Journal of Statistical Software

January 2007, Volume 18, Issue 5. *http://www.jstatsoft.org/*

Chemical Informatics Functionality in R

Rajarshi Guha
Indiana University

Abstract

The flexibility and scope of the R programming environment has made it a popular choice for statistical modeling and scientific prototyping in a number of fields. In the field of chemistry, R provides several tools for a variety of problems related to statistical modeling of chemical information. However, one aspect common to these tools is that they do not have direct access to the information that is available from chemical structures, such as contained in molecular descriptors.

We describe the **rcdk** package that provides the R user with access to the **CDK**, a Java framework for cheminformatics. As a result, it is possible to read in a variety of molecular formats, calculate molecular descriptors and evaluate fingerprints. In addition, we describe the **rpubchem** that will allow access to the data in PubChem, a public repository of molecular structures and associated assay data for approximately 8 million compounds. Currently, the package allows access to structural information as well as some simple molecular properties from PubChem. In addition the package allows access to bio-assay data from the PubChem FTP servers.

Keywords: R, Java, **CDK**, PubChem, cheminformatics.

1. Background

The R programming language (R Development Core Team 2006) has been used for the purposes of statistical analyses in a variety of fields. Numerous examples of its use in chemistry can be found (Versteeg, Richardson, and Rowe 2006; Vidal, Thormann, and Pons 2005; Guha, Dutta, and Jurs 2006). However, the feature common to these applications is that R is used for the purposes of modeling data that has been generated outside the R environment. Examples of such data are the results of spectroscopic or chromatographic analyzes and molecular descriptor values calculated using a variety of software. In many situations, this is sufficient since we are more interested in the data analysis rather than in how to get the data.

However, given that R can be considered a general purpose programming language, oriented towards prototyping scientific applications, it would be useful in a number of cases if we were able to access cheminformatics functionality within the R environment. That is, we are interested in being able to represent, manipulate and analyze chemical structures and related information from within R. A number of cheminformatics toolkits are available including **OEChem** (OpenEye Scientific Software, Inc. 2006), **JOELib** (**JOELib** Development Team 2006) and the **CDK** (Steinbeck, Hoppe, Kuhn, Floris, Guha, and Willighagen 2006). These toolkits provide a variety of functionality including the ability to read in chemical file formats and obtaining molecular features such as atom counts and so on. Furthermore, they allow the manipulation of chemical structures in terms of atoms, bonds and whole molecules. An additional feature of **JOELib** and the **CDK** also allow the calculation of a variety of molecular descriptors which are fundamental to the development of quantitative structure-activity relationship (QSAR) models.

Enhancing the R environment by integrating one or more of the above packages allows the chemical modeler to work in an environment that provides the tools to extract molecular information and subsequently develop statistical and mathematical models using the information. It should be noted that the above toolkits are libraries and thus, do not always provide prepackaged functionality. In general, cheminformatics applications will need to be composed using the methods provided by the toolkits. Hence, the integration of cheminformatics toolkits in R does not lead to a full-fledged cheminformatics modeling application. Rather, it provides the tools that are needed to develop such an application in the R environment.

In addition to being able to manipulate chemical structures it is also useful to be able to access repositories of chemical information. In the past such repositories were not common, unlike the field of bioinformatics which have a number of freely accessible databases for gene sequences and protein structures. Recently, efforts in the field of chemistry include ZINC (Irwin and Shoichet 2005) which is a database of more than 2.7 million, commercially available compounds, and the PubChem initiative (`http://pubchem.ncbi.nlm.nih.gov/`) which provides publicly accessible databases for chemical compounds and biological assay data. Currently, the compound database had more than 8 million entries, which included 2D structure, SMILES representations, synonyms and a number of constitutional descriptors (such as atom count, hydrogen bond donor and acceptor counts). The biological assay data currently contains 276 datasets ranging in size from a few hundred molecules to more than 80000 molecules.

Clearly, the PubChem databases are a rich source of chemical information and thus very attractive for the purposes of chemical data mining. Thus access to the PubChem repository would further enhance a cheminformatics modeling environment based on R.

2. The rcdk Package

The **rcdk** package was designed for the purposes of enhancing the R environment with cheminformatics functionality provided by the **CDK** toolkit, an open-source Java framework for cheminformatics. The following sections describe the requirements and functionality available and present some simple examples of the usage of this package.

2.1. Requirements

As noted above the **CDK** toolkit is a Java library. To access the **CDK** API from R we require an R-Java interface. Two such interfaces are available: **SJava** (Temple Lang and Chambers 2005) and **rJava** (Urbanek 2006). Versions of the **rcdk** package prior to 2.0 (Guha 2006a) required the **SJava** package. However, due to difficulties in installation of the **SJava**, we no longer support the older versions of **rcdk**. From version 2.0 onwards we now require **rJava**.

The **rcdk** package is available in source form and comes with a precompiled *jar* file and R source code. The *jar* file was compiled with Java 1.5.0. It is probably a good idea to ensure that the users **rJava** installation and the **rcdk** package use the same version of Java. The complete R and Java sources for the **rcdk** package are available from `http://cheminfo.informatics.indiana.edu/~rguha/code/R/#rcdk`

The **rcdk** package includes the *jar* files from the **CDK**, **JChemPaint** (Krause, Willighagen, and Steinbeck 2000) and **Jmol** (**Jmol** Development Team 2006) projects. Thus, the user is not required to obtain any external software to fully utilize the package.

2.2. Functionality

The **CDK** provides a wide variety of functionality including reading and writing molecular file formats, generating binary fingerprints and calculating molecular descriptors. Since the **CDK** is a toolkit library much of these functionalities must be tied together using an appropriate sequence of method calls. The goal of the **rcdk** package is to encapsulate many of the more common features of the **CDK** into simple one or two line function calls. In addition to exposing the cheminformatics API of the **CDK**, the **rcdk** package also integrates the **JChemPaint** 2D structure editor and the **Jmol** 3D structure viewer. The inclusion of these two tools allows the user to draw and subsequently manipulate molecular structures directly in the R environment. It should be noted that though the goal of the **rcdk** package is to allow easy use of **CDK** functionality, this does not preclude the user from directly accessing **CDK** classes and methods using **rJava**. Table 1 summarizes the R functions that are currently available.

Though these functions do not encompass the whole **CDK** API, they are sufficient for a number of tasks common in cheminformatics work such as loading molecules and evaluating molecular descriptors.

The functions related to molecular descriptors provide an example of the complexity hidden by this package. A molecular descriptor is a numerical characterization of some structural feature of a molecule. Many types of descriptors are available and a good overview can be found in Todeschini and Consonni (2002). The **CDK** provides a number of descriptor implementations. These are all derived from a set of base classes and implement a uniform interface. Ordinarily, we must first choose which descriptor to use and then instantiate the required class. Once the descriptor has been evaluated, the return value is not a numeric value, but rather a Java object, which must be interrogated to determine the actual numeric value (or values) of the descriptor. Though this whole process can be performed using **rJava** calls, it is very tedious. With the help of **rcdk** it is possible to write:

```
R> desc.values <- eval.desc(aMolecule,
+  "org.openscience.cdk.qsar.descriptors.molecular.BCUTDescriptor")
```

At this point, the **rcdk** interface does not allow setting the parameters for individual descriptor methods and as a result we are restricted to the default values, though this functionality will be included in later versions of the package. A question that might arise is how to determine the names of the descriptor classes for use in `eval.desc`. The **CDK** uses a dynamic

	Function	**Description**
Editing and viewing	`draw.molecule`	Draw 2D structures
	`view.molecule`	View a molecule in 3D
	`view.molecule.2d`	View a molecule in 2D
	`view.molecule.table`	View 3D molecules along with data
IO	`load.molecules`	Load arbitrary molecular file formats
	`write.molecules`	Write molecules to MDL SD formatted files
Descriptors	`eval.desc`	Evaluate a single descriptor
	`get.desc.engine`	Get the descriptor *engine* which can be used to automatically generate all descriptors
	`get.desc.classnames`	Get a list of available descriptors
Miscellaneous	`get.fingerprint`	Generate binary fingerprints
	`get.smiles`	Obtain SMILES representations
	`parse.smiles`	Parse a SMILES representation into a **CDK** molecule object
	`get.property`	Retrieve arbitrary properties on molecules
	`set.property`	Set arbitrary properties on molecules
	`remove.property`	Remove a property from a molecule
	`get.totalcharge`	Get the total charge of a molecule
	`get.total.hydrogen.count`	Get the count of (implicit and explicit) hydrogens
	`remove.hydrogens`	Remove hydrogens from a molecule

Table 1: A summary of the functions available in the **rcdk** package.

approach to determine available descriptor classes. Thus, by calling `get.desc.engine` and `get.desc.classnames` it is possible to get a list of available descriptor classes matching a specific type (atomic, bond or molecular descriptors). From this list we can then choose the descriptor of interest. Currently, the **rcdk** package will only provide a list of descriptor names.

In the area of input/output functionality, the **CDK** can read in a variety of file formats. This is accessed in R using the `load.molecules()` function which simply takes a vector of file names and loads the molecules into individual objects. Note that some formats allow multiple molecules in a single file. This function will load each molecule in a multi-molecule file. The result is a list of Java object references, that can be used in other **rcdk** functions.

Molecule objects can also be saved to disk in the MDL SD format via `write.molecules()`. Since the SD format can include numeric and character data by the use of *SD tags*, it is possible to annotate molecules with data (source information, descriptor values etc.) via `set.property()`. This is an important aspect since molecule objects in the R environment

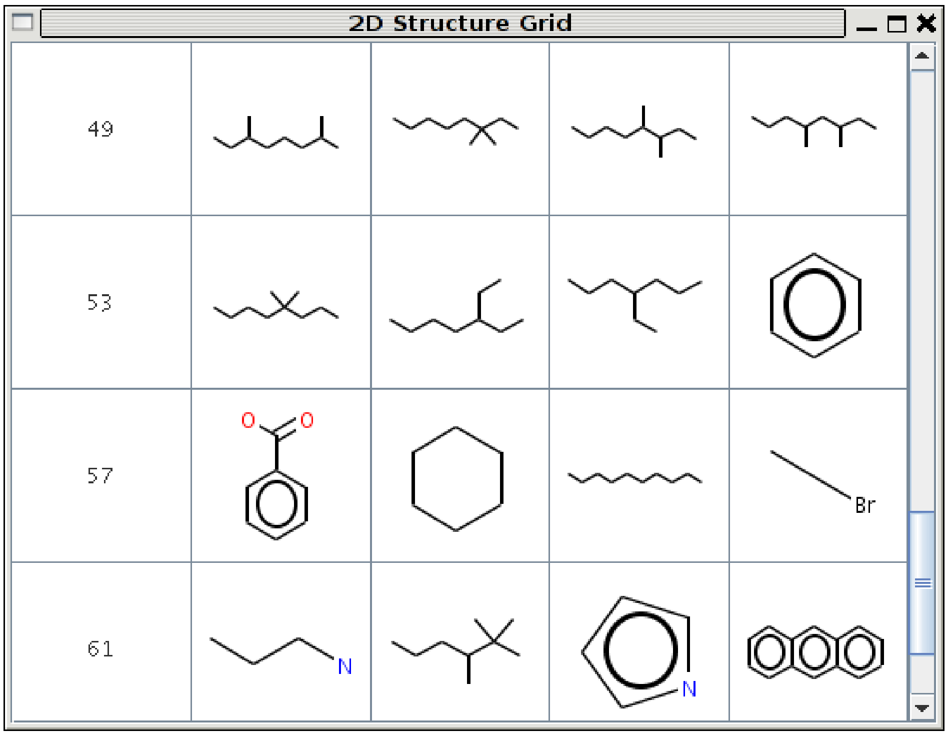

Figure 1: A display of a set of structures in 2D.

are really references to Java objects. As such it is not possible to save such objects via the `save` function to the R binary format. The use of annotated SD files allows for persistence.

One attractive feature of the **rcdk** package is the visualization functions. The **CDK** itself provides functionality to generate 2D structure diagrams from connectivity information. This feature can be accessed using the `view.molecule.2d()` function and an example of the 2D visualization of a set of structures is shown in Figure 1. In addition, the **JChemPaint** program which is built on top of the **CDK** toolkit can be used to draw 2D structures. This can be accessed via the `draw.molecule()` function. This is a very useful feature since it is possible to create 2D structures within the R environment, without having to use external editors. Currently, this approach restricts us to generating 2D molecular coordinates. However, the **CDK** project is currently working on the development of a 3D model builder. This would allow the 2D structure drawn in using **JChemPaint** to be geometry optimized resulting in 3D coordinates.

If the molecule is loaded with 3D coordinates, `view.molecule.3d()` can be used to view the 3D structure using the **Jmol** viewer. Figure 2 shows a display of a set of molecules in 3D in which each molecule can be rotated and zoomed. The **rcdk** package also provides a function to view a table of molecular structures along with one or more columns of data. This is very useful when performing QSAR modeling as we are usually interested in viewing the structures of certain groups of molecules and their associated descriptor and property values. Currently, the table view only provides 3D structures but this will be updated to allow the user to choose between 2D and 3D views.

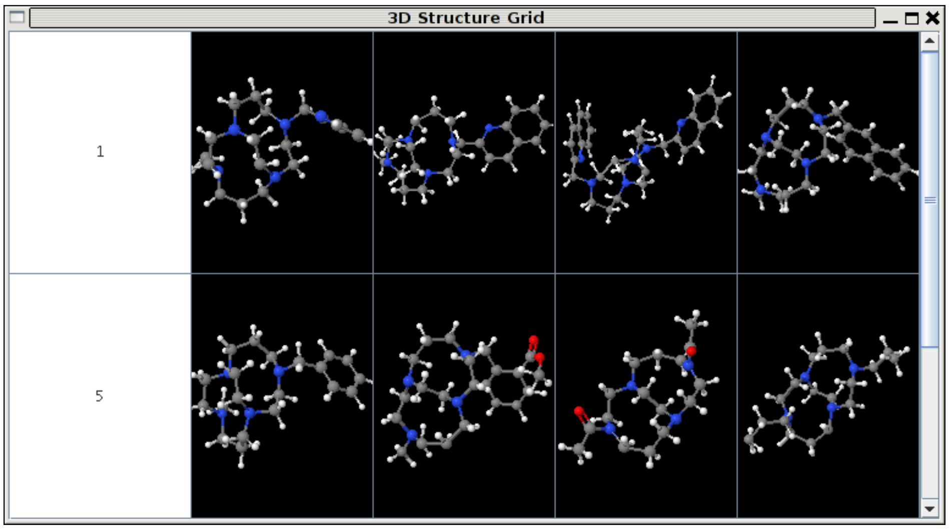

Figure 2: A display of a set of structures in 3D.

2.3. A clustering example

Using the **rcdk** package it is very easy to perform a number of cheminformatics tasks. A common example is the clustering of a set of molecules. There have been a number of studies on the application of clustering for cheminformatics and the most common method is to use binary fingerprints with some form of hierarchical clustering. Binary fingerprints are bit strings, ranging in length from 1024 bits to as many as 27000 bits, in which each bit characterizes a specific molecular feature. However, certain fingerprints using a hashing scheme, as a result of which, the one to one correspondence between bit positions and structural features are lost. A more detailed description of fingerprints can be found in Gasteiger (2003).

As an example of the usage of the **rcdk** package we present a clustering example. We considered a dataset of 756 dihydrofolate reductase (DHFR) inhibitors studied by Sutherland, O'Brien, and Weaver (2003). The first steps are to load the molecules and generate the fingerprints:

```
R> library("rcdk")
R> mols <- load.molecules("dhfr_3d.sd")
R> fp.list <- lapply(mols, get.fingerprint)
```

The result is to obtain a `list` of numeric vectors. Each element of the vector indicates a bit position that is set to 1 in the fingerprint. Before performing the clustering we need to generate a distance matrix. Traditionally, distance matrices for binary fingerprints are generated using the Tanimoto metric, though other metrics are available (Willet, Barnard, and Downs 1998; Holliday, Salim, Whittle, and Willet 2003). This step can be performed using the **fingerprint** package (Guha 2006b). This package is designed to handle binary fingerprint data and provides a function to evaluate a distance matrix using the Tanimoto distance. Thus we have

```
R> library("fingerprint")
```

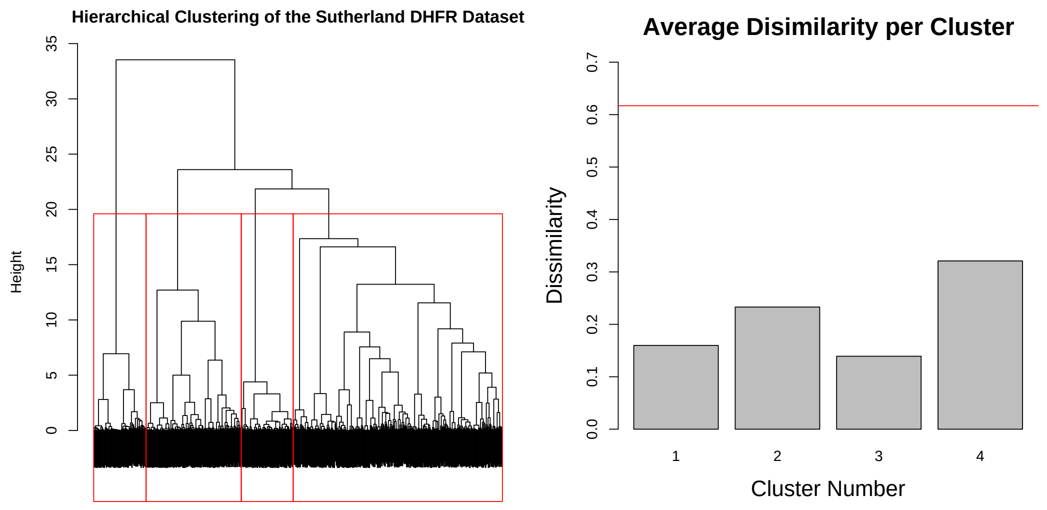

Figure 3: Hierarchical clustering of the DHFR dataset using binary fingerprints.

```
R> fp.dist <- fp.sim.matrix(fp.list)
R> fp.dist <- as.dist(1-fp.dist)
```

Note that we the elements of the similarity matrix from 1.0, since clustering routines work with a *dissimilarity* matrix. Note that the calculation of the similarity matrix can take a long time since we must evaluate $N(N-1)/2$ pairwise similarities, where N is the number of molecules in the dataset. Once we have the distance matrix we can then cluster the molecular structures. For this example we investigate a hierarchical clustering using `hclust()` using *wards* method (Ward 1963):

```
R> clus.hier <- hclust(fp.dist, method="ward")
```

Figure 3 shows a dendrogram of the resulting clustering. We then manually selected the four top-level clusters in the dendrogram (highlighted) and then evaluated the average dissimilarity for the individual clusters. These are plotted in the right hand graph in Figure 3. The red line indicates the average dissimilarity of the whole dataset.

```
R> plot(clus.hier, label=FALSE)
R> members <- identify(clus.hier, N=4) # manually select clusters
R> avg.dissim <- unlist(lapply(members, function(x) {
+  d <- as.matrix(fp.dist)
+  mean(d[x,x])/2
+  }))
```

Although the figure indicates that the clustering appears to be successful, the average silhouette width for the four clusters is only 0.11. Performing clustering using the `clara()` function with a variety of k indicated that the clustering was not of high quality as measured by the average silhouette width of the various clusterings.

2.4. A QSAR modeling example

Another example usage of the **rcdk** package involves the development of a QSAR model to predict some molecular property. QSAR models have been developed for a wide variety of physical properties (such as boiling point and aqueous solubility) and biological activities (such as carcinogenecity and toxicity). For this example we consider a data set of 259 molecules whose measured property was boiling point and has been studied previously by Goll and Jurs (1999).

Since we have already loaded the molecules we focus on the calculation of molecular descriptors. A molecular descriptor is a numerical representation of some aspect of the chemical structure. Molecular descriptors can be considered in terms of groups: constitutional (counts of atoms, bonds, etc.), geometrical (characterizing the 3D structure of a molecule), topological (characterizing the molecule in terms of a graph theoretical representation) and electronic (characterizing the molecule in terms of electronic features). The descriptors thus represent the structural features of a molecule that can be used to obtained predictions of the property under study. The **CDK** provides a number of molecular descriptors covering all the categories mentioned and is a continuous work in progress.

As described in Section 2.2, it is possible to query the **CDK** API to determine which descriptors are available. For the purposes of this example we will manually specify a set of descriptors to calculate.

```
R> # load the data
R> mols <- load.molecules("bp.sdf")

R> # load the dependent variable
R> depv <- scan("depv.txt")

R> desc.names <- c(
+  "org.openscience.cdk.qsar.descriptors.molecular.BCUTDescriptor",
+  "org.openscience.cdk.qsar.descriptors.molecular.MDEDescriptor",
+  "org.openscience.cdk.qsar.descriptors.molecular.CPSADescriptor",
+  "org.openscience.cdk.qsar.descriptors.molecular.XLogPDescriptor",
+  "org.openscience.cdk.qsar.descriptors.molecular.KappaShapeIndicesDescriptor",
+  "org.openscience.cdk.qsar.descriptors.molecular.GravitationalIndexDescriptor",
+  "org.openscience.cdk.qsar.descriptors.molecular.TPSADescriptor",
+  "org.openscience.cdk.qsar.descriptors.molecular.WeightedPathDescriptor")

R> # some setup to get column names
R> prefix <- gsub("org.openscience.cdk.qsar.descriptors.molecular",
+  "", desc.names)
R> prefix <- gsub("Descriptor", "", prefix)

R> # now calculate these descriptors
R> desc.list <- list()
R> for (i in 1:length(mols)) {
+  tmp <- c()
+  for (j in 1:length(desc.names)) {
```

```
+    values <-  eval.desc(mols.qsar[[i]], desc.names[j])
+    tmp <- c(tmp, values)
+    if (i == 1)
+      data.names <- c(data.names, paste(prefix[j],
+      1:length(values), sep="."))
+  }
+  desc.list[[i]] <- tmp
+  }
R> desc.data <- as.data.frame(do.call("rbind", desc.list))
R> names(desc.data) <- data.names
```

At this point we will have 73 calculated descriptors for the 259 molecules. Many of the descriptors will be correlated with each other and a number of them will be constant (or near-constant). The next step is then to reduce the size of the pool of descriptors. This is easily performed using R. After such reduction we are left with a pool of 38 descriptors. Using all the descriptors in the reduced pool will, in general, lead to over-fitting. Thus the next step is to obtain a small subset of descriptors from this reduced pool. A number of techniques are available for subset selection and for this example we use an exhaustive search using the **leaps** package.

```
R> library("leaps")
R> # autoscale the data
R> desc.data <- as.data.frame(apply(desc.data, 2,
+  function(x) (x-mean(x))/sd(x)))

R> # assign training and prediction set
R> tset <- 1:200
R> pset <- 201:259
R> sel <- regsubsets(x=desc.data[tset,], y=depv[tset],
+  method="exhaustive", nvmax=4)
```

Note that, in the above code, we have used a subset of the dataset to perform model selection. This subset is also used to build the final model, which is then tested for predictive ability using the remainder of the dataset.

Examining the output of the above function we see that descriptors `BCUT.6`, `CPSA.12`, `WeightedPath.1` and `MDE.11` were selected. Using these descriptor we can then build the linear regression model which is summarized below, followed by obtaining the predicted boiling point for the prediction set:

```
R> model <- lm(y ~ BCUT.6 + CPSA.12 + WeightedPath.3 + MDE.11,
+  data=data.frame(y=depv[tset], desc.data[tset,]))
R> summary(model)
Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)      25.611     13.743   1.864   0.0639 .
BCUT.6           26.155      1.503  17.406  < 2e-16 ***
CPSA.12        2565.552    181.668  14.122  < 2e-16 ***
```

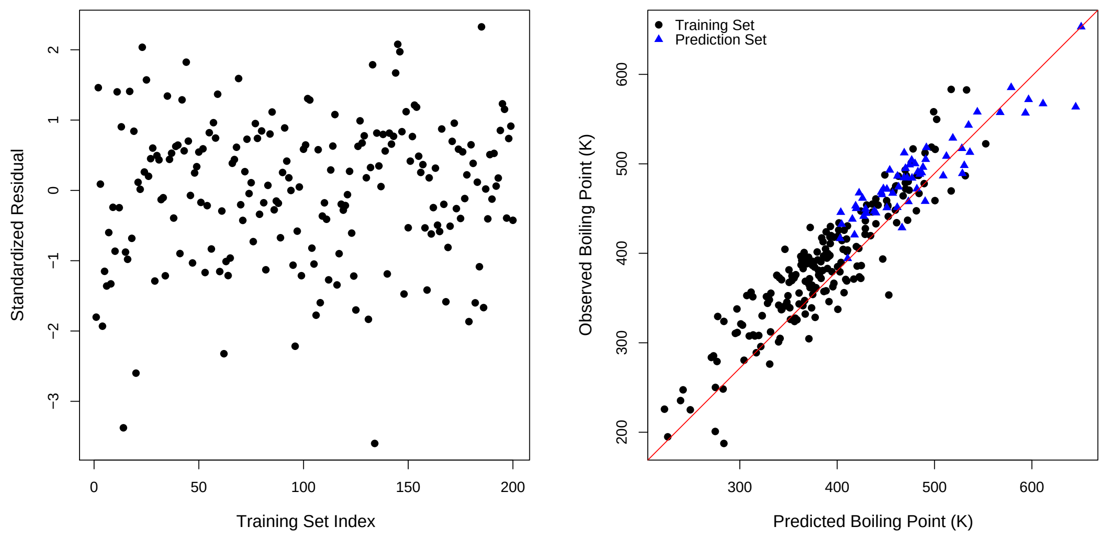

Figure 4: Standardized residual plot for the training set and a plot of the observed versus predicted boiling points for the training and prediction sets.

```
WeightedPath.1    7.645      0.608  12.573  < 2e-16 ***
MDE.11           91.355     17.916   5.099 8.05e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 28.88 on 195 degrees of freedom
Multiple R-Squared: 0.8311,     Adjusted R-squared: 0.8276
F-statistic: 239.9 on 4 and 195 DF,  p-value: < 2.2e-16

R> preds <- predict(model, desc.data[pset,])
R> cor(preds, depv[pset])^2
[1] 0.8351248
```

Figure 4 shows a residual plot and a plot of the observed versus predicted boiling points for the training and prediction sets. The model is statistically significant, though it is not the best model that can be obtained, in terms of performance. One reason for this is that we have calculated only a very small set of descriptors. In addition alternative feature selection algorithms might lead to a subset that would exhibit better performance. However the main point of this example is that it is possible to use R as an integrated environment for the purposes of modeling and analysis of chemical datasets.

3. The rpubchem Package

The goal of the PubChem initiative is to provide a public repository of chemical structures along with biological assay data. The structure and assay data are collected from a number of sources such as screening centers as well as other collections (such as ZINC). Currently the repository has structural information on more than 8 million compounds and has assay results for 276 assays. From the point of view of chemical data mining, the PubChem repository provide a rich source of data.

3.1. Goals

The PubChem website provides forms to search for structures and assays using a variety of methods. It is important to note that PubChem does not provide 3D structure information. As a result we obtain structures in the form of SMILES strings. In addition to the structure itself, PubChem precalculates a number of simple properties. The goal of the **rpubchem** package is to provide easy access structure, property and assay information from within R. Each compound in the PubChem repository is assigned a unique ID number. The **rpubchem** package allows us to download the SMILES string and property values for arbitrary sets of compound IDs. Analogously, each assay is also assigned an assay ID and assay data can be downloaded via ID number. However, in many cases the assay ID is not known but rather, we would like to determine which assays are related to a certain topic, like *yeast* or *cancer*. The **rpubchem** package provides keyword based searches of PubChem and then returns the assay IDs for those assays whose descriptions contain those keywords.

3.2. Usage

A simple example is to get the structure data for 10 random compound IDs.

```
R> library("rpubchem")
R> structs <- get.cid( sample(1:100000, 10) )
```

The result is a `data.frame` with 7 columns containing the canonical SMILES, molecular weight, topological polar surface area (Ertl, Rhode, and Selzer 2000), XLogP value (Wang, Fu, and Lai 1997), heavy atom count, formal charge and the count of hydrogen bond donors and acceptors. Though this information is useful, it is of limited value on its own, since they provide a limited view of the various structural features. However coupled with the **rcdk** package we can parse the column of SMILES strings to obtain molecule objects which can then be visualized, saved to alternate formats and evaluated for descriptors. Thus to view the structures that we just obtained we could do:

```
R> library("rcdk")
R> pmols <- lapply(structs[,2], parse.smiles)
R> view.molecule.2d(pmols)
```

However, structural information is not the whole story. To build and test models we require measured data, which is available from the PubChem bio-assay repository. As noted above it is possible to specify an assay ID and obtain the assay data. But when the ID is not known, it is useful to search for relevant assays. The **rpubchem** package provides the `find.assay.id()` function to perform this search

```
R> aids <- find.assay.id("lung+cancer+carcinoma")
```

The above search returns 27 assay IDs. At this point, this function does not return any other information than the assay ID. However, PubChem stores a description of each assay which

Field Name	**Meaning**
`PUBCHEM.SID`	Substance ID
`PUBCHEM.CID`	Compound ID
`PUBCHEM.ACTIVITY.OUTCOME`	Activity outcome, which can be one of *active, inactive, inconclusive, unspecified*
`PUBCHEM.ACTIVITY.SCORE`	PubChem activity score, higher is more active
`PUBCHEM.ASSAYDATA.COMMENT`	Test result specific comment

Table 2: The fields common to every assay dataset returned by `get.assay`

can be accessed by using the `get.assay.desc()` function. This function returns a `list` with three components which provide a short description, comments and explanation of the assay specific columns. The following call, for instance is necessary to get the one-line descriptions of all the assay IDs we had obtained in the previous step.

```
R> sapply(aids, function(x) get.assay.desc(x)$assay.desc)
```

Given the assay ID, we can now download the actual assay data

```
R> adata <- get.assay(175)
```

Note that the size of assays varies from a few hundred to more than 80000 observations. Thus, for larger assays the download can be lengthy. The above function will return the data in the form of a `data.frame`.

Each PubChem assay has a number of fixed fields. The `data.frame` contains a subset of these fixed fields listed in Table 2.

However, each assay can also contain an arbitrary number of extra fields, whose meaning is generally specific to the assay. The **rpubchem** package handles this by obtaining and parsing an XML description of the given assay from PubChem. This description contains names for the extra columns as well as information such as a description and comments for the assay in question as well as a description of each of the extra columns. As an example we can obtain this information for the assay we have downloaded before:

```
R> names(adata)
[1] "PUBCHEM.SID"               "PUBCHEM.CID"
[3] "PUBCHEM.ACTIVITY.OUTCOME"  "PUBCHEM.ACTIVITY.SCORE"
[5] "PUBCHEM.ASSAYDATA.COMMENT" "conc"
[7] "giprct"                    "nbrtest"
[9] "range"

R> attr(adata, "comments")
[1] "These data are a subset of the data from the NCI yeast anticancer
drug screen. Compounds are identified by the NCI NSC number. In the
Seattle Project numbering system, the mlh1 rad18 strain is SPY number
50858. Compounds with inhibition of growth(giprct) >= 70% were
```

```
considered active. Activity score was based on increasing values of giprct."

R> attr(adata, "types")
$conc
[1] "concentration tested, unit: micromolar"
[2] "uM"

$giprct
[1] "Inhibition of growth compared to untreated controls (%)"
[2] "NA"

$nbrtest
[1] "Number of tests averaged for this data point"
[2] "NA"

$range
[1] "Range of values for this data point" "NA"
```

The `"types"` attribute is a list keyed on the names of the extra columns. Each element of this list is a tuple containing a short description of the column and the units of the column (`NA` if it is unit-less).

Once we have the assay data, we can use the `PUBCHEM.CID` column to obtain the structural information of the compounds that have been tested in the assay. It is important to note that for assays where very large numbers of compounds have been tested, obtaining structural information for them at one go may not be possible at one go due to limits of the PubChem query mechanism. In this case, it is advised to download structural information in chunks of 1000 compound IDs at a time.

4. Conclusions

The **rcdk** and **rpubchem** packages enhance the utility of R in the field of cheminformatics. The latter allows us to manipulate and process chemical structures within the R environment and the former allows access to a rich source of chemical and biological assay data. We believe that the use of these packages will enable the cheminformatics oriented R user to work with chemical information in an efficient manner.

A number of enhancements to the **rcdk** package are planned. One of the main ongoing tasks is to include further coverage of the **CDK** API. This is tied to the development of the **CDK** itself such as the development of the 3D model builder. It should be noted that since we can access the **CDK** API directly, via **rJava**, the focus of the functions provided by the **rcdk** package is to make cheminformatics functionality easily available. That is, the **rcdk** package is task oriented.

One topic of future work is the issue of descriptor names. As the example in Section 2.4 showed, the user must define a set of names for the calculated descriptors. This is not an optimal situation since the resultant names may differ from user to user, thus hindering interpretation. This issue is being addressed within the **CDK** project and future versions of **rcdk** will assign names for the descriptors automatically. The **CDK** also includes additional meta-data for each descriptor such as a categorization and definition. This information can be used to enhance descriptor selection and future work will include this in the list of available descriptors.

As noted previously, output of molecule objects is restricted to the MDL SD format. Future work involves providing a variety of formats supported by the **CDK**, notably Chemical Markup Language (CML)

In terms of visualization, we plan to implement structure-data tables for the 2D case, which should be faster and more memory efficient than displaying such a table using **Jmol**.

With respect to the **rpubchem** package, we plan to implement a more transparent approach to downloading large sets of compound data, rather than requiring the user to manually download them in sets of 1000. As noted before this is restriction imposed by PubChem rather than the **rpubchem** package itself.

Acknowledgments

This work was supported by NIH grant No. NIH-NHGRI/P20 HG 003894-01/.

References

Ertl P, Rhode B, Selzer P (2000). "Fast Calculation of Molecular Polar Surface Area as a Sum of Fragment-Based Contributions and Its Application to the Prediction of Drug Transport Properties." *Journal of Medicinal Chemistry*, **43**(20), 3714–3717.

Gasteiger J (ed.) (2003). *Handbook of Cheminformatics*, volume 1. Wiley-VCH, Weinheim.

Goll E, Jurs P (1999). "Prediction of the Normal Boiling Points of Organic Compounds from Molecular Structures with a Computational Neural Network Model." *Journal of Chemical Information and Computer Science*, **39**(6), 974–983.

Guha R (2006a). "Generating, Using and Visualizing Molecular Information in R." *R News*, **3**(3), 28–33.

Guha R (2006b). **fingerprint***: Functions to Operate on Binary Fingerprint Data.* R package version 2.2, URL http://CRAN.R-project.org/.

Guha R, Dutta D, Jurs P (2006). "R-NN Curves: An Intuitive Approach to Outlier Detection Using a Distance Based Method." *Journal of Chemical Information and Modeling*, **46**(4), 1713–1722.

Holliday J, Salim N, Whittle M, Willet P (2003). "Analysis and Display of the Size Dependence of Chemical Similarity Coefficients." *Journal of Chemical Information and Computer Science*, **43**(3), 819–828.

Irwin J, Shoichet B (2005). "ZINC – A Free Database of Commercially Available Compounds for Virtual Screening." *Journal of Chemical Information and Modeling*, **45**(1), 177–182.

Jmol Development Team (2006). "**Jmol**: A Java-based 3D Molecular Structure Viewer." URL http://www.jmol.org/.

JOELib Development Team (2006). "**JOELib**: A Java-based Cheminformatics Library." URL http://www-ra.informatik.uni-tuebingen.de/software/joelib/.

Krause S, Willighagen E, Steinbeck C (2000). "JChemPaint - Using the Collaborative Forces of the Internet to Develop a Free Editor for 2D Chemical Structures." *Molecules*, **5**, 93–98.

OpenEye Scientific Software, Inc (2006). "**OEChem** 1.4.2." URL http://www.eyesopen.com/.

R Development Core Team (2006). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Steinbeck C, Hoppe C, Kuhn S, Floris M, Guha R, Willighagen E (2006). "Recent Developments of the Chemistry Development Kit (**CDK**) – An Open-Source Java Library for Chemo- and Bioinformatics." *Current Pharmaceutical Design*, **12**(17), 2110–2120.

Sutherland J, O'Brien L, Weaver D (2003). "Spline-Fitting with a Genetic Algorithm: A Method for Developing Classification Structure-Activity Relationships." *Journal of Chemical Information and Computer Science*, **43**(6), 1906–1915.

Temple Lang D, Chambers JM (2005). ***SJava**: The Omegahat Interface for R and Java*. Omegahat package version 0.69-0, URL http://www.omegahat.org/RSJava/.

Todeschini R, Consonni V (2002). *Handbook of Molecular Descriptors*. Wiley-VCH, Berlin.

Urbanek S (2006). ***rJava**: Low-Level R to Java Interface*. R package version 0.4-13, URL http://www.rforge.net/rJava/.

Versteeg R, Richardson A, Rowe T (2006). "Web-Accessible Scientific Workflow System for Performance Monitoring." *Environmental Science and Technology*, **40**(8), 2692–2698.

Vidal D, Thormann M, Pons M (2005). "LINGO, An Efficient Holographic Text Based Method to Calculate Biophysical Properties and Intermolecular Similarities." *Journal of Chemical Information and Modeling*, **45**(2), 386–393.

Wang R, Fu Y, Lai L (1997). "A New Atom-Additive Method for Calculating Partition Coefficients." *Journal of Chemical Information and Computer Science*, **37**(3), 615–621.

Ward J (1963). "Hierarchical Grouping to Optimize an Objective Function." *Journal of the American Statistical Association*, **58**, 236–244.

Willet P, Barnard J, Downs G (1998). "Chemical Similarity Searching." *Journal of Chemical Information and Computer Science*, **38**(6), 983–996.

Affiliation:

Rajarshi Guha
School of Informatics
Indiana University
Bloomington, IN 47408, United States of America
E-mail: rguha@indiana.edu
URL: http://cheminfo.informatics.indiana.edu/~rguha/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 18, Issue 5 *Submitted:* 2006-09-25
January 2007 *Accepted:* 2007-01-10