajsc235

```
subset_temp_lines = temp_lines.map(lambda x: ((int(x[1][0:4]),
int(x[1][5:7]), int(x[1][8:10]), x[2]),
                                              (float(x[3]),
b_station_data.value.get(int(x[0])))))
# Filter out posterior years, month and days
f_temp_lines = subset_temp_lines.filter(lambda x: x[0][0] <=
int(date[0:4]) and x[0][1] <= int(date[5:7]) and x[0][2] <
int(date[8:10]))

# cache the temperature data so that it does not have to be read again
f_temp_lines.cache()

sum_kernel_predictions = []
mult_kernel_predictions = []

for timestamp in times:

   # Isolate the hour from the timestamp
   timestamp_hour = int(timestamp[0:2])

   # Filter out posterior hours
   filtered_temp_lines = f_temp_lines.filter(lambda x: int(x[0][3][0:2]) <
timestamp_hour)

   #((year, month, day, time), ((distance_kernel, day_kernel,
hour_kernel), temp))
   combined_kernel = filtered_temp_lines.map(lambda x: (x[0],
((compute_dist_kern(x[1][1][0], x[1][1][1]), compute_day_kern(x[0][1],
x[0][2]), compute_time_kern(timestamp, x[0][3])), x[1][0])))

   # (1, (sum(u)*y, sum(u), mult(u)*y, mult(u)))
   kernels = combined_kernel.map(lambda x: (1, ((x[1][0][0] + x[1][0][1] +
x[1][0][2]) * x[1][1], x[1][0][0] + x[1][0][1] + x[1][0][2], (x[1][0][0] *
x[1][0][1] * x[1][0][2]) * x[1][1], x[1][0][0] * x[1][0][1] *
x[1][0][2])))

   # aggregating the values over all the training points
   kernels = kernels.reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1],
x[2] + y[2], x[3] + y[3]))
```

Erik Nordström erino349
Cajsa Schöld cajsc235

```
    # compute y(x), (agg_sum(u)*y / agg_sum(u), agg_mult(u)*y /
agg_mult(u))
    kernel_values = kernels.mapValues(lambda x: (x[0]/x[1], x[2]/x[3]))

    kernel_result = kernel_values.collectAsMap().get(1)

    sum_kernel = kernel_result[0]
    mult_kernel = kernel_result[1]
    sum_kernel_predictions.append((timestamp_hour, sum_kernel))
    mult_kernel_predictions.append((timestamp_hour, mult_kernel))

print(sum_kernel_predictions)
print(mult_kernel_predictions)
```