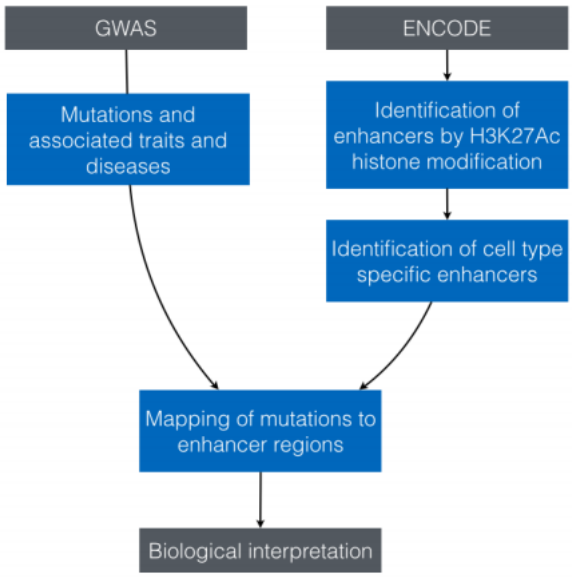
PyGML examples

April 12, 2017

1 General examples

In this example we implement the following pipeline:



Follows the GMQL query to be submitted at the system:

Listing 1: gmql_query_mutations.txt

```
# LOADING OF DATA FROM A REPOSITORY
# Mutations comes from a genome wide association study
GWAS = SELECT ( ) GWAS;

# We select the experiments related to acetylation
# at the 27th lysine residue of the histone protein 3 (H3K27Ac) from
# a genomic assembly for humans (HG19)
Ac = SELECT ( experiment_target == H3K27ac human ) HG19_ENCODE_BROAD;

# PREPROCESSING
```

```
# I take the middle of the regions
peaked = PROJECT ( region_update : peak AS RIGHT/2 + LEFT / 2 ) Ac ;
# Extend the region center by +- 1.5Kb
large = PROJECT( region_update : LEFT AS peak - 1500,
                                 RIGHT AS peak + 1500) peaked;

# MUTATION DISCOVERY
# merge replicats together
rep = COVER(groupby: biosample_term_name) large;

# find the cell-line specific enhancers
S = COVER(1,2) REP;
RepCount = MAP() REP S;
CSE = SELECT(region : count_REP_S > 0) RepCount;

# find the mutation occurring in those enhancers
M = MAP(bag AS BAG(trait)) CSE GWAS;
N = SELECT(region : count_Z_GWAS > 0) M;
P = PROJECT(count_Z_GWAS, bag) N;

MATERIALIZE P int test_final;
```

This is the same query but in the Python interactive mode:

```
import gmql as gl
from gmql.expressions import *

# We can use different parsers
gwas_parser = gl.parsers.GWASparser()
encode_parser = gl.parsers.ENCODEparser()

# We load the dataset we need
gwas_dataset = gl.GMQLDataset().load("gwas")
encode_dataset = gl.GMQLDataset().load("encode_hg19")

# When the datasets are firstly loaded, all the metadata are
# collected in memory and stored in a pandas dataframe.
# This enables the user to do an initial filtering
# of the properties that he is interested in.
# Due to the fact that we use a pandas dataframe we are able
# to do arbitrary filtering through lambda functions.

# This call selects all the samples that have as
# experiment target 'H3K27ac human', collects
# all the id_samples of the selected samples
# and send them to the GMQL engine in order to initially
# filter the dataset
Ac = encode_dataset.meta_select(lambda meta: meta.experiment_target ==
```

```
project_arg = {
  "peak": Record("RIGHT")/2 + Record("LEFT")/2
}
peaked = Ac.reg_project(new_attributes=project_arg)


project_arg = {
    "LEFT" : Record("peak") - 1500,
    "RIGHT": Record("peak") + 1500
}
large = peaked.reg_project(new_attributes=project_arg)

# merge replicats together
rep = large.cover(groupby=['biosample_term_name'])

# find the cell-line specific enhancers
S = rep.cover(minAcc=1, maxAcc=2)

# convention: the GMQLDataset calling map is the
# reference and the paramter of the function is the experiment
RepCount = rep.map(S)
CSE = RepCount.reg_select(condition = Record("count_REP_S") > 0)

# find the mutation occurring in those enhancers
M = CSE.map(gwas_dataset, {'bag' : BAG("trait")})
N = M.reg_select(condition = Record("count_Z_GWAS") > 0)
P = N.meta_project(['count_Z_GWAS','bag'])

# materialization in memory <---- notice that here we collect
# directly in the program for future analysis.
# The result is anyway stored in the HDFS for future access
materialized_P = P.materialize(filename="test_final")
```