Simon Jakobsson Simja649, Daniel Kouznetsov danko376

Lab report 3 - Machine Learning with Spark

In this lab we use Python and Spark to build a machine learning model to predict the temperature for a few time points for a given date. The model for the first task is made by combining three Gaussian kernels, the combining is done with a summation. Then we repeat this process but we use the product of the Gaussian kernels to predict the temperature, and finally we use two built-in models from MLlib to make predictions. The chosen MLlib algorithms were Random forest and gradient boosted trees. See appendix for data and code.

Predicted data point
Date: 2023-08-12
Location: (58.4274, 14.826), Vadstena, Östergötland

Kernel functions

The kernel functions written measured several distances, these were distance between stations (in km), date and time. The functions were used to calculate similarities and closeness for the different data points for the model to later make predictions on our new data point. Every kernel function is sent through a gaussian function, this means that when the distance is low we are going to be closer to 1 in our output for the kernel function. This means that no weights are going to get larger than 1 which could be an issue since what is measured varies a lot, for example distance in time has a maximum value of 12, distance in km can be >1000.

Distance kernel measures distance between the stations, since we have latitude and longitude for each station and for our own predicted point we also have coordinates. We make use of the haversine formula to get the correct distance between these points. Since Sweden is a big and narrow country the distances here might get very large >1000 km. With this in mind we set our smoothing coefficient to 100 to give more importance to stations that are in the vicinity of the point we want to predict. With the gaussian function in mind when choosing the smoothing coefficients, we can make sure that stations that are more than 250 km away from the predicted point are going to affect the prediction less.

Date kernel measures how far in between the predicted date is from another given date. Since a full year is 365 days and we have several years, we use modulo 365 on the initial distance since 4 years and 70 days distance should equal 0 years on 70 days distance in the end. There is one more thing to keep in mind, 364 days is also one day in distance because of the cyclic nature of dates, therefore we have to check if the initial distance is greater than 365/2 and if it is, we just calculate it with the cyclic nature in mind, i.e., distance = 365 - distance. The smoothing coefficient was set to 15, therefore days within 30 days of the predicted date are going to have an effect on our predicted value.

Hour kernel measures the time between the predicted data point from another data point. This distance is measured in hours. The same principle as for the date kernel is used here, 23 hours forward is also just one hour in absolute time but backwards. Therefore, we use the

same check here, if the initial distance in hours is greater than 12 we do distance = 24 - distance. The smoothing coefficient was set to 2 to make sure that only data points with times that are +/- 4 hours from the predicted time are going to make a difference for our predicted value.

Comparing kernels

The result from **summation kernel** taken hour and temperature:
(04, 3.71)
(06, 4.15)
(08, 5.56)
(10, 7.08)
(12, 7.88)
(14, 8.72)
(16, 8.90)
(18, 7.69)
(20, 8.27)
(22, 7.78)
(00, 3.17)

The result from **product kernel** taken hour and temperature:
(04, 11.92)
(06, 12.83)
(10, 16.72)
(08, 14.23)
(12, 18.01)
(14, 18.49)
(16, 18.48)
(18, 16.32)
(20, 15.42)
(22, 13.59)
(00, 12.43)

When comparing the kernel models to each other the product kernel seems to have way better predictions. With our predicted date in mind, in the middle of summer we expect to have temperatures around 18°C in the middle of the day which we see that we have for the product kernel along with reasonable temperatures at around 12°C in the middle of the night. For the summation kernel our temperatures are a bit cold for all time points, but relative to time it is colder at night which is good, but overall a bad result for this kernel.

Using MLlib library models to make predictions

From pyspark MLlib we import LabeledPoint, RandomForest and GradientBoostedTree.
We first reformat our data, providing temperature as key with values of date, time and lon lat. We convert keys and values with LabeledPoint so that the ML libraries can use this data. Then we train both of the ML algorithms on the data, this is also a reason why we choose two tree structured algorithms so no formatting of data between libraries had to be done. We

set the max depth of both ML libraries as the same to have some fairness. In the time loop we use the data provided by the date we want to predict, time and the longitude and latitude. Then appending the predicted output to a dictionary with the key of time.

The result from **random forest** taken hour and temperature:
(04, 11.92)
(06,11.85)
(08,17.21)
(10,17.96)
(12, 17.96)
(14,17.96)
(16,17.96)
(18, 12.11)
(20, 13.10)
(22, 12.78)
(00, 11.39)

The result from **Gradient boosted trees** taken hour and temperature:
(04, 12.14)
(06, 13.09)
(08, 16.32)
(10, 18.35)
(12, 19.18)
(14, 19.18)
(16, 18.91)
(18, 17.17)
(20, 15.27)
(22, 14.72)
(00, 12.24)

We observe that the ML libraries are mostly aligned with the product kernel. Providing about a margin of error with 2+- degrees. However, we do observe that both Gradient boosted trees and random forest provide the same value for some hours when the temperature is higher. The probable reason behind this is that we have a very limited decision tree that with very similar temperatures will come to the same conclusion, for example looking at hours 12-16. You could expand the depth and perhaps increase the amount of trees to get better predictions.
And with Gradient boosted trees with each iteration it should append the last tree and make it “better”. Which is observed in the results because it does look better than the random tree result. The depth and iteration could be expanded to get better results.

The main giveaway is that using ML methods should give more nuanced but very alike results as the product kernel, as we can see on the results. However, doing more work without overfitting could be beneficial to further expand the prediction of the temperature. We also observe that the summation kernel by its nature is very skewed and should not take into consideration predicting what temperature it should be given the day and time.

Simon Jakobsson Simja649, Daniel Kouznetsov danko376

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.

The choice for each kernel's width is reasonable since weather is usually the same for the distance span chosen for each smoothing coefficient. Weather is similar in a radius 200 km from a point, +/- 30 days from a given date, +/-4 hours from a given time. See more information above in the section on kernel functions.

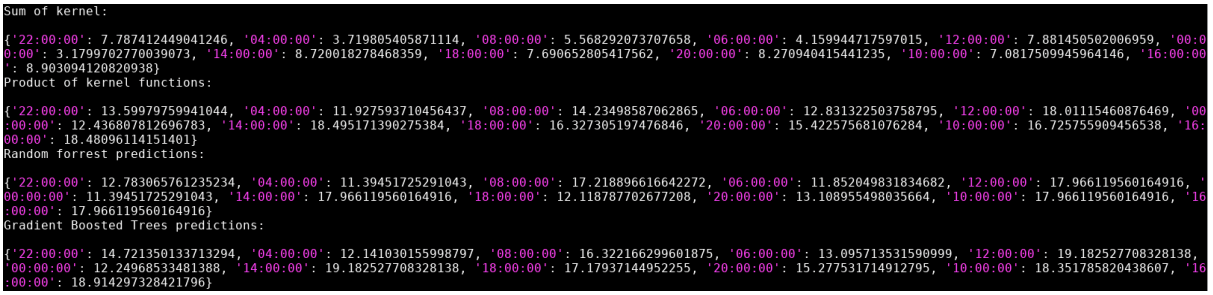
Repeat the exercise using a kernel that is the product of the three Gaussian kernels above. Compare the results with those obtained for the additive kernel. If they differ, explain why.
Yes, they do differ, with the product kernel having better results. This might be because this kernel results in a high output only when all the components "agree", i.e., all the components are near in distance because of the multiplication. If one has a high distance which should not affect the result that much, it won't use any of the other variables for that data point either, even if for example time and date are really close. This is not the case for the summation kernel, where all the individual distances matter more.

Repeat the exercise using at least two MLlib library models to predict the hourly temperatures for a date and place in Sweden. Compare the results with two Gaussian kernels. If they differ, explain why.
They do differ. The product kernel is very similar to the ML libraries and the summation is way off. See more above.

Simon Jakobsson Simja649, Daniel Kouznetsov danko376

Appendix

Result from slurm:



```
Sum of kernel:

{'22:00:00': 7.787412449041246, '04:00:00': 3.719805405871114, '08:00:00': 5.568292073707658, '06:00:00': 4.159944717597015, '12:00:00': 7.881450502006959, '00:0
0:00': 3.1799702770039073, '14:00:00': 8.720018278468359, '18:00:00': 7.690652805417562, '20:00:00': 8.270940415441235, '10:00:00': 7.0817509945964146, '16:00:00
': 8.903094120820938}
Product of kernel functions:

{'22:00:00': 13.59979759941044, '04:00:00': 11.927593710456437, '08:00:00': 14.23498587062865, '06:00:00': 12.831322503758795, '12:00:00': 18.01115460876469, '00
:00:00': 12.436807812696783, '14:00:00': 18.495171390275384, '18:00:00': 16.327305197476846, '20:00:00': 15.422575681076284, '10:00:00': 16.725755909456538, '16:
00:00': 18.48096114151401}
Random forrest predictions:

{'22:00:00': 12.783065761235234, '04:00:00': 11.39451725291043, '08:00:00': 17.218896616642272, '06:00:00': 11.852049831834682, '12:00:00': 17.966119560164916, '
00:00:00': 11.39451725291043, '14:00:00': 17.966119560164916, '18:00:00': 12.118787702677208, '20:00:00': 13.108955498035664, '10:00:00': 17.966119560164916, '16
:00:00': 17.966119560164916}
Gradient Boosted Trees predictions:

{'22:00:00': 14.721350133713294, '04:00:00': 12.141030155998797, '08:00:00': 16.322166299601875, '06:00:00': 13.095713531590999, '12:00:00': 19.182527708328138,
'00:00:00': 12.249685334813888, '14:00:00': 19.182527708328138, '18:00:00': 17.17937144952255, '20:00:00': 15.277531714912795, '10:00:00': 18.351785820438607, '16
:00:00': 18.914297328421796}
```

Code:

```python
from __future__ import division
from math import radians, cos, sin, asin, sqrt, exp
from datetime import datetime
from pyspark import SparkContext
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.tree import RandomForest, GradientBoostedTrees

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

# Smoothing coefficient, can be changed later
h_distance = 100 # Up to you
h_date = 15 # Up to you
```

```
h_time = 2 # Up to you

a = 58.4274 # Up to you
b = 14.826 # Up to you
date = "2013-08-12" # Up to you


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

stations_lines = stations.map(lambda line: line.split(";"))
temps_lines = temps.map(lambda line: line.split(";"))

# should broadcast stations because temp is 2GB
#key station: value: long and lat
stations_lon_lat = stations_lines.map(lambda x: (x[0], (float(x[3]),
float(x[4]))))
# need to flatten to array, collectAsMap return keyvalue pairs
broadcast_stations = sc.broadcast(stations_lon_lat.collectAsMap())

#smaller sample
#key station, date, time value: , temp long and lat
temp = temps_lines.sample(False,0.1).map(lambda x: ((x[0], x[1], x[2]),
(float(x[3]), broadcast_stations.value.get(x[0]))))

# Filters out dates that are posterior to our predicted date
temp = temp.filter(lambda x: datetime(int(x[0][1][0:4]),
int(x[0][1][5:7]), int(x[0][1][8:10])) < datetime(int(date[0:4]),
int(date[5:7]), int(date[8:10])))

temp.cache()


# Gaussian function
def gaussian(distance):
    return exp(-distance**2)

# Gaussian kernels

# Kernel 1: Distance from coordinates
def distance_kernel(lon1, lat1, lon2, lat2):
    return gaussian(haversine(lon1, lat1, lon2, lat2)/h_distance)

# Kernel 2: Distance measured for dates
def day_temp_kernel(date1, date2):
    #year month date
```

```
    date1 = datetime(int(date1[0:4]), int(date1[5:7]), int(date1[8:10]))
    date2 = datetime(int(date2[0:4]), int(date2[5:7]), int(date2[8:10]))
    dist = abs(date1 - date2).days % 365
    # 364 days is equal to a one-day difference, this solves that
    if (dist > 365/2):
        dist = 365 - dist
    return gaussian(dist / h_date)

# Kernel 3: Distance measured per hour
def hour_kernel(time1, time2):
    time1 = datetime(2020, 1, 1, int(time1[0:2]), int(time1[3:5]),
int(time1[6:8]))
    time2 = datetime(2020, 1, 1, int(time2[0:2]), int(time2[3:5]),
int(time2[6:8]))
    dist = abs(time1 - time2)
    hour = (dist.seconds / 3600) # Convert timedelta in seconds to hours
    # 23 hours is equal to a one-hour difference, this solves that
    if (hour > 12):
        hour = 24 - hour
    return gaussian(hour / h_time)


# ML librarires

#Make data work with ML libraries
#Key temp value: date (year, month, day), time(only first 2 digits i.e.,
hour), long lat
ml_temp = temp.map(lambda x: (x[1][0], (int(x[0][1][0:4]),
int(x[0][1][5:7]), int(x[0][1][8:10]), int(x[0][2][0:2]),
int(x[1][1][0]), int(x[1][1][1]))))
ml_train_data = ml_temp.map(lambda x: LabeledPoint(x[0], x[1]))

rf = RandomForest.trainRegressor(ml_train_data, {}, numTrees = 3,
maxDepth=8)
gbt = GradientBoostedTrees.trainRegressor(ml_train_data, {},
numIterations=10, learningRate=0.2, maxDepth=8)


prediction_date = datetime(int(date[0:4]), int(date[5:7]),
int(date[8:10]))

#Dict to all different predictions where key: time value: temp
predictions_gbt = {}
predictions_rf = {}
predictions_sum = {}
predictions_prod = {}
```

```
# Your code here
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"]:
    past_time = int(time[0:2])
    temp_filtered = temp.filter(lambda x: (int(x[0][2][0:2]) <=
past_time))
    temp_filtered.cache()

    total = temp_filtered.map(lambda x: (distance_kernel(a , b,
x[1][1][0], x[1][1][1]),
                                         day_temp_kernel(date,
x[0][1]),
                                          hour_kernel(time, x[0][2]),
x[1][0]))

    # calculate sum and product of kernels, multiply them with the
temperature, and keep track of the kernel sums separately
    kernel_sum_and_prod = total.map(lambda x: ((x[0]+x[1]+x[2]),
(x[0]*x[1]*x[2]), (x[0]+x[1]+x[2])*x[3], (x[0]*x[1]*x[2])*x[3]))


    # calculate the total sum and product of kernels and their weighted
sums and products
    reduced = kernel_sum_and_prod.reduce(lambda x1, x2: (x1[0]+x2[0],
x1[1]+x2[1], x1[2]+x2[2], x1[3]+x2[3]))

    # calculate the weighted sum and product, where the weight is the
kernel sum or product
    predictions_sum[time] = reduced[2] / reduced[0]
    predictions_prod[time] = reduced[3] / reduced[1]

    #ML part
    #from regressor put in same data format to predict temperature
    data = [prediction_date.year, prediction_date.month,
prediction_date.day, int(time[0:2]), a, b]
    predictions_rf[time] = rf.predict(data)
    predictions_gbt[time] = gbt.predict(data)

print("Sum of kernel: \n")
print(predictions_sum)

print("Product of kernel functions: \n")
print(predictions_prod)

print("Random forrest predictions: \n")
```

```
print(predictions_rf)

print("Gradient Boosted Trees predictions: \n")
print(predictions_gbt)
```