IDA@SMU

SMU | Bobby B. Lyle School of Engineering

Progress Report

Research Experience for Undergraduates (REU) under Grant No. IIS-0948893

John Forrest

12/08/10

http://lyle.smu.edu/IDA/TRACDS/

This research is supported by the National Science Foundation under Grant No. IIS-0948893. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Contents

Introduction

The Extensible Markov Model (EMM) is a modeling technique that uses clusters as nodes in a Markov chain to represent temporal states (1). EMM Sequence Analysis (EMMSA) is the application of the original EMM technique to biological data of DNA or RNA sequences (thus, sequence analysis) (2). The input of EMMSA consists of counted sub patterns within the sequence data of a specified window length. The resulting model from EMMSA reveals the calculated structure for the data set depending on the distance metric specified, and a threshold value identified for the specific distance metric.

The intent of the research project this semester is two-fold: to create an interactive application where users can be introduced to EMMSA, and to create more attention for EMMSA. Currently, there is no single software package that encompasses all of the work that has been done for EMM: a significant amount of the code is in the R package rEMM (3), but there are also various Java software packages that have new components of research. In order to incorporate all of the latest research, we will have to create an application that is able to use the code bases in both R[1] and Java.

Initial Work

Background

Neither Vladimir nor I had any experience in either R or the Java code base, so a lot of time was spent initially on ramping up on the code and learning the new development patterns for R. We decided that Vlad would focus on the Java code base, while I would focus mainly on R because of another project that would expose me to the language.

[1] R is a programming environment for statistical analysis. It contains extensible support for data mining techniques that have been developed by other researchers. More information at http://www.r-project.org/.

The most significant challenge in designing the application was combining the fractured code base. The R code is much more mature and consists of a very stable build of the general EMM algorithm, while the majority of the work specific to EMMSA has been done in Java, so there was no escaping the fact that we would have to design an application that uses both languages.

Architecture

Figure 1 depicts an overview of the architecture. Users connect to our Java applet, which opens a TCP socket connection to Rserve. This socket connection is maintained while the applet is active. The commands that the user inputs are passed from the Java applet to Rserve, which in turn returns results back to the Java applet which are then displayed to the user.

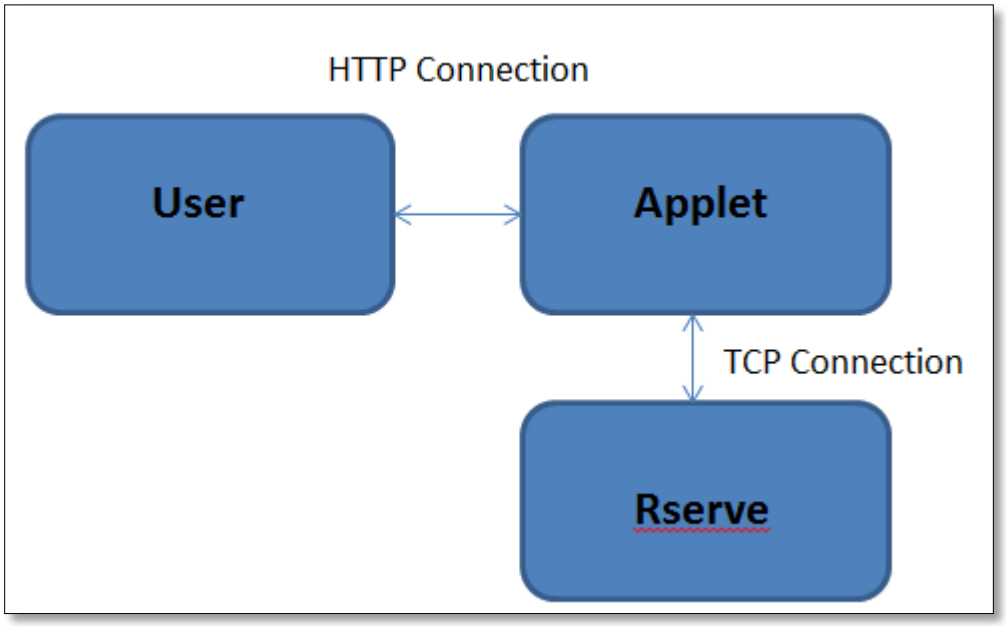


Figure 1: System Architecture

Java Process

I spent a lot of time researching creating a Process (4) object in Java that would be able to communicate with an R process and run the rEMM package with pre-defined input commands. This seemed like the right way to go, as it would take advantage of the easy threading capabilities in Java

and we would have a clear path to communicating with R. However, the more time I spent with this model, I realized many of its downfalls.

After closing the input stream to the Process object, the created R process would terminate and all of the variables created in the R workspace would be lost. I looked into creating a separate thread that would maintain the connection to the input stream and another thread that would listen for input to R and pass it on to the input stream – but this design quickly become cumbersome and difficult to handle. Additionally, once the output was generated by the R process, it would have to be parsed from a String into specific data types, which is also non-trivial because of all the different output expected.

Java/R Interface (JRI) (5)

My failed attempts at created an easy way to communicate from Java to R led me to discover that there was already a working implementation of exactly what I was trying to create: the Java/R Interface, or JRI.

JRI has been developed by an organization called RoSuDa, and is included in their software package rJava (which is essentially the opposite of JRI: it provides an interface from R to Java). It allows the developer to call R code directly in Java, and obtain the results in their corresponding Java form. For example, if a function in R returns a vector of numeric values, JRI allows you to obtain an array of Doubles that holds these values. This means that no parsing is involved with the R output.

Another advantage of the JRI is the fact that it communicates to R through a binary stream—no heavy lifting is necessary to send commands to R. Unfortunately, the JRI also requires an R installation on the host machine. This wouldn't be a problem if the software was running on our server, but then we would have to handle the management of workspaces for different users that connect at the same time—essentially we would have to write our own R server.

Rserve (6)

Once again, the solution I found was a software package developed by RoSuDa: Rserve. Rserve is a standalone server that listens for connecting clients and creates an individual R workspace for them. The R workspace is maintained while a TCP connection is maintained to Rserve, meaning that we would need something more than a stateless HTTP connection to the server if we want to preserve the variables created in the workspaces. The next section covers Rserve and how it's used in our design in more detail.

Final Design & Implementation

Introduction

In order to make greatest use Rserve, we needed a way to maintain a socketed TCP connection from the client to server. As mentioned previously, this isn't practical to do as a web service or a web page due to the fact that HTTP is stateless. Therefore, we had two routes to take: to develop a desktop application that users would be able to download and connect to our back-end, or to create a web applet that would have the same functionality within a browser.

Web Applet

Our team ended up deciding to develop a web applet. We decided to create an interactive web interface because experienced users are already able to download both rEMM and the Java software directly from the IDA website and use it on their personal computers. Web applets require no downloading or installation on the user's part, and they are able to interact directly with our application within a matter of seconds. The low barrier for entry allows users a quick an easy way to explore our algorithm, and if they are interested, they are able to follow up and download the comprehensive software packages after the demo.

Development

To develop the application, we started our own code base from scratch. This would allow us to focus on the components of the software that were absolutely essential, so the source code doesn't get bogged down with unused code or other paths of thought. We developed the Java code in

NetBeans, which has a built-in UI designer in which users can drag-and-drop UI items to create the interface. Although the drag-and-drop on the surface seemed like a good idea, the next section outlines some of the troubles that were faced during the creation of the UI.

In addition to designing the user interface, the communication between the Java process to the Rserve needed to be developed. My original intent was to send step by step commands to the Rserve, but Dr. Hahsler pointed out that it would be much more efficient to send small commands to the Rserve and let it do all the heavy-lifting server side.

The main tasks that the Rserve handles for us are: the sampling of the data and creation of a histogram, the actual creation of the EMM, and finally the creation of the models associated with the EMM. One problem that was encountered was how to identify unique connections, and provide them with the correct models. To do this, I assign each connection a unique universal identifier (UUID), and use that UUID to label all of the associated data for that connection (7).

All of the code is stored in an SVN (8) repository that we have been using for version control. Vladimir has been working on an interactive EMM model that he will discuss in his report. For a high level view of the implementation, please see Appendices A & B for the help documentation and screenshots.

Challenges

The greatest challenge of the project so far has been the user interface. Today's web environment demands user interfaces that work to attract the user's attention, but the capabilities for applet UI are just not there. Our team has little experience in developing UI in Java, and I spent a significant amount of time working to create our current UI that is functional, but not necessarily attractive.

The applet is embedded in an HTML page, which could be seen as advantageous, but the applet covers the majority of browser window (especially on small laptop screens), that it is difficult to add additional content to counter the drab look of the applet without cluttering the page.

I could list several more challenges about developing the UI of the applet but I will leave it at this—it is difficult. As I mentioned before, I was able to spend some time to design a UI that is usable, but not eye catching.

Next Steps

As with all software applications, there is always more work to be done. Below are a few of the next steps that need to be taken in the development of the application:

- The ability to upload external data
    - EMMSA counting
    - Smart sampling of the data for histogram creation
- The ability to download a comprehensive report

And of course,

- An eye-catching user interface

Works Cited

1. **Margaret Dunham, Michael Hahsler, Charlie Isaksson.** Extensible Markov Model (EMM). *IDA@SMU.* [Online] 2010. [Cited: December 12, 2010.] http://lyle.smu.edu/IDA/EMM/.

2. **Mallik Kotamarti, Margaret Dunham, Michael Hahsler.** EMM Sequence Analysis (EMMSA). *IDA@SMU.* [Online] 2010. [Cited: December 12, 2010.] http://lyle.smu.edu/IDA/EMMSA/.

3. **Hahsler, Michael and Dunham, Margaret.** rEMM: Extensible Markov Model for Data Stream Clustering in R. *Journal of Statistical Software.* [Online] July 5, 2010. [Cited: October 17, 2010.] http://www.jstatsoft.org/v35/i05/.

4. Class Process. *Java API.* [Online] Oracle, 2010. [Cited: December 12, 2010.] http://download.oracle.com/javase/6/docs/api/java/lang/Process.html.

5. JRI - Java/R Interface. *RForge.net.* [Online] RoSuDa, 2010. [Cited: December 12, 2010.] http://www.rforge.net/JRI/.

6. Rserve - Binary R Server. *RForge.net.* [Online] RoSuDa, 2010. [Cited: December 12, 2010.] http://www.rforge.net/Rserve/.

7. Universally unique identifier. *Wikipedia.org.* [Online] Wikipedia, 2010. [Cited: December 12, 2010.] http://en.wikipedia.org/wiki/Universally_unique_identifier.

8. Subversion. *Apache.* [Online] Apache, 2010. [Cited: December 12, 2010.] http://subversion.apache.org/.

Appendix A: Help Documentation

General Help

Introduction
The EMMSA applet has been designed to demonstrate the models created by the EMMSA algorithm, created here in the Intelligence Data Analysis (IDA) group at SMU. Each of the help links will go into more detail about the specific tab they are named after. The FAQ covers questions that come up while using the EMMSA applet.

Before getting started, it is important to note that the applet is still in development. Some of the features that appear on the UI are not yet functional.

The current non-working features include:

- Uploading data sets
- Downloading a comprehensive report of the experiment run

Our team is working hard to complete these features, so please check back in the near future for their addition.

Application Flow
On each page you will see a '?' button that will display detailed help for each action that is available on the tabs. Pressing the help button on the welcome tab will bring you to this help page. Additional, each page has a button that corresponds to the next tab, and pressing it will advance you through the application.

In just 3 steps with the EMMSA applet, you are able to create various models for DNA and RNA sequences:

1. Start by navigating to the Choose Data tab. Here you can choose example data or upload your own.
2. Next, navigate to the Choose Parameters tab. On this tab you are able to define the parameters for the algorithm. Recommended parameters have been pre-selected for you based on the data distribution.
3. Finally, click on Generate Models using the parameters you have defined. This will generate the models, and automatically navigate the applet to the Explore Models tab.

After generating the models, feel free to explore them and download a comprehensive report that has detailed information about the data, and all of the models that have been generated during that specific run.

Choosing Data

Introduction

This tab allows users to pick the data they would like run the algorithm on. Currently, users are only able to use example data that has been pre-counted by the EMMSA algorithm using triplets, with a window length of 100 without any overlap.

The two example data sets that are available are Alphaproteobacteria and Mollicutes, which are both from the 16S ribosomal RNA.

Uploading Data

Currently this feature isn't complete. In the future, users will be able to upload their own dataset (most likely in the FASTA format), and will be able to choose additional parameters for the EMMSA counting algorithm to pre-process the data.

Choosing Parameters

Introduction

By choosing different input parameters for the algorithm, the user is able to manipulate how the data is going to be processed, and in turn, the outcome of the generated models. The first label under the title displays the data set that has been selected on the previous tab.

Underneath that label, the user is able to choose a dissimilarity measure, and what threshold value to use for the measure.

Dissimilarity Measures & Threshold

The dissimilarity measure alters how the data points are grouped together, in other words, clustered.

A smaller dissimilarity measure means that the data points have to be closely related in order to be clustered together. Similarly, a large dissimilarity measure causes data points that aren't closely related to be clustered together.

These definitions correspond to the labels on the threshold slider: a smaller dissimilarity measure will produce more clusters (the data points are seen as unrelated, so new clusters are created to accommodate this), and a larger dissimilarity measure will produce less clusters (the data points are seen as related, so they are simply placed into existing clusters).

The available dissimilarity measures are:

- Euclidean
- Squared Euclidean
- Jaccard
- Kullback (all data +1)
- Cosine
- Manhattan

You may have seen some of these defined as similarity measures—however, in this application we are converting all of the measures to dissimilarity by subtracting them from 1 if needed.

Histogram
Upon choosing a new data set or new dissimilarity measure, a new histogram of the data is created. The histogram is based upon the dissimilarities between the data points. The default value of the slider is chosen based upon the 10^{th} quantile of the generated histogram.

Exploring Models

Introduction
This tab shows the output of the algorithm performed on the data set selected, and the parameters selected. The main idea here is to visualize the output in a way that the user can understand, and easily follow patterns in the data.

There are four options to choose from on this page:

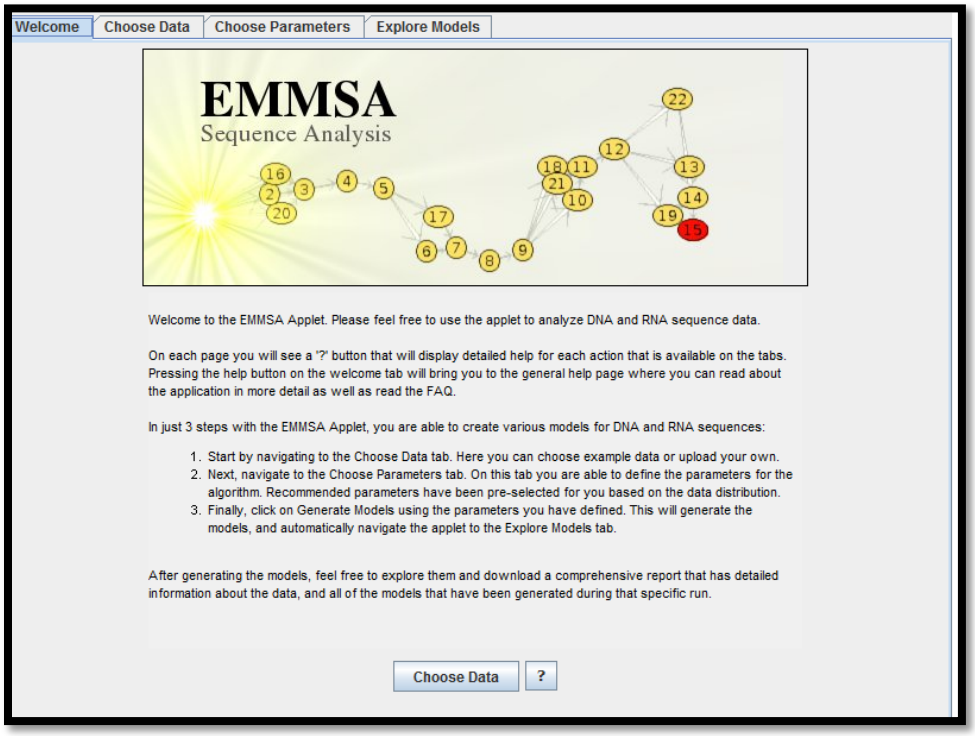
R Output
Shows the raw output received from R after creating the EMM.

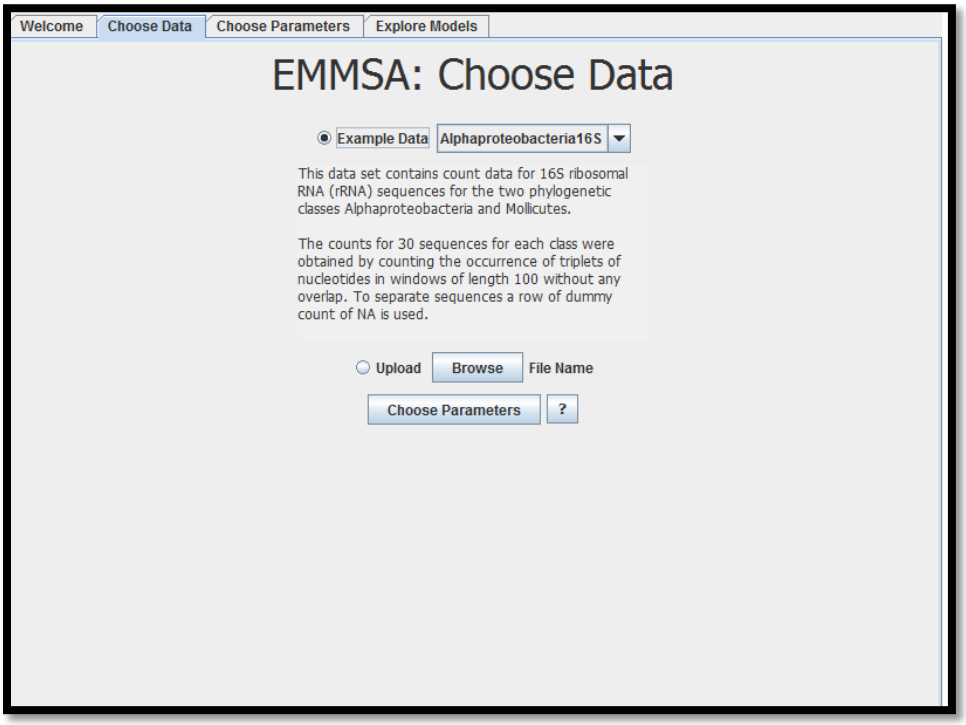
Cluster Similarity Model
Graphs the EMM states based upon their dissimilarity. In this model, the closer the clusters are together, the more similar they are. The thickness of the arrows also shows how likely the transitions between the states are.

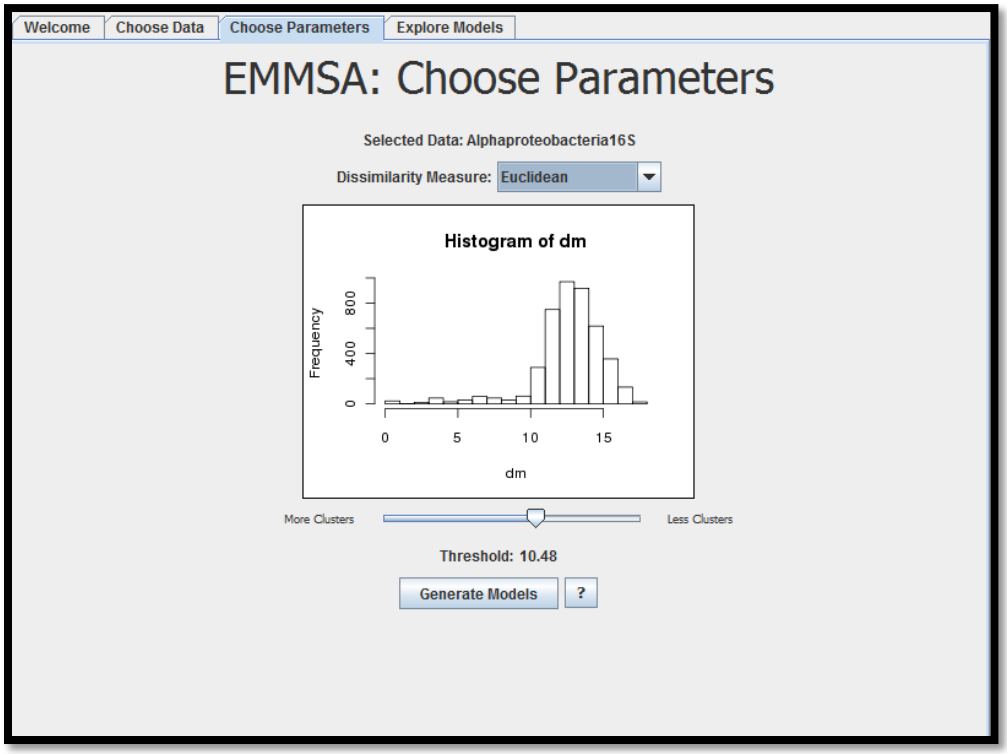
Conservation Model
Is the model generated by Graphviz, that graphs the EMM in a way that is pleasing to the human eye. It demonstrates clearly the structure of the EMM, and how the transitions occur throughout the sequence.

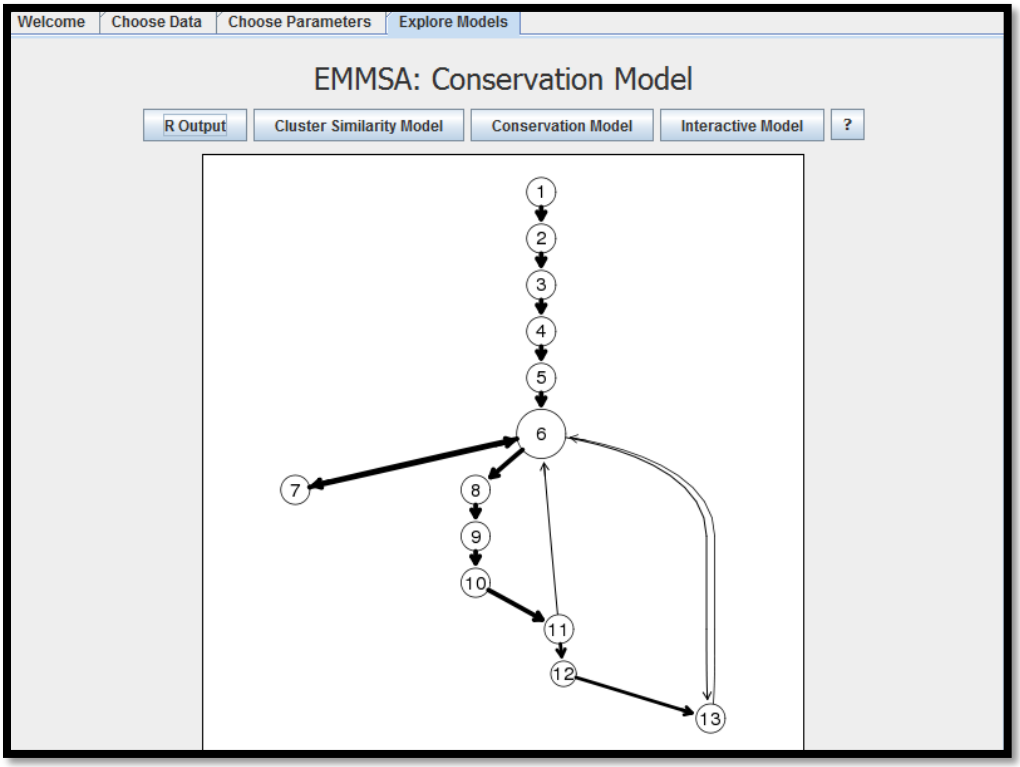
Interactive Model
This model is an interactive panel where the user can manipulate the EMM by pulling on the various states.

Appendix B: Screenshots






Welcome
Choose Data
Choose Parameters
Explore Models
EMMSA: Choose Parameters
Selected Data: Alphaproteobacteria16S
Dissimilarity Measure:
Euclidean
Histogram of dm
Frequency
0
400
800
0
5
10
15
dm
More Clusters
Less Clusters
Threshold: 10.48
Generate Models
?


Welcome
Choose Data
Choose Parameters
Explore Models
EMMSA: Conservation Model
R Output
Cluster Similarity Model
Conservation Model
Interactive Model
?
1
2
3
4
5
6
7
8
9
10
11
12
13

Appendix C: Installation

This section covers the installation that we are currently using for both the applet and Rserve.

Applet
The installation for the applet is rather straightforward. No specialized software needs to be available on the server – most modern browsers are able to interpret HTML tags that identify the applet and its resources, and run the applet if the user has the necessary Java installation. If the user does not have Java, or the correct version installed, they will be prompted to install the latest version of Java.

The corresponding HTML tag used on our landing page is:

```
<applet code="EMMSAApplet.EMMSAApplet.class" archive="EMMSAApplet.jar, REngine.jar,
Rserve.jar" align="baseline" height="600" width="800">
</applet>
```

The `code` attribute defines the applet class, which will automatically be run when the page is loaded. The `archive` tag defines all of the .jar files that are needed to run the applet; this includes both the .jar that contains the applet .class file, and any .jar files that are required libraries. Finally, the last three attributes are used simply to define the height, width, and alignment of the applet on the loaded web page.

It is important to note that the HTML code has to have an opening tag and a closing tag (`<applet>` and `</applet>`) instead of using a single tag: `<applet/>`. This may be a bug in the way browsers currently interpret the HTML tags, but it will prevent the applet from running if it is not written in the specified manner.

Rserve
Rserve comes in 2 flavors: a standalone application, or an R plugin that requires a working installation of R on the server. We decided to go with the latter option, because our server already includes an installation of R and uses other external packages such as rEMM. The package can be installed with a single R command:

```
install.plugins("Rserve", depend=TRUE)
```

Because the communication between the Rserve and the Java applet is over a socketed connection, the system also requires a port to be opened on the server to allow this connection. We used the default Rserve port of 6311.

Additionally, to secure our server if a malicious user was to discover a buffer over or exploit the fact that we have an Rserve running openly over port 6311, a new Unix user was created specifically to run these applications and has had its read/write access strictly limited to the folders that are necessary. Detailed Rserve documentation is available at: http://www.rforge.net/Rserve/.

Folder Structure

The final structure of the folders where the system is running is also important because many of the references in both the R code and Java code refer to their relative paths. The structure is as follows (other folders that exist but aren't needed have been omitted):

public_html/
- applet
  - **help**
  - **histograms**
  - **models**
  - **reports**
  - emmsa.html
  - index.html (alias for emmsa.html)
  - EMMSAApplet.jar
  - REngine.jar
  - RServe.jar
  - EMMSALogo.jpg
- **EMMSA** (alias for applet)
- **R** (contains the R installation)
  - startup.R (custom startup script)