# BDA3Report

May 24, 2023

## 1 LAB EXERCISE 3: MACHINE LEARNING

Implement in Spark (PySpark) a kernel model to predict the hourly temperatures for a date and place in Sweden. To do so, you should use the files temperature-readings.csv and stations.csv from previous labs. Specifically, the forecast should consist of the predicted temperatures from 4 am to 24 pm in an interval of 2 hours for a date and place in Sweden. Use a kernel that is the sum of three Gaussian kernels:

The first to account for the distance from a station to the point of interest.

The second to account for the distance between the day a temperature measurement was made and the day of interest.

The third to account for the distance between the hour of the day a temperature measurement was made and the hour of interest.

### 1.1 Kernel Model

**Note:** For hour interval we use [0:00:00, 04:00:00, 06:00:00 …] and not 24:00:00, this is because in temperature-reading.csv, temperature is recorded from 00:00:00, so we keep that for consistancy.

[ ]:
```
from __future__ import division
from math import radians, cos, sin, asin, sqrt, exp
from datetime import datetime
from pyspark import SparkContext
from pyspark.broadcast import Broadcast
import numpy as np

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 day_difference(day1, day2):
```

```
    diff = abs(datetime.strptime(str(day1), "%Y-%m-%d") - datetime.
↪strptime(str(day2), "%Y-%m-%d"))
    no_days = diff.days
    return no_days

def hour_diff(time1, time2):
    diff = abs(datetime.strptime(time1, "%H:%M:%S") - datetime.strptime(time2,␣
↪"%H:%M:%S"))
    diff = (diff.total_seconds()) / 3600
    return diff

def gaussian_kernel(u, h):
    return np.exp(-u**2 / (2 * h**2))

def sum_kernel(distance_kernel, day_kernel, time_kernel):
    res = distance_kernel + day_kernel + time_kernel
    return res

def product_kernel(distance_kernel, day_kernel, time_kernel):
    res = distance_kernel * day_kernel * time_kernel
    return res

# Value to predict
target_latitude = 58.68681
target_longitude = 15.92183
target_date = '2014-05-17'
hour_list= ["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"]


# Kernel width
h_dist = 100
h_day = 15
h_time = 5

# Create SparkContext
sc = SparkContext(appName="Lab 3 ML")

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

temperature_data = temperature_file.map(lambda x: x.split(';'))
stations_data = station_file.map(lambda x: x.split(';'))

#Filter the previous data, this can save some computation
target_date_strip = datetime.strptime(target_date, '%Y-%m-%d')
```

```python
prev_temp = temperature_data.filter(lambda x: datetime.strptime(x[1], '%Y-%m-%d') < target_date_strip)

# pre-calculate station_distkernel and broadcast it (to speed up process)
station_distkernel = stations_data.map(lambda x: (x[0], gaussian_kernel(haversine(target_longitude, target_latitude, float(x[4]), float(x[3])), h_dist))).collectAsMap()
broadcast_station_distkernel = sc.broadcast(station_distkernel)

#station id, time, temp, day_kernel, cache it to save time
temperature_data_datekernel = prev_temp.map(lambda x: (x[0], x[2], x[3], gaussian_kernel(day_difference(target_date, x[1]), h_day))).cache()


predictions = {}
for hour in hour_list:
    #temp, dist_kernel, day_kernel, hour_kernel
    temp_kernels = temperature_data_datekernel.map(lambda x: (float(x[2]),
                                                              broadcast_station_distkernel.value[x[0]],
                                                              x[3],
                                                              gaussian_kernel(hour_diff(hour, x[1]), h_time)))
    #temp,sumkernel, prodkernel
    temp_both_kernels = temp_kernels.map(lambda x: (x[0], sum_kernel(x[1], x[2], x[3]), product_kernel(x[1], x[2], x[3])))

    #sum_kernel, sum_kernel*temp, prod_kernel, prod_kernel*temp
    cal_kerneltemp = temp_both_kernels.map(lambda x : (x[1],x[0]*x[1],x[2],x[0]*x[2]))

    #sum all the elements
    sum_kerneltemp = cal_kerneltemp.reduce(lambda x, y: (x[0] + y[0], x[1] + y[1], x[2] + y[2], x[3] + y[3]))

    #sum_prediction,prod_prediction
    predictions[hour]=(sum_kerneltemp[1]/sum_kerneltemp[0],sum_kerneltemp[3]/sum_kerneltemp[2])

# Convert predictions dictionary to RDD
predictions_rdd = sc.parallelize(list(predictions.items()))
predictions_rdd = predictions_rdd.coalesce(1)
predictions_rdd = predictions_rdd.sortByKey()
# Save the RDD as text files
predictions_rdd.saveAsTextFile("BDA/output/predictions")
```

## 1.2 Time(Hour),Prediction using summation kernel, Prediction using Prodoct kernel

('00:00:00', (4.2202401242697389, 5.2914482209815956))
('04:00:00', (4.3552953714587552, 6.3959774649343277))
('06:00:00', (4.6245740528807513, 7.1339973346514123))
('08:00:00', (4.9627767000765868, 7.9163683859293865))
('10:00:00', (5.2939137306831405, 8.6202196406263436))
('12:00:00', (5.5383779571661025, 9.1060230972701959))
('14:00:00', (5.6461283024140405, 9.2821452649169416))
('16:00:00', (5.6191253696649843, 9.1510902056699788))
('18:00:00', (5.5036901822620719, 8.7955867007308655))
('20:00:00', (5.3640549651923441, 8.3245303183460653))
('22:00:00', (5.2613166212619547, 7.8280808797063388))

## 1.3 MLlib

[ ]:
```
from __future__ import division
from pyspark import SparkContext
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.feature import StandardScaler
from pyspark.mllib.regression import LabeledPoint, LinearRegressionWithSGD
from pyspark.mllib.tree import DecisionTree
from datetime import datetime
from math import radians, cos, sin, asin, sqrt, exp
from pyspark.broadcast import Broadcast

# For data handling, the plan is as below
# Use the distance difference from the geographical center of sweden
# Use how many dates have passed since 1950/01/01
# Use the hour difference from 00:00:00
# Which means the features are distance,day_diff, hour_diff,

# Function to calculate haversine distance
def haversine(lon1, lat1, lon2=16.321998712, lat2=62.38583179): #geographical␣
 ↪center of sweden
    # 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 day_diff(day1, day2="1950-01-01"):
```

```python
    diff = abs(datetime.strptime(str(day1), "%Y-%m-%d") - datetime.strptime(str(day2), "%Y-%m-%d"))
    no_days = diff.days
    return no_days

def hour_diff(time1, time2="00:00:00"):
    diff = abs(datetime.strptime(time1, "%H:%M:%S") - datetime.strptime(time2, "%H:%M:%S"))
    diff = (diff.total_seconds()) / 3600
    return diff


sc = SparkContext(appName="Lab 3 ML")
target_date = '2014-5-17'
target_latitude = 58.68681
target_longitude = 15.92183

target_distance = haversine(lon1=target_longitude, lat1=target_latitude)
target_date_diff = day_diff(day1=target_date)

temperature_file = sc.textFile("BDA/input/temperature-readings.csv")
#temperature_file = temperature_file.sample(withReplacement=False, fraction=0.1)
temperature_data = temperature_file.map(lambda x: x.split(';'))

#filter out data after target date
target_date_strip = datetime.strptime(target_date, '%Y-%m-%d')
prev_temp = temperature_data.filter(lambda x: datetime.strptime(x[1], '%Y-%m-%d') < target_date_strip)


station_file = sc.textFile("BDA/input/stations.csv")
stations_data = station_file.map(lambda x: x.split(';'))

#Same for kernel model, broadcast distance for faster access
stations_distance = stations_data.map(lambda x: (x[0], haversine(lat1=float(x[3]), lon1=float(x[4])))).collectAsMap()
broadcast_stations_distance = sc.broadcast(stations_distance)

training_temp = prev_temp.map(lambda x: (
    float(x[3]), day_diff(x[1]), hour_diff(x[2]), broadcast_stations_distance.value[x[0]]))

#standardized
features = training_temp.map(lambda x: x[1:])
standardizer = StandardScaler()
model = standardizer.fit(features)
features_transform = model.transform(features)
```

```python
label = training_temp.map(lambda x: x[0])
standardized_data = label.zip(features_transform)
#create training data with standardized features
train_data = standardized_data.map(lambda x: LabeledPoint(x[0], [x[1]]))

##Create models##

#Here the l2 regulation, step_size, iterations are set to avoid huge weight
lr_model = LinearRegressionWithSGD.train(train_data, regType='l2', step=0.25,␣
↪iterations = 50)

#Here, maxDepth is set to increase the split of model
dt_model = DecisionTree.trainRegressor(train_data, categoricalFeaturesInfo={},␣
↪maxDepth= 15)


# Create 2 hours interval target, and make it a rdd
hour_list = ["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"]

target_features = []
for hour in hour_list:
    target_hour = hour_diff(time1=hour)
    target_feature = Vectors.dense([float(target_date_diff), target_hour,␣
↪target_distance])
    target_features.append(target_feature)
target_features_rdd = sc.parallelize(target_features)


# Standardized target features, this is important, since the training features␣
↪are also standardized
target_features_transform = model.transform(target_features_rdd)
standardized_target_features = target_features_transform.collect()



# Predictions
lr_predictions = [lr_model.predict(feature) for feature in␣
↪standardized_target_features]
dt_predictions = [dt_model.predict(feature) for feature in␣
↪standardized_target_features]



# save the predictions to a text file
```

```python
lr_predictions = zip(hour_list, lr_predictions)
lr_predictions = sc.parallelize(lr_predictions)
lr_predictions = lr_predictions.coalesce(1).sortByKey()
lr_predictions.saveAsTextFile("BDA/output/lr_prediction")

dt_predictions = zip(hour_list, dt_predictions)
dt_predictions = sc.parallelize(dt_predictions)
dt_predictions = dt_predictions.coalesce(1).sortByKey()
dt_predictions.saveAsTextFile("BDA/output/dt_predictions")
```

### 1.4 Output of linear regression

('00:00:00', 3.1035369255720253)
('04:00:00', 3.6499689072471697)
('06:00:00', 3.9231848980847417)
('08:00:00', 4.1964008889223141)
('10:00:00', 4.4696168797598865)
('12:00:00', 4.7428328705974581)
('14:00:00', 5.0160488614350296)
('16:00:00', 5.289264852272602)
('18:00:00', 5.5624808431101744)
('20:00:00', 5.8356968339477469)
('22:00:00', 6.1089128247853193)

### 1.5 Output of Decision Tree

('00:00:00', 4.891478190630056)
('04:00:00', 4.635169183558103)
('06:00:00', 5.317851639885542)
('08:00:00', 5.983913995031351)
('10:00:00', 7.989802623632015)
('12:00:00', 7.989802623632015)
('14:00:00', 7.989802623632015)
('16:00:00', 7.989802623632015)
('18:00:00', 6.873959938366718)
('20:00:00', 4.760130335412936)
('22:00:00', 5.371705739692806)

#### 1.5.1 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.

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.

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.

### 1.5.2 Answer

**Ans1.** We choose below, kernels' width
h_dist = 100 km h_day = 15 days h_time = 5 hours Since the weather changes greatly across large distances. So we set the kernel width for distance as 100 km, which give larger weight to nearby station that is(< 100km).

For date kernel, since the temperature will varies from different seasons, and we will like to keep the larger weight to the observation closer to the target date. However, it is important to note that, this will also decrease the weight of the same date from previous year, but for simplicity, we keep it as 15

For time(hour) kernel, since we set the target location in a random place in Östergötlands län, and we experient large fluctuations of temperature from day and night recently. Hence, we choose 5 hour as our kernel's width to put more emphasize to closer hour.

**Ans2.** By observing our prediction, we can see that the prediction using summation model is lower and also does not vary much compare to multiplication model. This is due to the difference of combanation For example, if there is a observation that is at the same hour, one day before target date, but 1000km away. Using summation model it will still result in large weight because of hour and date. On the other hand, using multiplication the small weight of distance will keep the overall weight down. We will say that the multiplication model is more ideal because we need to consider all three kernels when doing prediction.

**Ans3.** The first model we use LinearRegression(LR) with L2 Regulation (RidgeRegression), the using of regulation is to get smaller and more balanced coefficients since we experience unbalanced coefficients when using default LinearRegression.
The Second model we use is Decision Tree regression since it is usable to capture non-linear relationship between variables.

In terms of result, we will say that LR model did not do the prediction well, as the temperature around noon is lower than night time. This might because the amount of data for different hour is not balanced so the coefficient is not good.

For Decision Tree(DT), the time of higher and lower temperature are as expected. However, it will output same value for different hour. We suspect this is due to the depth of the model, but increase the max_depth will further increase run time.

Overall, we will say that DT gives more accurate prediction than LR, but both not perfect. Also, the way we handle data (by giving reference starting point for location and date) have make our MLlib model result different to Gaussian kernels.

```
[ ]:
```