Flux Training - Day 2

InfluxDays 2020 - London

Authors


```
Marco Balduini - marco.balduini@quantiaconsulting.com
Emanuele Della Valle - emanuele.dellavalle@quantiaconsulting.com
Riccardo Tommasini - riccardo.tommasini@quantiaconsulting.com
```


Use Case Recall - Linear Pizza Oven

We have a linear oven to continuously cook pizza.

The cooking operation has two main steps:

- the cooking of the pizza base, and
- the mozzarella melting area.
- There are two sensors:

- S1 measures the temperature and the relative humidity of the pizza base cooking area.

- S2 measures the temperature and the relative humidity of the mozzarella melting area.

Both sensors send a temperature measurement every minute, but are not synchronised.

Note

- Start Time: 2020-06-07 12:00:00 GMT (2020-06-07T12:00:00Z)
- End Time: 2020-06-07 12:05:00 GMT (2020-06-07T12:05:00Z)

07 - Map and Custom Functions

Correct the temperature observations of the cooking base area by by subtracting a delta of 5°C to each value

doc

Inline map

```
from(bucket: "training")
  |> range(start: 2020-06-07T12:00:00Z, stop: 2020-06-07T12:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> filter(fn: (r) => r.sensor == "S1")
  |> map(fn: (r) => ({
      r with
      correctValue: r._value - 5.0
    }))
```

Note the `r with` clause maintains alla the original columns and adds the new one.

Create a custom function to be used in the inline map

```
from(bucket: "training")
  |> range(start: 2020-06-07T12:00:00Z, stop: 2020-06-07T12:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> filter(fn: (r) => r.sensor == "S1")
  |> map(fn: (r) => ({
      r with
      correctValue: adjValue(x:r._value, y:-5.0)
  }))
```

Create a costume pipe forwardable function that contains a map

```
adjValues = (tables=<-, x) =>
  tables
    |> map(fn: (r) => ({ r with correctValue: r._value + x}))

from(bucket: "training")
  |> range(start: 2020-06-07T12:00:00Z, stop: 2020-06-07T12:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> filter(fn: (r) => r.sensor == "S1")
  |> adjValues(x:-5.0)
```

Note Most Flux functions manipulate data piped-forward into the function. In order for a custom function to process piped-forward data, one of the function parameters must capture the input tables using the `<-` pipe-receive expression.

08 - Join

doc

Extract the difference between the temperature of the base cooking area and the mozzarella melting area

Join assuming synchronous time-series

```
ts = from(bucket: "training")
  |> range(start: 2020-06-07T12:00:00Z, stop: 2020-06-07T12:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")

ts1 = ts |> filter(fn: (r) => r.sensor == "S1")
ts2 = ts |> filter(fn: (r) => r.sensor == "S2")

join(tables: {key1: ts1, key2: ts2}, on: ["_time"], method: "inner")
```

Note No results!!!

Join assuming a fixed delta (timeShift)

Note Use `timeShift()` function. doc

```
ts = from(bucket: "training")
  |> range(start: 2020-06-07T12:00:00Z, stop: 2020-06-07T12:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")

ts1 = ts |> filter(fn: (r) => r.sensor == "S1")
  |> timeShift(duration: 45s, columns: ["_start", "_stop", "_time"])

ts2 = ts |> filter(fn: (r) => r.sensor == "S2")

join(tables: {key1: ts1, key2: ts2}, on: ["_time"], method: "inner")
  |> map(fn: (r) => ({ r with _value: r._value_key1 - r._value_key2 }))
  |> drop(columns: ["_measurement_key1", "_measurement_key2", "_start_key1", "_start_key2",
```

Note You are missing values!!!

Join on time exploiting windows

Note Use `aggregateWindow()` function. doc

```
ts = from(bucket: "training")
  |> range(start: 2020-06-07T12:00:00Z, stop: 2020-06-07T12:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")

ts1 = ts |> filter(fn: (r) => r.sensor == "S1")
  |> aggregateWindow(every: 1m, fn: mean)

ts2 = ts |> filter(fn: (r) => r.sensor == "S2")
  |> aggregateWindow(every: 1m, fn: mean)

join(tables: {key1: ts1, key2: ts2}, on: ["_time"], method: "inner")
  |> map(fn: (r) => ({ r with _value: r._value_key1 - r._value_key2 }))
  |> drop(columns: ["_measurement_key1", "_measurement_key2", "_start_key1", "_start_key2",
```

OR

```
ts = from(bucket: "training")
  |> range(start: 2020-06-07T12:00:00Z, stop: 2020-06-07T12:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> aggregateWindow(every: 1m, fn: mean)

ts1 = ts |> filter(fn: (r) => r.sensor == "S1")
ts2 = ts |> filter(fn: (r) => r.sensor == "S2")

join(tables: {key1: ts1, key2: ts2}, on: ["_time"], method: "inner")
  |> map(fn: (r) => ({ r with _value: r._value_key1 - r._value_key2 }))
  |> drop(columns: ["_measurement_key1", "_measurement_key2", "_start_key1", "_start_key2",
```

Extract the difference between the humidity levels of the base cooking area and the mozzarella melting area. Find if the differences are between 20% and 30%

Join assuming a fixed delta (timeShift)

Note Use `timeShift()` function. doc

```
import "math"

hs = from(bucket: "training")
  |> range(start: 2020-06-07T12:00:00Z, stop: 2020-06-07T12:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "humidity")

hs1 = hs |> filter(fn: (r) => r.sensor == "S1")
  |> timeShift(duration: 45s, columns: ["_start", "_stop", "_time"])

hs2 = hs |> filter(fn: (r) => r.sensor == "S2")

join(tables: {key1: hs1, key2: hs2}, on: ["_time"], method: "inner")
  |> map(fn: (r) => ({ r with _value: math.abs(x:(r._value_key1 - r._value_key2)) }))
  |> filter(fn: (r) => r._value < 20 and r._value < 30)
  |> drop(columns: ["_measurement_key1", "_measurement_key2", "_start_key1", "_start_key2",
```

Note You are missing values!!!

Join on time exploiting windows

Note Use `aggregateWindow()` function. doc

```
import "math"

hs = from(bucket: "training")
  |> range(start: 2020-06-07T12:00:00Z, stop: 2020-06-07T12:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "humidity")


hs1 = hs |> filter(fn: (r) => r.sensor == "S1")
  |> aggregateWindow(every: 1m, fn: mean)

hs2 = hs |>filter(fn: (r) => r.sensor == "S2")
  |> aggregateWindow(every: 1m, fn: mean)

join(tables: {key1: hs1, key2: hs2}, on: ["_time"], method: "inner")
  |> map(fn: (r) => ({ r with _value: math.abs(x:(r._value_key1 - r._value_key2)) }))
  |> filter(fn: (r) => r._value > 20 and r._value < 30)
  |> drop(columns: ["_measurement_key1", "_measurement_key2", "_start_key1", "_start_key2",
```

Alerts and Task

Record the temperature dispersion in the base-cooking area

```
option v = {
  bucket: "",
  timeRangeStart: -1h,
  timeRangeStop: now()
}

option task = {
  name: "Record the temperature dispersion in the base-cocking area",
  every: 1m,
}

from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "iot-oven")
  |> filter(fn: (r) => r["_field"] == "temperature")
  |> filter(fn: (r) => r["sensor"] == "S1")
  |> to(bucket: "task-output", org: <your-org>)
```

Advanced Flux

Time series enrichment

```
import "sql"

pizzas = sql.from(
  driverName: "postgres",
  dataSourceName: "postgresql://<user>:<pwd>@<server>:<port>/pizza-erp?sslmode=disable",
  query:"SELECT *
         FROM oven
         WHERE enteringtime/1000000000 >= extract(epoch from timestamp '2020-06-07T12:00:00
              enteringtime/1000000000 < extract(epoch from timestamp '2020-06-07T12:50:00')
        "
)

obs = from(bucket: "training")
  |> range(start: 2020-06-07T12:00:00Z, stop: 2020-06-07T12:10:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> drop(columns: ["_measurement", "_start", "_stop"])

join(tables: {p: pizzas, o: obs}, on: ["sensor" ], method: "inner")
  |> filter(fn: (r) => uint(v: r._time) >= r.enteringtime and uint(v: r._time) < r.exitingt
  |> group(columns: ["pid", "kind"], mode:"by")
```

Anomaly Detection

```
import "date"
import "math"

movingAvg = from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> aggregateWindow(every: 5m, fn: mean, createEmpty: false)
  |> filter(fn: (r) => r._stop != r._time)
  |> drop(columns: ["_start", "_stop", "_measurement"])
  |> rename(columns: {_value: "avg"})

movingStddev = from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> aggregateWindow(every: 5m, fn: stddev, createEmpty: false)
  |> filter(fn: (r) => r._stop != r._time)
  |> drop(columns: ["_start", "_stop", "_measurement"])
  |> rename(columns: {_value: "stddev"})
//  |> filter(fn: (r) => not(math.isNaN(f: r.stddev)))

allData = from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> drop(columns: ["_start", "_stop", "_measurement"])

join1 = join(tables: {all: movingAvg, avg: movingStddev}, on: ["_time","sensor","_field"], r

join2 = join(tables: {all: allData, j: join1}, on: ["sensor","_field"], method: "inner")
  |> filter(fn: (r) =>   uint(v: r._time_all) - uint(v: r._time_j) > 0 )
  |> filter(fn: (r) =>   uint(v: r._time_all) - uint(v: r._time_j) <= (5*60*1000000000))
  |> map(fn: (r) => ({ r with _value: if  math.abs(x: (r._value - r.avg)/r.stddev)>4 then 1
  |> group(columns: ["sensor", "_field"], mode:"by")

join2
```

Holt-Winters

Show real data

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "fill_level")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.tank == "B2")
```

Downsample the data

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "fill_level")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.tank == "B2")
  |> aggregateWindow(every: 1m, fn: first)
```

Apply hot-winters on downsampled data

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "fill_level")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.tank == "B2")
  |> aggregateWindow(every: 1m, fn: first)
  |> holtWinters(n: 10, seasonality: 8, interval: 1m, withFit: true)
```

NOTE use `withFit: false` to hide the fitted data

BC2 - City Water Tank dashboard queries (part 2)

Needed input files:

- valve-state-lp.txt

Level difference every minute

```
actualMeanLevel = from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "fill_level")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.tank == "city_water")
  |> aggregateWindow(every: 1m, fn: mean, createEmpty: false)


preMeanLevel = actualMeanLevel
  |> timeShift(duration: -1m)

join(
  tables: {pre:preMeanLevel, actual:actualMeanLevel},
  on: ["_time"]
) |> map(fn: (r) => ({
    _time: r._time,
    _value:  r._value_pre - r._value_actual
    }))
```

Open valve(s)

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "valve_state")
  |> filter(fn: (r) => r._field == "value")
  |> last()
  |> map(fn: (r) => ({
      r with
      _value:
      if r._value == true then 1
      else 0
    }))
  |> group()
  |> sum(column: "_value")
```

Fill Level of Tanks

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "fill_level")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.tank != "city_water")
  |> aggregateWindow(every: 1m, fn: mean, createEmpty: false)
  |> pivot(rowKey: ["_time"], columnKey: ["tank"], valueColumn: "_value")
  |> map(fn: (r) => ({ r with _value : r.A1 + r.A2 + r.B1 + r.B2 }))
```

Total Flow Rate

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "flow_rate")
  |> filter(fn: (r) => r._field == "value")
  |> last()
  |> group()
  |> sum(column: "_value")
```

Current Status of Valve V1

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "valve_state")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.valve == "V1")
  |> last()
  |> map(fn: (r) => ({
      r with
      _value:
      if r._value == true then 1
      else 0
    }))
```

Current Status of Valve V2

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "valve_state")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.valve == "V2")
  |> last()
  |> map(fn: (r) => ({
      r with
      _value:
      if r._value == true then 1
      else 0
    }))
```

Current Status of Valve V3

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "valve_state")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.valve == "V3")
  |> last()
  |> map(fn: (r) => ({
      r with
      _value:
      if r._value == true then 1
      else 0
    }))
```

Current Status of Valve V4

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "valve_state")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.valve == "V1")
  |> last()
  |> map(fn: (r) => ({
      r with
      _value:
      if r._value == true then 1
      else 0
    }))
```