memory (and potentially execution time) can be saved.

- Since the algorithm falls into the "embarrassingly parallel" category, one can run several random forests on different machines and then aggregate the `votes` component to get the final result.

Acknowledgment

We would like to express our sincere gratitute to Prof. Breiman for making the Fortran code available, and answering many of our questions. We also thank the reviewer for very helpful comments, and pointing out the reference Bylander (2002).

Bibliography

L. Breiman. Bagging predictors. *Machine Learning*, 24 (2):123–140, 1996. 18

L. Breiman. Random forests. *Machine Learning*, 45(1): 5–32, 2001. 18

L. Breiman. Manual on setting up, using, and understanding random forests v3.1, 2002. http://oz.berkeley.edu/users/breiman/Using_random_forests_V3.1.pdf. 18, 19

T. Bylander. Estimating generalization error on two-class datasets using out-of-bag estimates. *Machine Learning*, 48:287–297, 2002. 18, 22

R. Shapire, Y. Freund, P. Bartlett, and W. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26 (5):1651–1686, 1998. 18

W. N. Venables and B. D. Ripley. *Modern Applied Statistics in S*. Springer, 4th edition, 2002. 19

Andy Liaw
Matthew Wiener
Merck Research Laboratories
andy_liaw@merck.com
matthew_wiener@merck.com

Some Strategies for Dealing with Genomic Data

by R. Gentleman

Introduction

Recent advances in molecular biology have enabled the exploration of many different organisms at the molecular level. These technologies are being employed in a very large number of experiments. In this article we consider some of the problems that arise in the design and implementation of software that associates biological meta-data with the experimentally obtained data. The software is being developed as part of the Bioconductor project www.bioconductor.org.

Perhaps the most common experiment of this type examines a single species and assays samples using a single common instrument. The samples are usually homogeneous collection of a particular type of cell. A specific example is the study of mRNA expression in a sample of leukemia patients using the Affymetrix U95A v2 chips Affymetrix (2001). In this case a single type of human cell is being studied using a common instrument.

These experiments provide estimates for thousands (or tens of thousands) of sample specific features. In the Affymetrix experiment described previously data on mRNA expression for approximately 10,000 genes (there are 12,600 probe sets but these correspond to roughly 10,000 genes). The experimental data, while valuable and interesting require additional biological meta-data to be correctly interpreted. Considering once again the example we see that knowledge of chromosomal location, sequence, participation in different pathways and so on provide substantial interpretive benefits.

Meta-data is not a new concept for statisticians. However, the scale of the meta-data in genomic experiments is. In many cases the meta-data are larger and more complex than the experimental data! Hence, new tools and strategies for dealing with meta-data are going to be needed. The design of software to help manage and manipulate biological annotation and to relate it to the experimental data will be of some importance. As part of the Bioconductor project we have made some preliminary studies and implemented some software designed to address some of these issues. Some aspects of our investigations are considered here.

Issues for consideration

Some of the issues that need to be considered when designing software that associates biological meta-data with experimentally obtained data are listed next. The list is numbered for further reference and the numbering scheme does not reflect importance.

1. the biological meta data are constantly evolving
2. there is some commonality across species
3. there is some commonality across measurement instruments
4. data (either experimental or annotation) may be sensitive and there may be a desire for privacy
5. for most species there is far too much meta-data available to easily provide access to all relevant data without some curation

It is worth distinguishing these meta-data from the meta-data that report information about the samples themselves, the experimental conditions and a variety of other experiment specific data. These data are imporatant and must also be dealt with. For the analysis of microarray data the MIAME standard Brazma et al. (2001) has been proposed and is being widely adopted. These data are being dealt with in the **Biobase** package and are attached directly to the expression data. We now turn our attention to the genomic or biological meta-data for the remainder of this paper.

The design

After careful consideration of the issues raised in Section 8 we settled on a design based on the R package system. The specific design is currently to provide a number of *data* items in the package. For each biological variable of interest (such as chromosome location or LocusLink identifier) a seperate hash table is constructed mapping from the manufacturer's identifier (Affymetrix in our example) to the quantity of interest. These hash tables are then placed in the 'data' directory of the package and documentation for them placed in the 'man' directory. Other software is added as needed.

There are many benefits to supplying meta-data in the form of R packages. These include:

- a version number mechanism
- a mechanism for supplying additional software that may be needed only in specific cases
- mechanisms for documentation (both help pages and vignettes)
- an organizational structure that simplifies automatic generation, distribution and documentation
- mechanisms that support the implementation of quality control processes (the `R CMD` suite of functions)

Given the number of organisms (distinct genomes) and the number of different measurement devices that may be used a robust and reliable indexing and distribution system will be required. Some of these issues have already been addressed within R for distributing packages and that system will be extended to provide support for projects such as the current one.

Since the meta-data are evolving and must be compiled from a variety of sources some form of versioning will be essential. Many biological data resources (such as the Gene Ontology Consortium and GenBank) provide version information that must be propagated to the end user. These data are easily included in the manual page that describes the particular data element in the distributed package. These data specific *build numbers* are essential so that end users can easily compare their results.

One of the major projects undertaken by the Bioconductor project is to enhance the repository management tools that were initially developed by K. Hornik and F. Leisch to provide services through CRAN. The software is currently available in the **reposTools** package available form the Bioconductor anonymous cvs archive. This package represents the collaborative efforts of K. Hornik, F. Leisch, V. Carey, J. Gentry and myself.

From the perspective of biological meta-data the concept of distributing data as R packages via a centralized repository provides many benefits. The data generator (or assembler) has a specific protocol (API) for updating their offerings and a specific mechanism for providing broad access to both new materials and older versions. On the client side, again there is a specific protocol and mechanism for obtaining specific meta-data (generally the most current). The processes that the client must carry out are entirely moderated through R. While other alternatives are possible it seems that the end-user will find it easier if they have a single system to learn for both data management and data analysis.

Among the objectives of the repository project are to enhance the version management (users may want to maintain multiple versions of particular libraries) capabilities of R, to enable the use of multiple repositories as suppliers of software and to allow users to become providers of software.

Once **reposTools** is included in R users will interact with the software through `install.packages` and `update.packages`. In the current implementation these have a `2` appended to keep them distinct from the system versions. **reposTools** con-

tains vignettes that will help guide interested readers through the steps needed to set up their own repository and to export it. More details on this project will be presented in later editions of R News.

Another important component of the approach taken is the assumption that the meta-data packages will need to be automatically generated and that there must be a mechanism that allows them to be updated. Autogeneration of packages is the only practical method for dealing with the large number of species and of technologies that need to be accomodated.

To facilitate this process we have developed the **AnnBuilder** package. The basic philosophy embodied in this package is to allow the package builder to supply the locations of a set of source data files. From these different variables are extracted and assembled into a large coherent set of data files. In this article we present an overview of this process and we refer the reader to Zhang et al. (2002, in preparation) for specific details.

In essence **AnnBuilder** supports the construction and distribution of custom annotation packages. The user specifies the locations of data resources, the mechanisms for extracting and combining the data and finally the required output. Specification of data resources can be URLs or local files. The user must also specify which fields should be extracted from which packages.

The data processing part of **AnnBuilder** extracts the specified fields from each file. The output may be to a database or in some cases to a file. The main purpose of this step is often data reduction. The primary source data is often much larger than is needed for any particular experiment. These tables are then processed (using a variety of tools including SQL and R) and the resultant unified mapping table can be exported as an R package, an XML data file (the DTD is) or as a database file that is suitable for querying by any means (including R, Perl, Python etc).

When multiple mappings or sources are available for a particular data item some form of reduction is needed. Reasonable suggestions include using the most common value (mode), nominating one source as trusted and using it if there are differences. **AnnBuilder** provides a general mechanism that allows the user to associated a SQL query statement with each variable to perform the selection. In most cases the function is simply the identity (since we have only one source for many of the quantities).

In many ways this sort of data assemblage is often required in other fields. There is nothing about **AnnBuilder** that is specific to biological data and it may be beneficial to consider the application of **AnnBuilder** in those areas. We would also like to consider slightly different models for data acquisition. As web-services become more common it will be important to extend the **AnnBuilder** concept to deal with data resources of this type.

For specific examples of the output of **AnnBuilder** the reader is referred to the Bioconductor Web site. Many different data packages are available by simply following the links. Interactions with the various Bioconductor packages are primarily mediated by the **annotate** package.

Discussion

We have considered some approaches to dealing with data complexity. By abstracting out both the platform (technology) and the species we have developed an annotation system that is practical and that can be maintained and extended.

While we have tried to make the data processing automatic it is not. The data are sufficiently complex and different to prevent complete automation. However, the data assemblage process is largely automatic and well documented. The requirements on both the server side and the client side are reasonably explicit and will be further refined as the projects evolve. We believe our efforts constitute a reasonable solution and have been using this methods for some time now.

The distribution of the resultant data is somewhat easier. Once appropriate names have been selected for the data packages they can easily be distributed and updated using R's repository system. The software in supplied in the **reposTools** package will need some testing but should provide an adequate basis for the distribution of these data packages.

By assembling data packages we also define an interface that can be used by other packages to access that data. In particular the functions in the package **annotate** are designed to be applied to any conforming experimental data and any conforming annotation package. The requirements placed on both the experimental data and the annotation package are documented and any package or data source that satisfies those requirements can be used.

It is rather interesting to close by reviewing some of the issues that have been raised in developing this paradigm.

- using http connections to access data on the web
- using DBI to interact with the data base that will do the heavy duty processing
- using XML to process the data
- automatic package generation
- using R resources to build in quality control procedures

We finish our discussion by reviewing the issues raised in Section 8. By using R packages we address issue 1 since the package system allows us to version

and update the data. The **AnnBuilder** approach addresses issues 2, 3 and 5. Data reduction is under the control of the data assembler. The notion of a package that generates packages is our method of dealing specifically with issues 2 and 3. The repository mechanism allows the data assembler to distribute the assembled data (they may choose to restrict distribution only to specific sites).

Acknowledgment

Robert Gentleman's work is supported in part by NIH/NCI Grant 2P30 CA06516-38.

Bibliography

Affymetrix. *Affymetrix Microarray Suite User Guide*. Affymetrix, Santa Clara, CA, version 5 edition, 2001. 22

A Brazma, P Hingamp, and J Quackenbush. Minimum information about a microarray experiment: towards standards for microarray data. *Nature Genetics*, 29:365–371, 2001. 23

Jianhua Zhang, Vincent Carey, and Robert Gentleman. An extensible application for assembling annotation for genomic data. *Bioinformatics*, 19, 2002. 24

Jianhua Zhang, Vincent Carey, A. J. Rossini, and R. C. Gentleman. Annbuilder - an open source application for genomic data annotation. *JSS*, in preparation. 24

Robert Gentleman
DFCI
rgentlem@jimmy.harvard.edu

Changes to the R-Tcl/Tk package

by Peter Dalgaard

Introduction

In a previous issue of R News, I gave a small tutorial on the basic use of the `tcltk` package for creating graphical user interfaces (Dalgaard, 2001). Since then, there has been a number of changes to the package, and the purpose of this paper is to outline the new possibilities offered by the modified interface.

Some of the changes were incompatible with old code. In particular, some of the examples in Dalgaard (2001) no longer run. The last section of the paper contains a revised version of the scripting widget.

Control variable changes

The old-style interface to Tcl variables allowed you to change the Tcl variable `foo` using syntax like

```
tclvar$foo <- "Hello, World"
```

With this approach, `foo` becomes a global Tcl variable, and there is no way to ensure that two R functions do not accidentally use the same variable name. (This is less of a problem in Tcl because there you can use techniques like prefixing the variable with the window name.)

Therefore, a new technique is introduced using the object class `"tclVar"`. The creator function for that class is `tclVar()` and it works by creating a new Tcl variable. You generally do not need to care about the names on the Tcl side, but they are simply `::RTcl1`, `::RTcl2`, and so forth (the `::` prefix ensures that they are in Tcl's global namespace). The accessor function is `tclvalue()` and a typical usage is like this:

```
foo <- tclVar()
tclvalue(foo) <- "Hello, World"
```

Notice that you have to create the variable before it can be used.

The `"tclVar"` objects are subject to finalization, so that when they go out of scope, the garbage collector will eventually remove their Tcl counterparts.

Tcl objects

The C interface to Tcl contains the notion of *dual ported* objects. All Tcl objects have a *string representation*, but they can also have a "dual personality" as doubles, integers, or lists of strings, doubles, or integers. (There are other possibilities but we ignore them here.)

This allows you to access Tcl variables without going via the string representation. The latter can be a major pain because of "quoting hell": The need to escape certain characters because they otherwise have special meaning to the Tcl interpreter.

The interface from R maps double, integer, and character vectors to Tcl lists of the corresponding object types. This works via an accesor function called `tclObj()`. An example should convey the gist of the interface: