GMQL: GenoMetrics Query Language

Simone Pallotta

2017-10-02

Contents

1 Introduction

Improvement of sequencing technologies and data processing pipelines is rapidly providing sequencing data, with associated high-level features, of many individual genomes in multiple biological and clinical conditions.
For this purpose GMQL has been proposed a high-level, declarative GenoMetric Query Language (GMQL) and a toolkit for its use.

1.1 Purpose

This package provides a set of functions to create, manipulate and extract genomic data from different datasources from local and remote datasets.
Also, these functios allow performing complex queries without the knowledge of GMQL syntax.

2 Dataset

We usually distinguish two kinds of dataset layout:
These contains large number of information describing regions of genome.
Data are encoded in human readable format using plain text file.

- GMQL standard layout :

  Dataset is composed basically of three type of file:
  1) region files usually terminating in .gtf or .gdm
  2) metadata files terminating in .meta

3) schema XML file containing regions attributes

Each region sample file owns its metadata file. All these files must reside in unique folder called files.

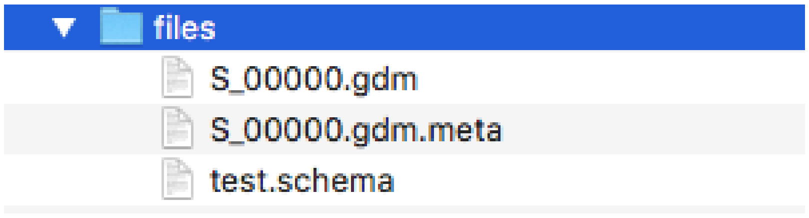

Figure 1: GMQL dataset folder

- Generic text based dataset:

  Dataset composed by heterogeneous sample organised in simple text files probably stem from different medical, biological sytem Sample files are simply contained on a folder whose name must be specified as input on read function.

In our package dataset files are considered read-only. Once read genomic information is represented in abstract structure inside package.

3 Basic Requirements

- javaSE version 8
- java environment correctly set (i.e JAVA_HOME)
- scala version 2.11.8
- scala environment correctly set (i.e SCALA_HOME)
- network connectivity to web services (if required)

4 How to Install

The GMQL package can be installed by executing the following R expression.

```
source("https://bioconductor.org/biocLite.R")
biocLite("GMQL")
```

5 Processing Environments

This package allow to create, manipulate and extract genomic data from different datasets using different processing modes.

5.1 Local Processing

Query processing consumes computational power directly from local CPUs/system while managing datasets (both GMQL or generic text plain dataset).

5.1.1 Initialisation

Before starting using any GMQL operation we need to initialise the GMQL context with the following code:

```
initGMQL()
```

No parameter means that we are initialising the context with GTF as output format for our regions sample files and metadata files. Of Course, other parameter are available.

5.1.2 Datasource

After initialisation we need to read the dataset. We present different source we can get the data from:
we can read local GMQL dataset:

```
gmql_dataset_path <- system.file("example","DATA_SET_VAR_GTF",package = "GMQL")
data_out = readDataset("gmql_dataset_path")
```

In case of remote datasets, user have to download it locally using specifying function:

```
test_url <- "http://130.186.13.219/gmql-rest"
login.GMQL(test_url)
downloadDataset(test_url,"dataset_test",path = getwd())
```

where *test_url* is a R variable that define URL of remote server where web services are located.
Once local, these datatset behave like local dataset as written above.
Also, for better integration in R environment and with other packages, we provide a function to read a GrangesList:

```
gr1 <- GRanges(seqnames = "chr2",
ranges = IRanges(103, 106),
strand = "+",
score = 5L, GC = 0.45)

gr2 <- GRanges(seqnames = c("chr1", "chr1"),
ranges = IRanges(c(107, 113), width = 3),
strand = c("+", "-"),
score = 3:4, GC = c(0.3, 0.5))

grl <- GRangesList("txA" = gr1, "txB" = gr2)
data_out <- read(grl)
```

Every read function return a value, this value is used as first step for execution the subsequent GMQL operation.

5.1.3 Queries

Thwe core concept of GMQL package is build a query as the name *GMQL* suggest. Unfortunatley is not the same as any query language. the building of query is more like a batch workflow

5.1.4 Execution

5.2 Remote Environment

Query processing consumes computational power from remote clusters/system while managing datasets that are only GMQL dataset.

Remote processing exits in two flavour:

- BATCH execution:
  Similar to local execution; user read data and the system automatically upload it on remote system: once loaded you can isue R function to manage remote data.

- REST web services:
  user can write GMQL queries to be executed remotely on remote data (or local data previous upload)

5.2.1 REST web services

We talk about only for REST web service porocessing, because batch remote processing is quite similar to local processing.

5.2.1.1 Initialization Rest servicies required login so the first step is to perform logon using user and password or as guest. Upon succesfull logon you get a request token must use in every subsequent REST call. Login can be performed using function:

```
library("GMQL")

test_url = "http://130.186.13.219/gmql-rest"
login.GMQL(test_url)
## [1] "your Token is 8c9fe841-199c-4b8a-b481-4eb42a03f44e"
```

that saves token in R environment.

5.2.2 Datasource

5.2.3 Queries

5.2.4 Execution

Saved data will be stored in repository and eventually can be downloaded locally.

6 Biological Example

This section collects several examples where GMQL is used to answer practical questions/tasks of biological and clinical interest. For each example, after an initial textual statement describing the question/task to be answered, the GMQL query is reported with a detailed commented description of the query and its results.