bioassayR: small molecule bioactivity analysis

Tyler Backman, Thomas Girke
Email contact: thomas.girke@ucr.edu

June 10, 2015

1 Introduction

bioassayR is a flexible computational tool for statistical analysis of biological screening data. It allows users to store, organize, and systematically analyze data from a large number of small molecule bioactivity experiments. Users have the option of supplying their own bioactivity data for analysis, or downloading a database from the authors website (`http://chemmine.ucr.edu/bioassayr`) pre-loaded with bioactivity data sourced from NCBI PubChem Bioassay[1, 2]. The pre-loaded database contains the results of hundreds of thousands of bioassay experiments, where small molecules were screened against a defined biological target. *bioassayR* allows users to powerfully leverage these data as a reference to identify small molecules active against a protein or organism of interest, identify target selective compounds that may be useful as drugs or chemical genomics probes, and identify and compare the activity profiles of small molecules.

Contents

2 Recently Added Features

- added `inactiveTargets` which finds targets a compound is inactive against

- added `targetSelectivity` which computes target selectivity for a list of compounds
- added `screenedAtLeast` which returns a list of CIDs screened at least X times
- added `loadIdMapping` which stores protein identifier mappings
- added `translateTargetId` which retrieves stored protein identifier mappings

3 Getting Started

3.1 Installation

The R software for running bioassayR can be downloaded from CRAN (http://cran.at.r-project.org/). The *bioassayR* package can be installed from R using the `bioLite` install command.

```
> source("http://bioconductor.org/biocLite.R") # Sources the biocLite.R installation script.
> biocLite("bioassayR") # Installs the package.
```

3.2 Loading the Package and Documentation

```
> library(bioassayR) # Loads the package
```

```
> library(help="bioassayR") # Lists all functions and classes
> vignette("bioassayR") # Opens this PDF manual from R
```

3.3 Quick Tutorial

This example walks you through creating a new empty database, adding example small molecule bioactivity data, and performing queries on these data.

This example includes real experimental data from an antibiotics discovery experiment. These data are a "confirmatory bioassay" where 57 small molecules were screened against the mevalonate kinase protein from the *Streptococcus pneumonia* (SP) bacteria. Mevalonate kinase inhibitors are one possible class of antibiotic drugs that may be effective against this infamous bacteria. These data were published as assay identifier (aid) 1000 in the NCBI PubChem Bioassay database, by Dr. Thomas S. Leyh.

First, create a new database. For purposes of this manual a temporary file is used, but you can replace the `tempfile` function call with a filename of your choice if you wish to save the resulting database for later.

```
> library(bioassayR)
> library(RSQLite)
> myDatabaseFilename <- tempfile()
> mydb <- newBioassayDB(myDatabaseFilename, indexed=F)
```

Next, specify the source and version of the data you plan to load. This is a required step, which makes it easier to track the origin of your data later. Feel free to use the current date for a version number, if your data isn't versioned.

```
> addDataSource(mydb, description="PubChem Bioassay", version="unknown")
```

After adding a data source, create or import a *data.frame* which contains the activity scores for each of the molecules in your assay. This *data.frame* must contain four columns which includes a cid (unique compound identifier) for each compound, an sid (often used to distinguish distinct samples of the same compound structure), a binary activity score ($1 = active$, $0 = inactive$), and a numeric activity score. Consult the *bioassay* man page for more details on formatting this *data.frame*. The *bioassayR* package contains an example activity score data frame that can be accessed as follows:

```
> data(samplebioassay)
> samplebioassay[1:10,] # print the first 10 scores
```

```
        cid      sid activity score
1    730195 26736081        0     0
2  16749973 26736082        1    80
3  16749974 26736083        1    80
4  16749975 26736084        1    80
5  16749976 26736085        1    80
6  16749977 26736086        1    80
7  16749978 26736087        1    80
8  16749979 26736088        1    80
9  16749980 26736089        1    80
10 16749981 26736090        1    80
```

All bioactivity data is loaded into the database, or retrieved from the database as an *bioassay* object which contains details on the assay experimental design, molecular targets, and the activity scores. A bioassay object which incorporates activity scores can be created as follows. The source id value must exactly match that loaded earlier by `addDataSource`. The molecular target(s) for the assay are optional, and an unlimited number can be specified for a single assay as a vector passed to the targets option. The target types field should be a vector of equal length, describing the type of each target in the same order.

```
> myAssay <- new("bioassay",aid="1000", source_id="PubChem Bioassay",
+     assay_type="confirmatory", organism="unknown", scoring="activity rank",
+     targets="116516899", target_types="protein", scores=samplebioassay)
> myAssay
```

```
class:                 bioassay
aid:                   1000
source_id:           PubChem Bioassay
assay_type:           confirmatory
organism:           unknown
scoring:           activity rank
targets:           116516899
target_types:           protein
total scores:           57
```

The `bioassay` object can be loaded into the database with the `loadBioassay` function. By repeating this step with different data, a large number of distinct assays can be loaded into the database.

```
> loadBioassay(mydb, myAssay)
```

Wait a minute! We accidentally labeled that assay as organism "unknown" when we know that it's actually a screen against a protein from *Streptococcus pneumonia*. After loading an assay into the database, you can later retrieve these data with the `getAssay` function. By combining this with the ability to delete an assay (the `dropBioassay` function) one can edit the database by (1) pulling an assay out, (2) deleting it from the database, (3) modifying the pulled out object, and (4) reloading the assay. For example, we can update the species annotation for our assay as follows:

```
> tempAssay <- getAssay(mydb, "1000") # get assay from database
> dropBioassay(mydb, "1000") # delete assay from database
> organism(tempAssay) <- "Streptococcus pneumonia" # update organism
> loadBioassay(mydb, tempAssay)
```

It is recommended to index your database after loading all of your data. This significantly speeds up access to the database, but can also slow down loading of data if indexing is performed before loading.

```
> addBioassayIndex(mydb)
```

After indexing, you can query the database. Here are some example queries. First view the database summary provided by *bioassayR*:

```
> mydb
```

```
class:                  BioassayDB
assays:                  1
sources:          PubChem Bioassay
writeable:         yes
```

Next, you can query the database for active targets for a given compound by cid. In this case, since only one assay has been loaded only a single target can be found. Experiment with loading more assays for a more interesting result!

```
> activeTargets(mydb, 16749979)

          fraction_active total_screens
116516899        26736088             1
```

While many pre-built queries are provided (see other examples and man pages) advanced users can also build their own SQL queries. First you will want to see the structure of the database as follows:

```
> queryBioassayDB(mydb, "SELECT * FROM sqlite_master WHERE type='table'")

   type               name           tbl_name rootpage
1 table           activity           activity        2
2 table             assays             assays        3
3 table            domains            domains        4
4 table            sources            sources        5
5 table            targets            targets        6
6 table targetTranslations targetTranslations        7
                                                                                     sql
1                  CREATE TABLE activity (aid INTEGER, cid INTEGER, activity INTEGER, score INTEGER)
2 CREATE TABLE assays (source_id INTEGER, aid INTEGER, assay_type TEXT, organism TEXT, scoring TEXT)
3                                                  CREATE TABLE domains (domain TEXT, target INTEGER)
4           CREATE TABLE sources (source_id INTEGER PRIMARY KEY ASC, description TEXT, version TEXT)
5                                CREATE TABLE targets (aid INTEGER, target TEXT, target_type TEXT)
6                      CREATE TABLE targetTranslations (target TEXT, category TEXT, identifier TEXT)
```

As this is a SQLite database, you can consult http://www.sqlite.org for specifics on building SQL queries. For example, you can find all assays a given compound has participated in as follows:

```
> queryBioassayDB(mydb, "SELECT DISTINCT(aid) FROM activity WHERE cid = '16749979'")

   aid
1 1000
```

This example prints the first 10 activity scores from a specified assay:

```
> queryBioassayDB(mydb, "SELECT * FROM activity WHERE aid = '1000' LIMIT 10")

    aid      cid activity score
1  1000   730195 26736081     0
2  1000 16749973 26736082     1
3  1000 16749974 26736083     1
4  1000 16749975 26736084     1
5  1000 16749976 26736085     1
6  1000 16749977 26736086     1
7  1000 16749978 26736087     1
8  1000 16749979 26736088     1
9  1000 16749980 26736089     1
10 1000 16749981 26736090     1
```

Lastly, disconnecting from the database after analysis reduces the chances of data corruption. If you are using a pre-built database in read only mode (as demonstrated in the Prebuilt Database Example section) you can optionally skip this step.

```
> disconnectBioassayDB(mydb)
```

4 Loading User Supplied Data

This section demonstrates the process for creating a new bioactivity database from user supplied data. As an example, we will demonstrate the process of downloading an assay from the NCBI PubChem Bioassay bioactivity data repository, and loading this into a new database[2].

First, get two files from Pubchem Bioassay for the assay of interest: an XML file containing details on how the experiment was performed, and a CSV (comma separated value) file which contains the actual activity scores. For the purposes of this example, we will use the data from assay 1000, which is a confirmatory assay (titration assay) of 57 small molecules against a mevalonate kinase protein. More details on this assay were provided in the "Quick Tutorial," where the same data is used. These files can be downloaded from PubChem Bioassay at http://pubchem.ncbi.nlm.nih.gov/ or loaded from the example data repository included in this package as follows:

```
> library(bioassayR)
> extdata_dir <- system.file("extdata", package="bioassayR")
> assayDescriptionFile <- file.path(extdata_dir, "exampleAssay.xml")
> activityScoresFile <- file.path(extdata_dir, "exampleScores.csv")
```

Next, create a new empty database for loading these data into. This example uses the R `tempfile` function to create the database in a random location. If you would like to keep your resulting database, replace `myDatabaseFilename` with your desired path and filename.

```
> myDatabaseFilename <- tempfile()
> mydb <- newBioassayDB(myDatabaseFilename, indexed=F)
```

We will also add a data source to this database, specifying that our data here mirrors an assay provided by PubChem Bioassay.

```
> addDataSource(mydb, description="PubChem Bioassay", version="unknown")
```

The XML file provided by PubChem Bioassay contains extensive details on how the assay was performed, molecular targets, and results scoring methods. You can extract these using the `parsePubChemBioassay` function as follows. The `parsePubChemBioassay` function also requires a .csv file which contains the activity scores for each assay, in the standard CSV format provided by PubChem Bioassay. For data from sources other than PubChem Bioassay, you may need to write your own code to parse out the assay details- or type them in manually.

```
> myAssay <- parsePubChemBioassay("1000", activityScoresFile, assayDescriptionFile)
> myAssay

class:                 bioassay
aid:                   1000
source_id:             PubChem Bioassay
assay_type:            confirmatory
organism:              NA
scoring:               IC50
targets:               116516899
target_types:          protein
total scores:          57
```

Next, load the resulting data parsed from the XML and CSV files into the database. This creates records in the database for both the assay itself, and it's molecular targets.

```
> loadBioassay(mydb, myAssay)
```

To load additional assays, repeat the above steps. After all data is loaded, you can significantly improve subsequent query performance by adding an index to the database.

```
> addBioassayIndex(mydb)
```

After indexing, perform a test query on your database to confirm that the data loaded correctly.

```
> activeAgainst(mydb,"116516899")
```

	fraction_active	total_assays
16749973	1	1
16749974	1	1
16749975	1	1
16749976	1	1
16749977	1	1
16749978	1	1
16749979	1	1
16749980	1	1
16749981	1	1
16749982	1	1
16749983	1	1
16749984	1	1
16749985	1	1
16749986	1	1
16749987	1	1
16749988	1	1
16749989	1	1
16749990	1	1
16749991	1	1
16749992	1	1
16749993	1	1
16749994	1	1
16749995	1	1
16749996	1	1
16749997	1	1
16749998	1	1
16749999	1	1
16750000	1	1
16750001	1	1
16750002	1	1
16750003	1	1
16750004	1	1
16750005	1	1
16750006	1	1
16750007	1	1
16750016	1	1

Lastly, disconnect from the database to prevent data corruption.

```
> disconnectBioassayDB(mydb)
```

5 Prebuilt Database Example: Investigate Activity of a Known Drug

A pre-built database containing large quantities of public domain bioactivity data sourced from the PubChem Bioassay database, can be downloaded from http://chemmine.ucr.edu/bioassayr. While downloading the full database is recommended, it is possible to run this example using a small subset of the database, included within the *bioassayR* package for testing purposes. This example demonstrates the utility of *bioassayR* for identifying the bioactivity patterns

of a small drug-like molecule. In this example, we look at the binding activity patterns for the drug acetylsalicylic acid (aka Aspirin) and compare these binding data to annotated targets in the DrugBank.ca drug database[3].

The DrugBank database is a valuable resource containing numerous data on drug activity in humans, including known molecular targets. In this exercise, first take a look at the annotated molecular targets for acetylsalicylic acid by searching this name on http://drugbank.ca. This will provide a point of reference for comparing to the bioactivity data we find in the prebuild PubChem Bioassay database. Note that DrugBank also contains the PubChem CID of this compound, which you can use to query the bioassayR PubChem Bioassay database.

To get started first connect to the database. The variable `sampleDatabasePath` can be replaced with the filename of the full test database you downloaded, if you would like to use that instead of the small example version included with this software package.

```
> library(bioassayR)
> extdata_dir <- system.file("extdata", package="bioassayR")
> sampleDatabasePath <- file.path(extdata_dir, "sampleDatabase.sqlite")
> pubChemDatabase <- connectBioassayDB(sampleDatabasePath)
```

Next, use the `activeTargets` function to find all protein targets which acetylsalicylic acid shows activity against in the loaded database. These target IDs are NCBI Protein identifiers as provided in PubChem Bioassay. In which cases do these results agree with or disagree with the annotated targets from DrugBank?

```
> drugTargets <- activeTargets(pubChemDatabase, "2244")
> drugTargets
```

```
          fraction_active total_screens
116241312            1.00             2
117144               1.00             1
166897622            1.00             1
317373262            1.00             1
3914304              1.00            14
3915797              0.75             8
548481               1.00            21
6686268              1.00             1
84028191             1.00             1
```

Next, we will use the *ape* software library to get the protein sequences for these active targets, and to determine which species they may be from[4]. For more details on working with these sequences, consult the *ape* documentation.

```
> library(ape)
> targetSequences <- read.GenBank(row.names(drugTargets), species.names = TRUE)
```

Last, we print a table of the species names for each of these targets.

```
> cbind(attr(targetSequences, "species"), names(targetSequences))
```

```
      [,1]           [,2]
 [1,] "Homo_sapiens" "116241312"
 [2,] "Homo_sapiens" "117144"
 [3,] "Bos_taurus"   "166897622"
 [4,] "Homo_sapiens" "317373262"
 [5,] "Ovis_aries"   "3914304"
 [6,] "Homo_sapiens" "3915797"
 [7,] "Ovis_aries"   "548481"
 [8,] "Homo_sapiens" "6686268"
 [9,] "Homo_sapiens" "84028191"
```

6 Identify Target Selective Compounds

In the previous example, acetylsalicylic acid was found to show binding activity against numerous proteins, including the COX-1 cyclooxygenase enzyme (NCBI Protein ID 166897622). COX-1 activity is theorized to be the key mechanism in this molecules function as a nonsteroidal anti-inflammatory drug (NSAID). In this example, we will look for other small molecules which selectively bind to COX-1, under the assumption that these may be worth further investigation as potential nonsteroidal anti-inflammatory drugs. This example shows how *bioassayR* can be used identify small molecules which selectively bind to a target of interest, and assist in the discovery of small molecule drugs and molecular probes.

First, we will start by connecting to a database. Once again, the variable `sampleDatabasePath` can be replaced with the filename of the full PubChem Bioassay database (downloadable from http://chemmine.ucr.edu/bioassayr), if you would like to use that instead of the small example version included with this software package.

```
> library(bioassayR)
> extdata_dir <- system.file("extdata", package="bioassayR")
> sampleDatabasePath <- file.path(extdata_dir, "sampleDatabase.sqlite")
> pubChemDatabase <- connectBioassayDB(sampleDatabasePath)
```

The `activeAgainst` function can be used to show all small molecules in the database which demonstrate activity against COX-1 as follows. Each row name represents a small molecule cid. The column labeled "total assays" shows the total number of times each small molecule has been screened against the target of interest. The column labeled "fraction active" shows the portion of these which were annotated as active as a number between 0 and 1. This function allows users to consider different binding assays from distinct sources as replicates, to assist in distinguishing potentially spurious binding results from those with demonstrated reproducibility.

```
> activeCompounds <- activeAgainst(pubChemDatabase, "166897622")
> activeCompounds[1:10,] # look at the first 10 compounds
```

```
       fraction_active total_assays
2244                 1            1
2662                 1            3
3033                 1            1
3194                 1            1
3672                 1            1
3715                 1            2
133021               1            2
247704               1            1
444899               1            1
445580               1            1
```

Looking only at compounds which show binding to the target of interest is not sufficient for identifying drug candidates, as a portion of these compounds may be target unselective compounds (TUCs) which bind indiscriminately to a large number of distinct protein targets. The R function `selectiveAgainst` provides the user with a list of compounds that show activity against a target of interest (in at least one assay), while also showing limited activity against other targets.

The `maxCompounds` option limits the maximum number of results returned, and the `minimumTargets` option limits returned compounds to those screened against a specified minimum of distinct targets. Results are formatted as a `data.frame` whereby each row name represents a distinct compound. The first column shows the number of distinct targets this compound shows activity against, and the second shows the total number of targets it was screened against.

```
> selectiveCompounds <- selectiveAgainst(pubChemDatabase, "166897622",
+     maxCompounds = 10, minimumTargets = 1)
> selectiveCompounds
```

```
        active_targets tested_targets
2662                 1              1
3033                 1              1
133021               1              1
```

```
11314954              1              1
13015959              1              1
44563999              1              1
44564000              1              1
44564001              1              1
44564002              1              1
44564003              1              1
```

In the example database these compounds are only showing one tested target because very few assays are loaded. Users are encouraged to try this example for themselves with the full PubChem Bioassay database downloadable from http://chemmine.ucr.edu/bioassayr for a more interesting and informative result.

Users can combine *bioassayR* with the *ChemmineR* library to obtain structural information on these target selective compounds, and then perform further analysis- such as structural clustering, visualization, and computing physicochemical properties.

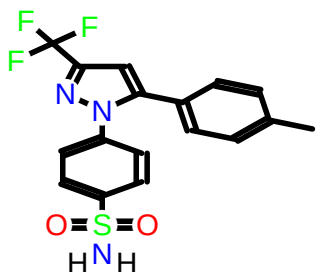
The *ChemmineR* software library can be used to download structural data for any of these compounds, and to visualize these structures as follows[5]. This example requires an active internet connection, as the compound structures are obtained from a remote server.

```
> library(ChemmineR)
> structures <- getIds(as.numeric(row.names(selectiveCompounds)))
```

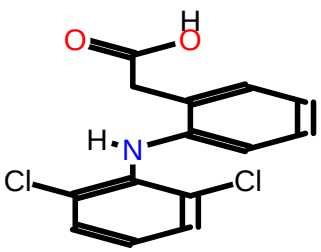
Here we visualize just the first four compounds found with `selectiveAgainst`. Consult the vignette supplied with the *ChemmineR* for numerous powerful examples of visualizing and analyzing these structures further.

```
> plot(structures[1:4], print=FALSE) # Plots structures to R graphics device
```

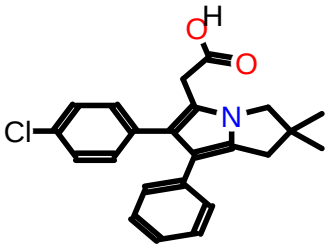
CMP1



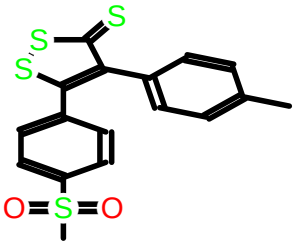
CMP2



CMP3



CMP4



7 Cluster Compounds by Activity Profile

This example demonstrates an example of clustering small molecules by similar bioactivity profiles across several distinct bioassay experiments. In many cases it is too cpu and memory intensive to cluster all compounds in the database, so we first pull just a subset of these data from the database into an `bioassaySet` object, and then convert that into a compounds vs targets activity matrix for subsequent clustering according to similarities in activity profiles. The function `getBioassaySetByCids` extracts the activity data for a given list of compounds. Alternatively, the entire data for a given list of assay ids can be extracted with the function `getAssays`.

First, connect to the included sample database:

```
> library(bioassayR)
> extdata_dir <- system.file("extdata", package="bioassayR")
> sampleDatabasePath <- file.path(extdata_dir, "sampleDatabase.sqlite")
> sampleDB <- connectBioassayDB(sampleDatabasePath)
```

Next, select data from just 3 compounds to extract into a `bioassaySet` object for subsequent analysis.

```
> compoundsOfInterest <- c("2244", "2662", "3715")
> selectedAssayData <- getBioassaySetByCids(sampleDB, compoundsOfInterest)
```

```
> selectedAssayData
```

```
class:                 bioassaySet
assays:                 1176
compounds:         3
targets:         453
sources:         bioassayR_testdata
```

The function perTargetMatrix converts the activity data extracted earlier into a matrix of targets (rows) vs compounds (columns). Data from multiple assays hitting the same target is summarized by giving a "1" (active) if any of these data are active. Both inactive scores and untested combinations are assigned zeros (inactive) to create a "complete matrix." We caution the user to carefully consider if this step makes sense within the context of the specific experiments being analyzed. Here a sparse matrix (which omits actually storing 0 values) is used to save memory.

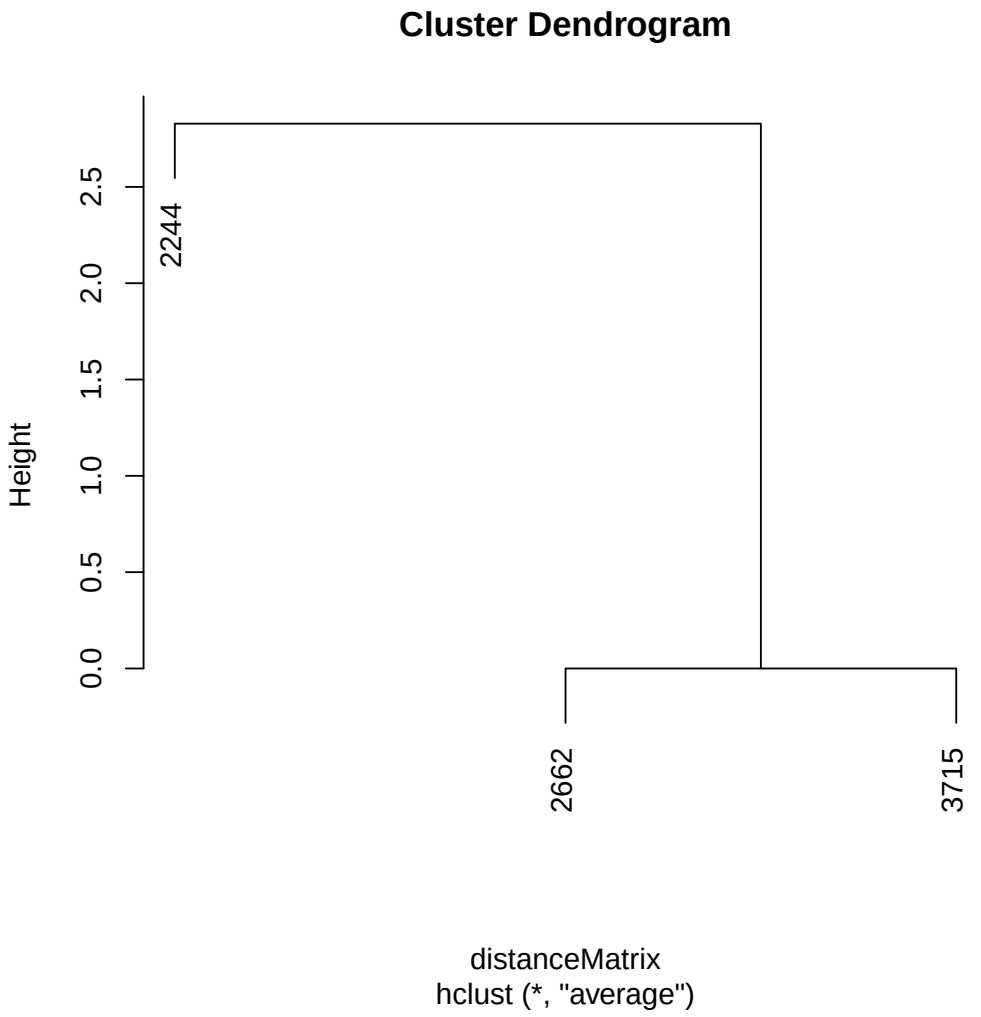
```
> myActivityMatrix <- perTargetMatrix(selectedAssayData)
> myActivityMatrix
```

```
9 x 3 sparse Matrix of class "dgCMatrix"
          2244 2662 3715
548481       1    .    .
3914304      1    .    .
3915797      1    .    .
166897622    1    1    1
317373262    1    .    .
117144       1    .    .
6686268      1    .    .
84028191     1    .    .
116241312    1    .    .
```

Cluster using the built in euclidean clustering functions within R to cluster. This provides a dendrogram which indicates the similarity amongst compounds according to their activity profiles.

```
> transposedMatrix <- t(myActivityMatrix)
> distanceMatrix <- dist(transposedMatrix)
> clusterResults <- hclust(distanceMatrix, method="average")
```

```
> plot(clusterResults)
```



Finally, disconnect from the database.

```
> disconnectBioassayDB(sampleDB)
```

8 Analyze and Load Raw Screening Data

This example demonstrates the basics of analyzing and loading data from a high throughput screening experiment with scores for 21,888 distinct compounds.

This example is based on the *cellHTS2* library. Example data is used which is included with *cellHTS2*. This is actually data from screening dsRNA against live cells, however we will treat it as small molecule binding data against a protein target as the data format is the same.

First read in the screening data provided with cellHTS2.

```
> library(cellHTS2)
> library(bioassayR)
> dataPath <- system.file("KcViab", package="cellHTS2")
> x <- readPlateList("Platelist.txt",
+                    name="KcViab",
```

```
+                    path=dataPath)
> x <- configure(x,
+                descripFile="Description.txt",
+                confFile="Plateconf.txt",
+                logFile="Screenlog.txt",
+                path=dataPath)
> xn <- normalizePlates(x,
+                       scale="multiplicative",
+                       log=FALSE,
+                       method="median",
+                       varianceAdjust="none")
```

Next, score and summarize the replicates.

```
> xsc <- scoreReplicates(xn, sign="-", method="zscore")
> xsc <- summarizeReplicates(xsc, summary="mean")
```

Parse the annotation data.

```
> xsc <- annotate(xsc, geneIDFile="GeneIDs_Dm_HFA_1.1.txt", path=dataPath)
```

Apply a sigmoidal transformation to generate binary calls.

```
> y <- scores2calls(xsc, z0=1.5, lambda=2)
> binaryCalls <- round(Data(y))
```

Convert the binary calls into an activity table that *bioassayR* can parse.

```
> scoreDataFrame <- cbind(geneAnno(y), binaryCalls)
> scoreDataFrame <- scoreDataFrame[wellAnno(y) == "sample",]
> activityTable <- cbind(cid=scoreDataFrame[,1], sid=scoreDataFrame[,1],
+      activity=scoreDataFrame[,2], score=Data(xsc))
> activityTable <- as.data.frame(activityTable)
> activityTable[1:10,]
```

```
       cid     sid activity              score
1  CG11371 CG11371        1   5.55306760617242
2  CG31671 CG31671        1   3.59768552803871
3  CG11376 CG11376        0     2.147814893189
4  CG11723 CG11723        0   4.00105151825918
5  CG12178 CG12178        0  0.955550327380792
6   CG7261  CG7261        0  0.768879954745058
7   CG2674  CG2674        0   1.12105534792054
8   CG7263  CG7263        0 -0.126481089856157
9   CG4822  CG4822        0  0.706942495432899
10  CG4265  CG4265        0  0.920455054477106
```

Create a new (temporary in this case) *bioassayR* database to load these data into.

```
> myDatabaseFilename <- tempfile()
> mydb <- newBioassayDB(myDatabaseFilename, indexed=F)
> addDataSource(mydb, description="other", version="unknown")
```

Create an assay object for the new assay.

```
> myAssay <- new("bioassay",aid="1", source_id="other",
+      assay_type="confirmatory", organism="unknown", scoring="activity rank",
+      targets="2224444", target_types="protein", scores=activityTable)
```

Load this assay object into the *bioassayR* database.

```
> loadBioassay(mydb, myAssay)
> mydb
```

```
class:                 BioassayDB
assays:                 1
sources:          other
writeable:         yes
```

Now that these data are loaded, you can use them to perform any of the other analysis examples in this document.

Lastly, for the purposes of this example, disconnect from the example database.

```
> disconnectBioassayDB(mydb)
```

9 Version Information

```
> toLatex(sessionInfo())
```

- R version 3.2.1 beta (2015-06-09 r68498), `x86_64-apple-darwin13.4.0`
- Locale: `C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, grDevices, graphics, grid, methods, parallel, stats, utils
- Other packages: Biobase 2.28.0, BiocGenerics 0.14.0, ChemmineR 2.20.3, DBI 0.3.1, Matrix 1.2-1, RColorBrewer 1.1-2, RSQLite 1.0.0, ape 3.3, bioassayR 1.6.1, cellHTS2 2.32.0, genefilter 1.50.0, hwriter 1.3.2, locfit 1.5-9.1, rjson 0.2.15, splots 1.34.0, vsn 3.36.0
- Loaded via a namespace (and not attached): AnnotationDbi 1.30.1, BiocInstaller 1.18.3, BiocStyle 1.6.0, Category 2.34.2, DEoptimR 1.0-2, GSEABase 1.30.2, GenomeInfoDb 1.4.0, IRanges 2.2.4, MASS 7.3-40, RBGL 1.44.0, RCurl 1.95-4.6, Rcpp 0.11.6, S4Vectors 0.6.0, XML 3.98-1.2, affy 1.46.1, affyio 1.36.0, annotate 1.46.0, bitops 1.0-6, cluster 2.0.1, colorspace 1.2-6, digest 0.6.8, ggplot2 1.0.1, graph 1.46.0, gtable 0.1.2, lattice 0.20-31, limma 3.24.10, magrittr 1.5, munsell 0.4.2, mvtnorm 1.0-2, nlme 3.1-120, pcaPP 1.9-60, plyr 1.8.2, prada 1.44.0, preprocessCore 1.30.0, proto 0.3-10, reshape2 1.4.1, robustbase 0.92-3, rrcov 1.3-8, scales 0.2.4, splines 3.2.1, stats4 3.2.1, stringi 0.4-1, stringr 1.0.0, survival 2.38-1, tools 3.2.1, xtable 1.7-4, zlibbioc 1.14.0

10 Funding

This software was developed with funding from the National Science Foundation: ABI-0957099, 2010-0520325 and IGERT-0504249.

References

[1] T W Backman, Y Cao, and T Girke. ChemMine tools: an online service for analyzing and clustering small molecules. *Nucleic Acids Res*, 39(Web Server issue):486–491, Jul 2011. URL: http://www.hubmed.org/display.cgi?uids=21576229, doi:10.1093/nar/gkr320.

[2] Yanli Wang, Jewen Xiao, Tugba O Suzek, Jian Zhang, Jiyao Wang, Zhigang Zhou, Lianyi Han, Karen Karapetyan, Svetlana Dracheva, Benjamin A Shoemaker, Evan Bolton, Asta Gindulyte, and Stephen H Bryant. PubChem's BioAssay Database. *Nucleic acids research*, 40(Database issue):D400–12, January 2012.

[3] David S Wishart, Craig Knox, An Chi Guo, Dean Cheng, Savita Shrivastava, Dan Tzur, Bijaya Gautam, and Murtaza Hassanali. DrugBank: a knowledgebase for drugs, drug actions and drug targets. *Nucleic acids research*, 36(Database issue):D901–6, January 2008.

[4] E. Paradis, J. Claude, and K. Strimmer. APE: analyses of phylogenetics and evolution in R language. *Bioinformatics*, 20:289–290, 2004.

[5] Y Cao, A Charisi, L C Cheng, T Jiang, and T Girke. ChemmineR: a compound mining framework for R. *Bioinformatics*, 24(15):1733–1734, Aug 2008. URL: http://www.hubmed.org/display.cgi?uids=18596077, doi:10.1093/bioinformatics/btn307.