Using the GEOTools for MATLAB

Adapted from

Using the *GEOmetadb* package, R/Bioconductor

Yidong Chen[*], Jack Zhu[†] and Sean Davis[‡]
Genetics Branch, National Cancer Institute, National Institutes of Health

April 20, 2008

Contents

[*]yidong@mail.nih.gov
[†]zhujack@mail.nih.gov
[‡]sdavis2@mail.nih.gov

1 Overview of MATALAB GEOTools

The NCBI Gene Expression Omnibus (GEO) represents the largest repository of microarray data in existence. One difficulty in dealing with GEO is finding the microarray data that is of interest. As part of the NCBI Entrez search system, GEO can be searched online via web pages or using NCBI Eutils. However, the web search is not as full-featured as it could be, particularly for programmatic access. NCBI Eutils offers another option for finding data within the vast stores of GEO, but it is cumbersome to use, often requiring multiple complicated Eutils calls to get at the relevant information. We have found it **absolutely critical** to have ready access not just to the microarray data, but to the metadata describing the microarray experiments. To this end we have created *GEOmetadb*, and now part of tools are ported to MATLAB GEOTools. This document is largely adapted from '*Using the GEOmetadb Package*' vignette for R, replaced with examples with MATLAB/GEOTools.

All examples included in this user guide were texted under MATLAB 2007b with Intel Mac Pro. Other machines will be tested later. MATLAB Statistics toolbox and Bioinformatics toolbox are required for the Applications Section.

1.1 What is *GEOmetadb*?

The *GEOmetadb* is an attempt to make querying the metadata describing microarray experiments, platforms, and datasets both easier and more powerful. At the heart of *GEOmetadb* is a SQLite database that stores nearly all the metadata associated with all GEO data types including GEO samples (GSM), GEO platforms (GPL), GEO data series (GSE), and curated GEO datasets (GDS), as well as the relationships between these data types. This database is generated by our server by parsing all the records in GEO and needs to be downloaded via a simple helper function to the user's local machine before *GEOmetadb* is useful. Once this is done, the entire GEO database is accessible with simple SQL-based queries. With the GEOmetadb database, queries that are simply not possible using NCBI tools or web pages are often quite simple.

The relationships between the tables in the GEOmetadb SQLite database can be seen in figure 1.

1.2 What is GEOTools/MATLAB?

Modeled after *GEOmetadb*, The GEOTools/MATLAB is a set of MATLAB fuctions to access GEO meta data stored in SQLite database *GEOmetadb.sqlite*. Users are highly encouraged to read the original document of "Using the *GEOmetadb* Package" for R/Bioconductor. However, the main difference of this user guide from the original R/Bioconductor version are merely in MATLAB specific sectioin and all examples. Nothing else changed in terms of database, and actual result obtained from both packages.

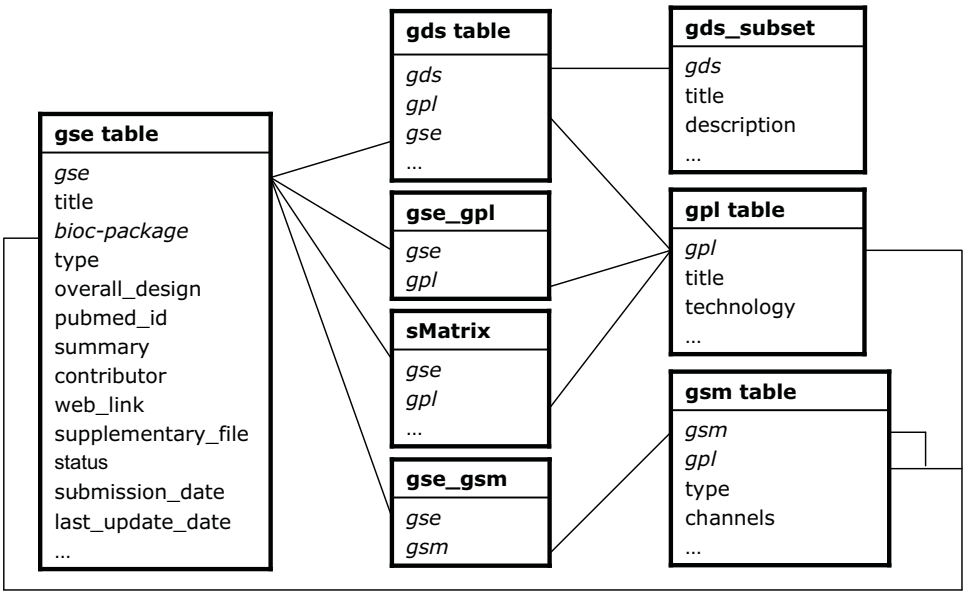

Figure 1: A graphical representation (sometimes called an *Entity-Relationship Diagram*) of the relationships between the tables in the GEOmetadb SQLite database

1.3 Conversion capabilities

A very typical problem for large-scale consumers of GEO data is to determine the relationships between various GEO accession types. As examples, consider the following questions:

- What samples are associated with GEO platform "GPL96", which represents the Affymetrix hgu133a array?
- What GEO Series were performed using "GPL96"?
- What samples are in my favorite three GEO Series records?
- How many samples are associated with the ten most popular GEO platforms?

Because these types of questions are common, *GEOmetadb* contains the function `geoConvert` that addresses these questions directly and efficiently.

1.4 What *GEOmetadb*, GEOTools/MATLAB is not

We have faithfully parsed and maintained in GEO when creating *GEOmetadb* and MATLAB *GEOTools*. This means that limitations inherent to GEO are also inherent in *GEOmetadb* and MATLAB *GEOTools*. We have made no attempt to curate, semantically recode, or otherwise "clean up" GEO; to do so would require significant resources, which we do not have.

GEOmetadb and GEOTools/MATLAB does not contain any microarray data. For access to microarray data from within R/Bioconductor, please look at the *GEOquery* package

and GEOTools/MATLAB. In fact, we would expect that many users will find that the combination of *GEOmetadb* and *GEOquery*, for R/Bioconductor, and GEOTools/MATLAB with Bioinformatics Toolbox/MATLAB are quite powerful.

2 Getting Started

Installation of MATLAB GEOTools are straightforward,

1. Download and unzip from http://meltzerlab.nci.nih.gov/apps/geo.
2. Copy the folder to a location you prefer (we suggest to put in a directory called /Applications/MATLAB_R2007b/UserPackages/GEOTools). Here directory /Applications/MATLAB_2007b is the *matlabroot* for current installation. User may change it according to his own MATLAB installation.
3. Launch MATLAB, and type

```
>> cd /Applications/MATLAB_R2007b/UserPackages/GEOTools
>> GEOToolsSetup
```

`GEOToolSetup` function will check for the latest update (you may want to run this program at a later time), download the *GEOmetadb.sqlite*, build the necessary *.mex* code, and then set the correct path for MATLAB access.

2.1 Getting the GEOmetadb database

This package does not come with a pre-installed version of the database. This has the advantage that the user will get the most up-to-date version of the database to start; the database can be re-downloaded using the same command as often as desired. Remember, after installing GEOTools and set the MATLAB path to the location, you are ready for downloading database. The download and uncompress steps are down automatically with a single command, `GEOgetSQLiteFile`.

```
>> GEOgetSQLiteFile();
The old version of 'GEOmetadb.sqlite' has been renamed to 'GEOmetadb.sqlite.backup'.
The SQLite DB 'GEOmetadb.sqlite' (size of 510.5 MB) has been installed in ...
```

The default storage location is in the location where *GEOgetSQLiteFile.m* function is, and the default filename is *GEOmetadb.sqlite*; it is best to leave the name unchanged unless there is a pressing reason to change it. The function also saves a *GEOmetadb_version.log* file such that if you invoke the program twice, or later, it will check the version number before downloading a new copy. However, if one would like to replace the SQLite database anyway, either the database *GEOmetadb.sqlite* and/or the log file *GEOmetadb_version.log* has to be deleted from the location.

Since this SQLite file is of key importance in *GEOmetadb* and MATLAB *GEOTools*, it is perhaps of some interest to know some details about the file itself.

```
>> ls( '-l GEOmetadb.sqlite' );
-rw-r--r--   1 yidong  yidong  535349248 Apr 14 18:57 GEOmetadb.sqlite'.
```

Now, the SQLite file is available for connection. Different from the standard *DBI* functionality as implemented in *RSQLite* function `dbConnect`, MATLAB GEOTools implementation connects to *GEOmetadb.sqlite* each time with SQL query command and closes connection immediately. The implementation is not efficient for large number of SQL queries, but sufficient enough for our purpose. Users are highly recommended to read MATLAB mex/C code, as well as SQLite C/C++ API (`www.sqlite.org/c3ref/intro.html`) for additional functionalities if needed.

2.2 A word about SQL

The Structured Query Language, or SQL, is a very powerful and standard way of working with relational data. GEO is composed of several data types, all of which are related to each other; in fact, NCBI uses a relational SQL database for metadata storage and querying. SQL databases and SQL itself are designed specifically to work efficiently with just such data. While the goal of many programming projects and programmers is to hide the details of SQL from the user, we are of the opinion that such efforts may be counterproductive, particularly with complex data and the need for *ad hoc* queries, both of which are characteristics with GEO metadata. We have taken the view that exposing the power of SQL will enable users to maximally utilize the vast data repository that is GEO. We understand that many users are not accustomed to working with SQL and, therefore, have devoted a large section of the vignette to working examples. Our goal is not to teach SQL, so a quick tutorial of SQL is likely to be beneficial to those who have not used it before. Many such tutorials are available online and can be completed in 30 minutes or less.

3 Examples

3.1 Interacting with the database

The functionality covered in this section is covered in much more detail in the SQLite web site. We have built some functions here only to be useful.

The `GetSQLiteTables()` function lists all the tables in the SQLite database. As we state before, we embeded database connection and closing within each function all. Therefore, you do not need to perform open/close connections as other database tools normally required (see MATLAB Database Toolbox, or *RSQLite*). In following example, table names are stored in `geoTables`. The function also prints out all table names.

```
>> GetSQLiteTables( 'GEOmetadb.sqlite' );

 No.   | Table Name
------+------------------
   1   | gds
```

```
   2  | gds_subset
   3  | geoConvert
   4  | geodb_column_desc
   5  | gpl
   6  | gse
   7  | gse_gpl
   8  | gse_gsm
   9  | gsm
  10  | metaInfo
  11  | sMatrix
------+----------------}
```

There is also the `ListSQLiteFields()` function that can list database fields associated with a table. Note that if table name is not specified, the function behaves just as `GETSQLiteTables`.

```
>> ListSQLiteFields( 'GEOmetadb.sqlite', 'gse' )

 No.  | Name                       | Type
------+----------------------------+-----------
   1  | ID                         | REAL
   2  | title                      | TEXT
   3  | gse                        | TEXT
   4  | status                     | TEXT
   5  | submission_date            | TEXT
   6  | last_update_date           | TEXT
   7  | pubmed_id                  | INTEGER
   8  | summary                    | TEXT
   9  | type                       | TEXT
  10  | contributor                | TEXT
  11  | web_link                   | TEXT
  12  | overall_design             | TEXT
  13  | repeats                    | TEXT
  14  | repeats_sample_list        | TEXT
  15  | variable                   | TEXT
  16  | variable_description       | TEXT
  17  | contact                    | TEXT
  18  | supplementary_file         | TEXT
------+----------------------------+-----------
```

Sometimes it is useful to get the actual SQL schema associated with a table. As an example of doing this and using an SQLite PRAGMA function, we can get the table schema for the *gpl* table.

```
>> SQLiteQuery( 'GEOmetadb.sqlite', 'PRAGMA TABLE_INFO(gpl);' )
ans =
    [ 0]    'ID'                    'REAL'    [0]    [NaN]    [0]
    [ 1]    'title'                 'TEXT'    [0]    [NaN]    [0]
    [ 2]    'gpl'                   'TEXT'    [0]    [NaN]    [0]
    [ 3]    'status'                'TEXT'    [0]    [NaN]    [0]
    [ 4]    'submission_date'       'TEXT'    [0]    [NaN]    [0]
    [ 5]    'last_update_date'      'TEXT'    [0]    [NaN]    [0]
```

```
[ 6]    'technology'            'TEXT'    [0]    [NaN]    [0]
[ 7]    'distribution'          'TEXT'    [0]    [NaN]    [0]
[ 8]    'organism'              'TEXT'    [0]    [NaN]    [0]
[ 9]    'manufacturer'          'TEXT'    [0]    [NaN]    [0]
[10]             [1x20 char]    'TEXT'    [0]    [NaN]    [0]
[11]    'coating'               'TEXT'    [0]    [NaN]    [0]
[12]    'catalog_number'        'TEXT'    [0]    [NaN]    [0]
[13]    'support'               'TEXT'    [0]    [NaN]    [0]
[14]    'description'           'TEXT'    [0]    [NaN]    [0]
[15]    'web_link'              'TEXT'    [0]    [NaN]    [0]
[16]    'contact'               'TEXT'    [0]    [NaN]    [0]
[17]    'data_row_count'        'REAL'    [0]    [NaN]    [0]
[18]    'supplementary_file'    'TEXT'    [0]    [NaN]    [0]
[19]    'bioc_package'          'TEXT'    [0]    [NaN]    [0]
```

3.2 Writing SQL queries and getting results

Select 5 records from the *gse* table and show the first 7 columns.

```
>> rs = SQLiteQuery( 'GEOmetadb.sqlite', 'SELECT * FROM gse LIMIT 5;' );
>> format long
>> rs(:,1:7)
ans =
  Columns 1 through 2
    [1]    'NHGRI_Melanoma_class'
    [2]    'Cerebellar development'
    [3]    'Renal Cell Carcinoma Differential Expression'
    [4]    'Diurnal and Circadian-Regulated Genes in Arabidopsis'
    [5]    'Global profile of germline gene expression in C. elegans'
  Columns 3 through 6
    'GSE1'    'Public on Jan 22 2001'    '2001-01-22'    '2005-05-29'
    'GSE2'    'Public on Apr 26 2001'    '2001-04-19'    '2005-05-29'
    'GSE3'    'Public on Jul 19 2001'    '2001-07-19'    '2005-05-29'
    'GSE4'    'Public on Jul 20 2001'    '2001-07-20'    '2005-05-29'
    'GSE5'    'Public on Jul 24 2001'    '2001-07-24'    '2005-07-18'
  Column 7
    [10952317]
    [     NaN]
    [11691851]
    [11158533]
    [11030340]
```

Get the GEO series accession and title from GEO series that were submitted by "Sean Davis". The "%" sign is used in combination with the "like" operator to do a "wildcard" search for the name "Sean Davis" with any number of characters before or after or between "Sean" and "Davis".

```
>> sqlcmd = 'SELECT gse,title FROM gse WHERE contributor like ''%Sean%Davis%'';';
>> rs = SQLiteQuery('GEOmetadb.sqlite', sqlcmd )
rs =
    'GSE2553'    'NHGRI_Sarcoma_Baird'
    'GSE4406'    'Gene expression profiling of CD4+ T-cells and GM6990 lymphoblastoid cell lines
```

```
    'GSE5357'    'NHGRI Menin ChIP-Chip'
    'GSE7376'    'Detection of novel amplification units in prostate cancer'
    'GSE8486'    'Whole genome DNAse hypersensitivity in human CD4+ T-cells'
```

As another example, GEOmetadb can find all samples on GPL96 (Affymetrix hgu133a) that have .CEL files available for download.

```
>> sqlcmd = ['SELECT gsm, supplementary_file FROM gsm '];
>> sqlcmd = [sqlcmd 'WHERE gpl=''GPL96'' and supplementary_file like ''%CEL.gz'';'];
>> rs = SQLiteQuery('GEOmetadb.sqlite', sqlcmd );
>> size( rs )
    ans =
        8342           2
```

But why limit to only GPL96? Why not look for all Affymetrix arrays that have .CEL files? And list those with their associated GPL information, as well as the Bioconductor annotation package name?

```
>> sqlcmd = ['SELECT gpl.bioc_package,gsm.gpl,gsm,gsm.supplementary_file '];
>> sqlcmd = [sqlcmd 'FROM gsm JOIN gpl ON gsm.gpl=gpl.gpl '];
>> sqlcmd = [sqlcmd 'WHERE gpl.manufacturer=''Affymetrix'' ' ];
>> sqlcmd = [sqlcmd 'AND gsm.supplementary_file like ''%CEL.gz'';'];
>> rs = SQLiteQuery('GEOmetadb.sqlite', sqlcmd );
>> rs(1:5,:)
  Columns 1 through 3
    'hu6800'    'GPL80'    'GSM575'
    'hu6800'    'GPL80'    'GSM576'
    'hu6800'    'GPL80'    'GSM577'
    'hu6800'    'GPL80'    'GSM578'
    'hu6800'    'GPL80'    'GSM579'
  Column 4
    'ftp://ftp.ncbi.nlm.nih.gov/pub/geo/DATA/supplementary/samples/GSMnnn/GSM575/GSM575.cel.gz'
    'ftp://ftp.ncbi.nlm.nih.gov/pub/geo/DATA/supplementary/samples/GSMnnn/GSM576/GSM576.cel.gz'
    'ftp://ftp.ncbi.nlm.nih.gov/pub/geo/DATA/supplementary/samples/GSMnnn/GSM577/GSM577.cel.gz'
    'ftp://ftp.ncbi.nlm.nih.gov/pub/geo/DATA/supplementary/samples/GSMnnn/GSM578/GSM578.cel.gz'
    'ftp://ftp.ncbi.nlm.nih.gov/pub/geo/DATA/supplementary/samples/GSMnnn/GSM579/GSM579.cel.gz'
```

Of course, we can combine programming and data access. A simple loop shows how to query each of the tables for number of records. Note that here we request table names from function `GetSQLiteTables()`, but instead of printing all names out, we store it in a cell-array `geoTables`.

```
>> geoTables = GetSQLiteTables( 'GEOmetadb.sqlite' );
>> for i = 1:length( geoTables )
>>     sqlcmd = ['SELECT count(*) FROM ' geoTables{i} ';'];
>>     count(i) = SQLiteQuery( 'GEOmetadb.sqlite', sqlcmd );
>>     fprintf( 1, '%20s | %d \n', geoTables{i}, count );
>> end

                 gds | 2085
          gds_subset | 12225
```

```
       geoConvert | 971946
geodb_column_desc | 104
              gpl | 4505
              gse | 8301
          gse_gpl | 11410
          gse_gsm | 235207
              gsm | 214340
         metaInfo | 2
          sMatrix | 10423
```

3.3 All about `SQLiteQuery()` function

`SQLiteQuery` is an essential function for accessing SQLite database. We have implemented a simple yet powerful enough query for MEX function such that MATLAB Database Toolbox is not required. As we stated before, this implementation is not intend for complicated SQL commands, rather, it serves a purpose for demonstrating the capability of GEOmetadb.sqlite and beyond, as we will continue this user guide in later section. Again, for each SQL query, we connect to database, execute SQL command, and then close the connection immediately. In case you more information than the examples showed in last section, here are three variants of the same function call.

```
SQLiteQuery( dbName, sqlCommand )
rs = SQLiteQuery( dbName, sqlCommand );
[rs, cNames] = SQLiteQuery( dbName, sqlCommand );
[rs, cNames, cTypes] = SQLiteQuery( dbName, sqlCommand );
```

The first example will display the query result directly to the screen. The second example will store the query result into variable `rs`. The third example will return column names into `cNames`, and the forth example will also return the data type for each column into `cTypes`. Both cNames and cTypes are cell arrays with text strings. The contents of cNames depend on the SQL command, while cTypes are related to the coresponding column. Valid strings for cTypes can be found in "Interacting with the database" section, such as in the sample of `SQLiteQuery( 'GEOmetadb.sqlite', 'PRAGMA TABLE_INFO(gpl);' )` (third column in query result). Please note that the length of the `dbName` of the SQLite database can not be longer than 256 characters, the length of `sqlCommand` can not be longer than 5000 characters, and the sqlCommand has to be terminated with ';', as you may find throughout this user guide. Here we provide one quick illustration,

```
>> [rs, cNames, cTypes] = SQLiteQuery( 'GEOmetadb.sqlite', 'SELECT * FROM gse LIMIT 5;' );
>> cNames'
ans =
    'ID'
    'title'
    'gse'
    'status'
    'submission_date'
    'last_update_date'
```

```
    'pubmed_id'
    'summary'
    'type'
    'contributor'
    'web_link'
    'overall_design'
    'repeats'
    'repeats_sample_list'
    'variable'
    'variable_description'
    'contact'
    'supplementary_file'
>> cType
  Columns 1 through 8
     'REAL'    'TEXT'    'TEXT'    'TEXT'    'TEXT'    'TEXT'    'INTEGER'    'TEXT'
  Columns 9 through 16
     'TEXT'    'TEXT'    'TEXT'    'TEXT'    'TEXT'    'TEXT'    'TEXT'    'TEXT'
  Columns 17 through 18
     'TEXT'    'TEXT'
```

These information may be useful for further data processing.

3.4 Conversion of GEO entity types

Large-scale consumers of GEO data might want to convert GEO entity type from one to others, e.g. finding all GSM and GSE associated with 'GPL96'. Function `GEOconvert()` does the conversion with a very fast mapping between entity types. Example of converting 'GPL96' to other possible GEO types in the GEOmetadb.sqlite is provided below,

```
>> outGEOacc = GEOconvert( 'GEOmetadb.sqlite', 'GPL96' );
```

Check what GEO types and how many entities in each type in the conversion. Conversion results are saved into MATLAB structure with appropriate conversion type, and the nx2 cell-array *acc* stores *from_acc* in first column and *to_acc* in second column.

```
>> outGEOacc
outGEOacc =
        gsm: [1x1 struct]
        gse: [1x1 struct]
        gds: [1x1 struct]
    smatrix: [1x1 struct]
>> outGEOacc.gse
ans =
     acc: {532x2 cell}
```

The `GEOconvert` connects all GSM, GSE, GDS, and sMatrix to 'GPL96'. As shown above, there are total of 532 GEO Series that utilize GPL96 platform. Examine first 5 GSE, GDS, and seriesMatrix accession numbers with GPL96.

```
>> outGEOacc.gse.acc(1:5,:)
ans =
    'GPL96'    'GSE1000'
```

```
    'GPL96'    'GSE10024'
    'GPL96'    'GSE10043'
    'GPL96'    'GSE10072'
    'GPL96'    'GSE10089'
>> outGEOacc.gds.acc(1:5,:)
ans =
    'GPL96'    'GDS1023'
    'GPL96'    'GDS1036'
    'GPL96'    'GDS1041'
    'GPL96'    'GDS1050'
    'GPL96'    'GDS1062'
>> outGEOacc.smatrix.acc(1:5,:)
 ans =
    'GPL96'    'GSE1000_series_matrix.txt.gz'
    'GPL96'    'GSE10024_series_matrix.txt.gz'
    'GPL96'    'GSE10043_series_matrix.txt.gz'
    'GPL96'    'GSE10072_series_matrix.txt.gz'
    'GPL96'    'GSE10089_series_matrix.txt.gz'
```

3.5 More advanced queries

Now, for something a bit more complicated, we would like to find all the human breast cancer-related Affymetrix gene expression GEO series.

```
>> sqlSelect = ['SELECT DISTINCT gse.title, gse.gse '];
>> sqlFrom   = ['FROM gsm JOIN gse_gsm ON gsm.gsm=gse_gsm.gsm '];
>> sqlFrom   = [sqlFrom 'JOIN gse ON gse_gsm.gse=gse.gse '];
>> sqlFrom   = [sqlFrom 'JOIN gse_gpl ON gse_gpl.gse=gse.gse '];
>> sqlFrom   = [sqlFrom 'JOIN gpl ON gse_gpl.gpl=gpl.gpl '];
>> sqlWhere  = ['WHERE gsm.molecule_ch1 like ''%total RNA%'' '];
>> sqlWhere  = [sqlWhere 'AND gse.title LIKE ''%breast cancer%'' '];
>> sqlWhere  = [sqlWhere 'AND gpl.organism LIKE ''%Homo sapiens%'';'];
>> rs = SQLiteQuery('GEOmetadb.sqlite', [sqlSelect sqlFrom sqlWhere] );
>> rs(1:5,:)
ans =
    [1x118 char]    'GSE2294'
    [1x116 char]    'GSE8465'
    [1x97  char]    'GSE9893'
    [1x63  char]    'GSE1864'
    [1x92  char]    'GSE4000'
```

Perhaps a little programming will provide more compact display of the result. Remeber, the *rs* is a cell-array, so use curly bracket to access the content in each cell.

```
>> for i = 1:5, fprintf( 1, '%50s | %s\n', rs{i,1}(1:50), rs{i,2} ); end
A Modular Analysis of Breast Cancer Reveals a Nove | GSE2294
A Phase II Study of Neoadjuvant Gemcitabine Plus D | GSE8465
A gene expression signature predicting the recurre | GSE9893
A genomic view of estrogen actions in human breast | GSE1864
A molecular 'signature' of primary breast cancer c | GSE4000
```

Again, you do not need to disconnection from database if you use the functions provided by *GEOTools*, since all connections are immediately closed after querying.

If you want to remove old GEOmetadb.sqlite file before retrieve a new version from the server (this step is not necessary, since the function `GEOgetSQLiteFile()` rename the old copy of *GEOmetadb.sqlite* to a backup version, and then download the updated database), execute the following codes:

```
>> delete( 'GEOmetadb.sqlite' );
```

or, just go to the directory and delete the file name *GEOmetadb.sqlite*, and possibly *GEOmetadb.sqlite.backup*.

4 Applications of *GEOTools* for data analysis

Let's assume that we would like to explore the inflammatory breast cancer studies in GEO database. One of the quickest search is,

```
>> [rs, cNames] = SQLiteQuery( 'GEOmetadb.sqlite', ...
          'SELECT * FROM gse WHERE summary like ''%inflammatory%breast%cancer%'';');
>> [cNames' rs']
ans =
    'ID'                    [           1354]    [      4901]
    'title'                      [1x26  char]    [1x35  char]
    'gse'                   'GSE1561'            'GSE5847'
    'status'                     [1x21  char]    [1x21  char]
    'submission_date'       '2004-07-14'         '2006-09-15'
    'last_update_date'      '2005-05-29'         '2008-01-24'
    'pubmed_id'             [       15897907]    [  17999412]
    'summary'                    [1x744 char]    [1x236 char]
    'type'                  'parallel sample'    'LCM'
    'contributor'           'Richard,D,Iggo'     [1x42  char]
    'web_link'                             ''               ''
    'overall_design'                       ''    [1x204 char]
    'repeats'                              ''               ''
    'repeats_sample_list'                  ''               ''
    'variable'                             ''               ''
              [1x20 char]                  ''               ''
    'contact'                    [1x279 char]    [1x157 char]
    'supplementary_file'         [1x84  char]    [1x84  char]
```

2 data sets were deposited to GEO as we find. And we can quickly examine some key features: associated array platform, number of samples, etc, by doing followings,

```
>> out = GEOconvert( 'GEOmetadb.sqliite', rs(:,3) );
>> out.gpl.acc
ans =
    'GSE1561'    'GPL96'
    'GSE5847'    'GPL96'
>> length( strmatch( rs{1,3}, out.gsm.acc(:,1) ) )
ans =
    49
```

```
>> length( strmatch( rs{2,3}, out.gsm.acc(:,1) ) )
ans =
    95
```

Apparently, both are GPL96, which is a Affymetrix GeneChip Human Genome U133 Array Set HG-U133A, which you can simply use the command

```
>> rs = SQLiteQuery( 'GEOmetadb.sqlite', 'SELECT * FROM gpl WHERE gpl=''GPL96'';' );
```

to examine the details of this array platform. The GSE1561 has 49 samples, and GSE5847 has 95 samples. Suppose we would like to take the latest deposition to GEO, GSE5847 for analysis, there are two choices to proceed: 1) use seriesMatrix, or 2) download every attachment and process accordingly. Here we present the steps by using seriesMatrix. The second choice envolving downloading individual *.cel* file and then using standard **MATLAB bioinformatics Toolbox** functions. For reading seriesMatrix, we have provided some functions for quick data access,

```
>> data = GEOReadsMatrix( 'GSE5847' );
Checking this URL = ftp://ftp.ncbi.nih.gov/pub/geo/DATA/SeriesMatrix/GSE5847/GSE5847_series_matr
Total of 95 columns (samples)...
Get GPL annotation...
Checking this URL = ftp://ftp.ncbi.nih.gov/pub/geo/DATA/annotation/platforms/GPL96.annot.gz, dow
Map IDs ...
```

What do you get from `GEOReadsMatrix` function? Here is a quick look at all elements of *data*,

```
>> data
data =
        Series: [1x1 struct]
        Samples: [1x95 struct]
        refID: {22283x1 cell}
        value: [22283x95 double]
        gplAnnot: [1x1 struct]
        idmap: [22283x1 double]
        probeAnnot: [1x1 struct]
```

Obviously, it contains all information (and in the right order in terms of probe annotation) necessary for further analysis: sample information is stored in *data.Samples*, and gene annotation is stored in *data.probeAnnot*, and data matrix is stored in *data.value*. *data.gplAnnot* is the annotation file associated with the GPL, but the annotation may not be in the same order required for data matrix. We keep *data.gplAnnot* in the structure because it is the original (and may be more or less number of annotation required for data matrix). The annotation downloaded from GEO contains (which follow exactly the same order and content of GEO platform annotation),

```
>> data.probeAnnot
ans =
                    Platform: [1x1 struct]
                          ID: {22283x1 cell}
                 Gene_title: {22283x1 cell}
               Gene_symbol: {22283x1 cell}
                   Gene_ID: {22283x1 cell}
```

```
                    UniGene_title: {22283x1 cell}
                   UniGene_symbol: {22283x1 cell}
                       UniGene_ID: {22283x1 cell}
                  Nucleotide_Title: {22283x1 cell}
                                GI: {22283x1 cell}
                 GenBank_Accession: {22283x1 cell}
                  Platform_CLONEID: {22283x1 cell}
                      Platform_ORF: {22283x1 cell}
                   Platform_SPOTID: {22283x1 cell}
              Chromosome_location: {22283x1 cell}
            Chromosome_annotation: {22283x1 cell}
      Gene_Ontology_Function_term: {22283x1 cell}
       Gene_Ontology_Process_term: {22283x1 cell}
     Gene_Ontology_Component_term: {22283x1 cell}
Gene_Ontology_Function_identifier: {22283x1 cell}
 Gene_Ontology_Process_identifier: {22283x1 cell}
Gene_Ontology_Component_identifier: {22283x1 cell}
```

The following lines will display first 47 sample information (there are total of 95 samples).

```
>> for i = 1:47
  fprintf( 1, '%d -> %s, %s\n', i, data.Samples(i).source_name_ch1{1}, ...
   data.Samples(1).characteristics_ch1{1} );
   end
>>
1 -> human breast cancer stroma, IBC
2 -> human breast cancer stroma, IBC
3 -> human breast cancer stroma, IBC
4 -> human breast cancer stroma, IBC
5 -> human breast cancer stroma, IBC
6 -> human breast cancer stroma, IBC
7 -> human breast cancer stroma, IBC
8 -> human breast cancer stroma, IBC
9 -> human breast cancer stroma, IBC
10 -> human breast cancer stroma, IBC
11 -> human breast cancer stroma, IBC
12 -> human breast cancer stroma, IBC
13 -> human breast cancer stroma, IBC
14 -> human breast cancer stroma, non-IBC
15 -> human breast cancer stroma, non-IBC
16 -> human breast cancer stroma, non-IBC
17 -> human breast cancer stroma, non-IBC
18 -> human breast cancer stroma, non-IBC
19 -> human breast cancer stroma, non-IBC
20 -> human breast cancer stroma, non-IBC
21 -> human breast cancer stroma, non-IBC
22 -> human breast cancer stroma, non-IBC
23 -> human breast cancer stroma, non-IBC
24 -> human breast cancer stroma, non-IBC
25 -> human breast cancer stroma, non-IBC
26 -> human breast cancer stroma, non-IBC
27 -> human breast cancer stroma, non-IBC
```

```
28 -> human breast cancer stroma, non-IBC
29 -> human breast cancer stroma, non-IBC
30 -> human breast cancer stroma, non-IBC
31 -> human breast cancer stroma, non-IBC
32 -> human breast cancer stroma, non-IBC
33 -> human breast cancer stroma, non-IBC
34 -> human breast cancer stroma, non-IBC
35 -> human breast cancer stroma, non-IBC
36 -> human breast cancer stroma, non-IBC
37 -> human breast cancer stroma, non-IBC
38 -> human breast cancer stroma, non-IBC
39 -> human breast cancer stroma, non-IBC
40 -> human breast cancer stroma, non-IBC
41 -> human breast cancer stroma, non-IBC
42 -> human breast cancer stroma, non-IBC
43 -> human breast cancer stroma, non-IBC
44 -> human breast cancer stroma, non-IBC
45 -> human breast cancer stroma, non-IBC
46 -> human breast cancer stroma, non-IBC
47 -> human breast cancer stroma, non-IBC
```

As we can see there are 13 non-IBC and 34 IBC samples from stroma cells. To find significantly differentially expressed genes in stroma cells between IBC and non-IBC, we have,

```
>> logValue = log2( data.value );
>> p = mattest( logValue(:,1:13), logValue(:,14:47) );
>> idx = find( p < 0.001 );
```

where p is the p-value obtained from t-test (mattest), and idx is the indices to the p-value vector where p value is less than 0.001. We directly selecting genes at $p < 0.001$, we have about 127 genes. In practice, one may choose to use different false-discovery rate (FDR) to control the false positive. However, since it is not our intention to explain the analysis method in this manual, we only provide some examples for FDR based gene selection without elaboration.

```
>> fdr = mafdr( p ); % this is the FDR control by Storey (2002)
>> fdr = mafdr( p,'BHFDR', 'true');  % this is the Benjamini-Hochberg.
```

In following examples, we use 127 genes selected by simple p-value less than 0.001. To view dendrogram/clusters of gene expression, we have (and result is shown in figure 2),

```
>> zValue = zscore(logValue(idx,1:47)')';
>> geneLabels = data.probeAnnot.Gene_symbol(idx);
>> clustergram( zValue, 'DIMENSION', 2, 'ROWLABELS', geneLabels );
```

One may notice that there are 48 IBC samples derived from epithelium cells. One of the natural questions is whether this 127 differential expressed genes from IBC troma cells can be used as signature gene set to correctly classify IBC epithelium cells. Here we provide a quick kNN classification approach just to demonstrate the utility of the GEOTools,

```
>> groups = [repmat( {'IBC'}, 1, 13) repmat( {'non-IBC'}, 1, 34 )];
>> class = knnclassify( logValue(idx, 48:95)', logValue(idx,1:47)', groups );
```

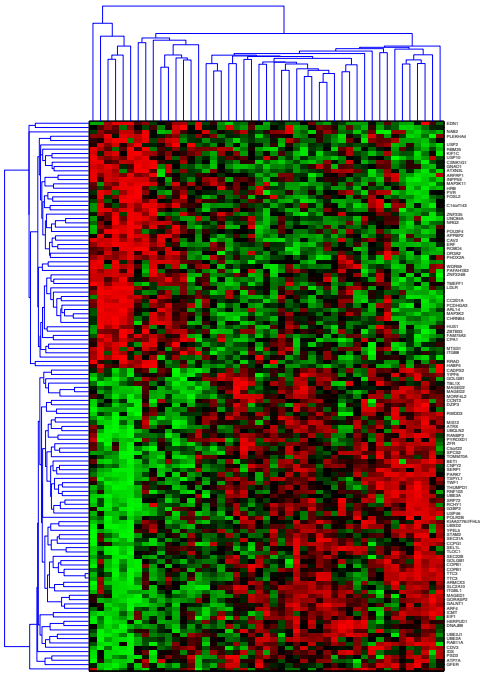

Figure 2: Clustergram from 127 selected genes and 47 samples.

There are some particular MATLAB coding details here: the first line set up the training group assignment (first 13 is IBC, and rest is non-IBC). The second line does the KNN classification (since knnclassify performs on row, while most microarray data store sample data in columns, therefore we have to transpose the matrix, or in MATLAB, simply a after matrix. Default KNN algorithm chooses $k = 1$). The result is

```
>> for i = 1:47
    fprintf( 1, '%9s -> %s\\n', data.Samples(i+47).characteristics\_ch1{1}, class{i} );
end
      IBC -> IBC
      IBC -> IBC
      IBC -> IBC
      IBC -> IBC
      IBC -> IBC
      IBC -> IBC
      IBC -> IBC
      IBC -> IBC
      IBC -> non-IBC
      IBC -> non-IBC
      IBC -> IBC
      IBC -> non-IBC
      IBC -> IBC
  non-IBC -> non-IBC
  non-IBC -> non-IBC
  non-IBC -> non-IBC
  non-IBC -> non-IBC
```

```
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
non-IBC -> non-IBC
```

Notice that first 13 samples from epithelium cells are also IBC, the classification result shows first 13 samples to be 10 IBC and 3 non-IBC (wrong classification results). Overall, there are 4 mistakes in the assignment, which yields classification accuracy to be (48-4)/48, or about 92%. Note that to fully implement classification scheme, one needs to add cross-validation into the algorithm, which is another subject beyond this user manual.