POC_PyGMQL

March 27, 2017

1 Proof of concept: *PyGMQL*

```
In [1]: import gmql as gl
        import numpy as np
        import pandas as pd
```

1.1 Loading a dataset

```
In [2]: # path of the dataset
        input_path = "/home/luca/Scrivania/GMQL-Python/resources/hg_narrowPeaks/"
```

The library provides the user with a set of different parsers for datasets. In particular, for this demonstration we use the `NarrowPeakParser`.

```
In [3]: np_parser = gl.parsers.NarrowPeakParser()
```

One of the main abstractions of the library is the `GMQLDataset`. It is the main access point of the user to the data. Each data manipulation operation is operated on a `GMQLDataset` and returns a *new* object of the same type.

```
In [4]: dataset = gl.GMQLDataset(parser=np_parser)
```

```
In [5]: dataset = dataset.load_from_path(path=input_path)
```

```
2017-03-27 19:00:48,082 - gmql_logger - INFO - loading metadata
2017-03-27 19:00:59,205 - gmql_logger - INFO - parsing metadata
2017-03-27 19:00:59,211 - gmql_logger - INFO - collecting metadata
2017-03-27 19:01:31,010 - gmql_logger - INFO - dataframe construction


100%|| 115/115 [01:48<00:00,  1.04it/s]

2017-03-27 19:03:19,794 - gmql_logger - INFO - loading region data




2017-03-27 19:03:22,106 - gmql_logger - INFO - parsing region data
```

1.2 Metadata management

Differently from the GMQL query engine, PyGMQL stores metadata directly in local memory as a **pandas dataframe** whose index (`id_sample`) is the sample id generated by the GMQL engine (based on the hash of the file name from which the sample comes from). Each column of the dataframe represents one of the found attributes in the dataset, therefore it is possible that a sample (a row) has zero values for one column.

Other important fact is that each cell of the metadata dataframe is a **list** due to the fact that an attribute can have multiple values.

```
In [6]: # for visualization purposes we only show 3 columns of
        # the dataframe (for a total of 115) and we use the 'head'
        # function to show only the first rows of the dataframe
        dataset.meta_dataset.head()[['antibody','cell','antibody_lab']]
```

Out[6]:

id_sample	antibody	cell	antibody_lab
-9220584259719780958	[E2F6]	[HeLa-S3]	[Farnham, Myers]
-9219803729611809025	[CTCF]	[AG04449]	[Myers, Hardison, Snyder]
-9211745969161520790	[eGFP-HDAC8]	[K562]	[White]
-9210823539931286134	[GABP]	[H1-hESC]	[Myers]
-9197969371849812436	[]	[HPF]	[]

1.2.1 Select based on metadata with a logical predicate

We select the samples of the dataset in which 'antibody' has 'CTCF' value. The function to be applied is the `meta_select` which accepts a generic predicate, which is basically an arbitrary complex function; in this example we use a lambda expression. The selection affects both metadata and region samples.

```
In [7]: filtered_dataset = dataset.meta_select(lambda row: 'CTCF' in row['antibody'])
```

```
In [8]: # visualize only the first rows of the metadata pandas dataframe
        filtered_dataset.meta_dataset.head()[['antibody','cell','antibody_lab']]
```

Out[8]:

id_sample	antibody	cell	antibody_lab
-9219803729611809025	[CTCF]	[AG04449]	[Myers, Hardison, Snyder]
-9120762041249846625	[CTCF]	[MCF-7]	[Myers, Hardison, Snyder]
-9118037537398139811	[CTCF]	[MCF-7]	[Myers, Hardison, Snyder]
-8760850962206896694	[CTCF]	[MCF-7]	[Myers, Hardison, Snyder]
-8556045950597285261	[CTCF]	[A549]	[Myers, Hardison, Snyder]

We can use the function `get_reg_sample(n)` to materialize a little sample of the regions in memory to a pandas dataframe

```
In [9]: filtered_dataset.get_reg_sample(1)
```

Out[9]:

	chr	id_sample	name	pValue	peak	qValue	score	signalValue	start	stop	strand
0	chr1	-4153672139227562561	.	4.03439	-1.0	-1.0	0.0	9.0	16120	16270	*

1.2.2 Project metadata based on an attribute list

An other possible metadata operation is the `meta_project` which, in its simplest form, takes only the specified columns of the dataframe and (as always) returns a new `GMQLDataset`

```
In [10]: filtered_proj_data = filtered_dataset.meta_project(['antibody', 'cell'])
         filtered_proj_data.meta_dataset.head()
```

```
Out[10]:                      antibody       cell
         id_sample
         -9219803729611809025   [CTCF]  [AG04449]
         -9120762041249846625   [CTCF]    [MCF-7]
         -9118037537398139811   [CTCF]    [MCF-7]
         -8760850962206896694   [CTCF]    [MCF-7]
         -8556045950597285261   [CTCF]     [A549]
```

1.2.3 Add a new column

If the user wants to add a new attribute to the metadata (basically a new column of the dataframe) he needs to call the `add_meta` function that takes the name of the new attribute and the default value to assign to each sample of the dataset

```
In [11]: filtered_proj_data = filtered_proj_data.add_meta('creator', 'luca')
         filtered_proj_data.meta_dataset.head()
```

```
Out[11]:                      antibody       cell creator
         id_sample
         -9219803729611809025   [CTCF]  [AG04449]  [luca]
         -9120762041249846625   [CTCF]    [MCF-7]  [luca]
         -9118037537398139811   [CTCF]    [MCF-7]  [luca]
         -8760850962206896694   [CTCF]    [MCF-7]  [luca]
         -8556045950597285261   [CTCF]     [A549]  [luca]
```

```
In [12]: # we can visualize all the attribute names
         all_attributes = filtered_proj_data.get_meta_attributes()
         all_attributes
```

```
Out[12]: ['antibody', 'cell', 'creator']
```

1.2.4 Project and also compute new columns based on complex functions

The `meta_project` function can take an other argument which is a dictionary of the following type:

```
new_attributes = {
    'new_attribute_name_1' : complex_function_1,
    'new_attribute_name_2' : complex_function_2,
    ...

    'new_attribute_name_N' : complex_function_N,
}
```

This argument enables the user to build new columns/attributes of the metadata dataframe based on the values of the other attributes.

```
In [13]: # define a function that operates on rows of the metadata dataset and
         # gives us the resulting new column value

         # in particular this function simply concatenates the lists of
         # antibody and cell values
         def complex_function(row):
             x = list(row['antibody'])
             y = list(row['cell'])
             #print("antibody: {}\t cell: {}".format(x, y))
             return x + y

In [14]: new_attr_dict = {
             'extended' : complex_function
         }

         extended_dataset = filtered_proj_data.meta_project(attr_list=all_attributes,
                                                            new_attr_dict=new_attr_dict)

In [15]: extended_dataset.meta_dataset.head()
```

Out[15]:

id_sample	antibody	cell	creator	extended
-9219803729611809025	[CTCF]	[AG04449]	[luca]	[CTCF, AG04449]
-9120762041249846625	[CTCF]	[MCF-7]	[luca]	[CTCF, MCF-7]
-9118037537398139811	[CTCF]	[MCF-7]	[luca]	[CTCF, MCF-7]
-8760850962206896694	[CTCF]	[MCF-7]	[luca]	[CTCF, MCF-7]
-8556045950597285261	[CTCF]	[A549]	[luca]	[CTCF, A549]

1.3 *Example: working with metadata*

We demonstrate the usage of some of the function described above with a very simple (and stupid) example. The user adds two new attributes (the same for every sample) describing birth date and death date of the patient. Then he generates a third new attribute given by the other two that represents the age of the patient.

```
In [16]: from datetime import datetime

         born_date = datetime.strptime("30 Nov 1935","%d %b %Y")
         death_date = datetime.strptime("30 Nov 1999","%d %b %Y")

In [17]: example_dataset = filtered_proj_data.add_meta('born_date', born_date)
         example_dataset = example_dataset.add_meta('death_date', death_date)
         all_attributes = example_dataset.get_meta_attributes()
         all_attributes
```

Out[17]: ['antibody', 'cell', 'creator', 'born_date', 'death_date']

```
In [18]: def calculate_age(row):
             #print(row)
             born_date = row['born_date'][0]
             death_date = row['death_date'][0]
             return (death_date - born_date).days / 365

In [19]: new_attr_dict = {
             'age' : calculate_age
         }
         example_dataset = example_dataset.meta_project(attr_list=all_attributes,
                                                        new_attr_dict=new_attr_dict)

In [20]: example_dataset.meta_dataset.head()
```

Out[20]:

id_sample	antibody	cell	creator	born_date	death_date	age
-9219803729611809025	[CTCF]	[AG04449]	[luca]	[1935-11-30 00:00:00]	[1999-11-30 00:00:00]	[64.04383561643836]
-9120762041249846625	[CTCF]	[MCF-7]	[luca]	[1935-11-30 00:00:00]	[1999-11-30 00:00:00]	[64.04383561643836]
-9118037537398139811	[CTCF]	[MCF-7]	[luca]	[1935-11-30 00:00:00]	[1999-11-30 00:00:00]	[64.04383561643836]
-8760850962206896694	[CTCF]	[MCF-7]	[luca]	[1935-11-30 00:00:00]	[1999-11-30 00:00:00]	[64.04383561643836]
-8556045950597285261	[CTCF]	[A549]	[luca]	[1935-11-30 00:00:00]	[1999-11-30 00:00:00]	[64.04383561643836]