Tree Representations of XML and JSON Data Formats with an Implementation in R

Xiaotian Dai[*] Jürgen Symanzik[†]

Abstract
Extensible Markup Language (XML) and JavaScript Object Notation (JSON) are widely used in web applications for data storage. Both, XML and JSON documents, have a hierarchical data structure and can be summarized as a tree structure, called XML tree or JSON tree. A tree data structure represents a hierarchical data structure with a set of linked nodes. It is important for data scientists to understand the node structure and the relationship among the nodes within XML and JSON documents. The tree representation can provide an intuitive visualization of the data structures. Based on this idea, in this article, we will discuss the implementation of XML trees and JSON trees in the R software environment.

Key Words: Extensible Markup Language; JavaScript Object Notation; Hierarchical Data Structure; Visualization.

1. Introduction

Both, Extensible Markup Language (XML) and JavaScript Object Notation (JSON), are widely used in web applications for data storage. Many datasets can be serialized and stored in either format. XML and JSON have their own features and applications. However, there are also similarities in their syntaxes.

An XML document must contain a root node that is the parent node of all the other nodes. All nodes in an XML document can contain child nodes, text, and attributes. The structure of an XML document starts at the root node and branches to the lowest level of the nodes. Although JSON has a different set of notations and formatting rules, the structure of a JSON document is also based on parent-child relationships. Hence, both XML and JSON have a hierarchical structure, which can be interpreted as a tree structure. Tree representations can be useful for the visualization of hierarchical data structures. There already exist some free JSON tree or XML tree viewers online, such as jQuery4u.com (2013). Currently, all of these tree viewers are built in programming languages other than R (R Development Core Team, 2013), such as JavaScript. The research goal presented in this article is to create tree plots in R to make a simple tree representation of the structure of a JSON or XML document.

In this article, we will present two functions developed for the tree representation: **treeplot** and **print.treeplot**. In Sections 2 and 3, we will discuss the implementations of XML tree and JSON tree, respectively. We will finish with our conclusion in Section 4. More details and additional examples are provided in Dai (2013).

[*]Utah State University, Department of Mathematics and Statistics, 3900 Old Main Hill, Logan, UT 84322, USA. Phone: (435)754-4980, Fax: (435)797-1822, E–mail: `xiaotian.dai@aggiemail.usu.edu`

[†]Utah State University, Department of Mathematics and Statistics, 3900 Old Main Hill, Logan, UT 84322, USA. Phone: (435)797-0696, Fax: (435)797-1822, E-mail: `symanzik@math.usu.edu`

2. Implementation of XML Tree

XML documents are in a format that is both human-readable and machine-readable. They are created to store and transport data and information. By definition, an XML document is a string of characters. The characters making up an XML document are divided into markup and content, which may be distinguished by the application of simple syntactic rules (Bray et al., 1997). Generally, strings that constitute markup begin with the character "<" and end with a ">". These strings are called tags. There are start-tags and end-tags. The strings in the tags are the "nodes", i.e., they can be referred to as the names of the variables. The end-tags start with "/" and the strings in the end-tags must match those in the start-tags. Between the start-tag and the end-tag, there can be one value or other child nodes. As introduced in Section 1, the structure of the XML object starts at the root node and branches to the lowest level of the nodes. If a node consists of other identifiers within a start-tag, the contents within the start-tag are describing the attributes of this node.

Due to the popular use of XML databases, many programming languages have built-in functions to aid software developers with the processing of XML documents. The **XML** package (Temple Lang, 2012) in R contains built-in functions for handling data in XML format. It provides the parser to read in XML documents and store the data in R as a list data type. Then, we can use two newly developed functions from Dai (2013), **treeplot** and **print.treeplot**, to skecth a tree plot of the data structure.

The tree plot can be organized similarly to the original XML document, in which keys of the same level will be placed around one vertical line and the child nodes will be placed on the lower right side of its parent node. A data frame is used to store the information of this tree structure temporarily. The function **treeplot** will take an XML list data type as the input argument and it returns such a data frame. This data frame contains the location information of the nodes to be placed in a plot. We just need some lines to connect the nodes and sketch a tree. The function **print.treeplot** can finish this job and produce a sketch of the XML tree. This function is built based on the S3 object-oriented programming models and the data frame with class "treeplot" can be inherited by the generic function **print()**, which means users of the function can call the function **print** directly instead of **print.treeplot**. The user can even hide the **print.treeplot** function by just calling **treeplot**, as shown in the following examples.

```
<doc><part><name>ABC</name><type>XYZ</type><cost>3.54</cost>
<status>available</status></part></doc>
```

Figure 1: Simple XML document: "simple.xml"

For example, the string in Figure 1 is a simple XML document used in the documentation of the **XML** package, called "simple.xml". It is not a well-structured XML document, and we cannot easily distinguish the parent-child relationships within the nodes. A tree plot would be helpful for understanding the internal structure of this document. Before sketching the tree plot, the functions **xmlParse** and **xmlToList** from the **XML** package were to be called to convert the XML document into an R list, as shown in Figure 2. Then we can sketch the tree plot.

```
> library(XML)
> xmlData = xmlParse("<doc><part><name>ABC</name><type>XYZ</type>
+                      <cost>3.54</cost><status>available</status></part></doc>")
> simple.xml = xmlToList(xmlData)
```

Figure 2: R expressions required to parse and convert an XML document into an R list

```
> treeplot(simple.xml)
```

```
 ROOT
   |
   |    - - - -  part
                   |
                   |    - - - -  name
                   |
                   |    - - - -  type
                   |
                   |    - - - -  cost
                   |
                   |    - - - - status
```

Figure 3: Tree plot for “simple.xml” (shown in Figure 1)

As shown in Figure 3, the node “part” is the child of the root node. The other four nodes, “name”, “type”, “cost”, and “status”, are parallel and the children of “part”.

This is only a simple implementation of the XML tree. There are also other special features in XML documents. As introduced earlier in this section, if a node consists of other variables within the start-tag, the contents within the start-tag are describing the attributes of this node. Attributes often provide information that is not a part of the data. The XML object in Figure 4 is an example provided by w3schools (2013), called “note.xml”. The nodes “day”, “month”, and “year” are the child nodes of “date”, and “date” is one of the child nodes of the root. We can sketch the structure of “note.xml” using **treeplot** function, as shown in Figure 5.

The XML object in Figure 6 contains exactly the same information as “note.xml”. However, the three nodes under “date” are placed in the first start-tag. They become the attributes of the root node.

The tree plot in Figure 7 is generated from “note.xml” with attributes. We can take a closer look at the differences in the tree structures in Figure 5 and 7. The tree plot shown in Figure 7 uses the string “attrs” to represent attributes. The node “attrs” is under its parent node, the root node, and it is drawn in parallel to the other child nodes of the root node.

```
<note>
    <date>
        <day>10</day>
        <month>01</month>
        <year>2008</year>
    </date>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

Figure 4: XML document: "note.xml"

```
> treeplot(note.xml)

 ROOT
   |
   |    - - - -  date
   |               |
   |               |    - - - -   day
   |               |
   |               |    - - - -  month
   |               |
   |               |    - - - -  year
   |
   |    - - - -   to
   |
   |    - - - -  from
   |
   |    - - - - heading
   |
   |    - - - -  body
```

Figure 5: Tree plot for "note.xml" (shown in Figure 4)

3. Implementation of JSON Tree

As another example of a hierarchical data format, the structure of a JSON document can also be summarized as a tree plot using the **treeplot** and **print.treeplot** functions.

Similar to XML, JSON is also a human-readable and machine-readable format. A JSON document is a collection of variable and value pairs, and each pair can be referred to as a node. A colon separates the variables from the values, and a comma separates the nodes. The variables are strings and the types of value presented in JSON can be strings, numbers, Booleans, object, arrays, or even NULL. The variables are wrapped in quote marks. The data wrapped in curly braces or square brackets after a colon represent lower level nodes included in a higher level node. We can often read a JSON document if it is well organized.

The JSON example in Figure 8 is called "menu.json" and has been provided by JSON (2013). The node "menu" is the parent node of "id", "value", and "popup". The node "popup" has one child, "menuitem", and "menuitem" has two other child

```
<note day="10" month="01" year="2008">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

Figure 6: XML document: “note.xml” with attributes, called “note.attrs.xml”

```
> treeplot(note.attrs.xml)

 ROOT
   |
   |    - - - -   to
   |
   |    - - - -  from
   |
   |    - - - - heading
   |
   |    - - - -  body
   |
   |    - - - -  attrs
                  |
                  |    - - - -   day
                  |
                  |    - - - -  month
                  |
                  |    - - - -  year
```

Figure 7: Tree plot for “note.xml” with attributes (shown in Figure 6)

nodes, “value” and “onclick”. This is how this JSON object starts at the root node and branches to the lowest level of the nodes.

Generally, JSON is a natural presentation of data, and it is easy for computers to generate and parse (Crockford, 2006). There exist open-source JSON libraries in R, such as **rjson** (Couture-Beil, 2012) and **RJSONIO** (Temple Lang, 2011). The functions included in the **RJSONIO** package are built on the C programming language, which can significantly speed up the processing of the data. More importantly, the **fromJSON** function in **RJSONIO** can directly convert a JSON document into an R list. Hence, we can easily create a tree representation of the JSON document introduced in Figure 8 (shown in Figure 9) using the **treeplot** function.

The **treeplot** function works in the same way for JSON as for XML. When working with JSON documents, the function will take in a JSON list data type as the input argument and it returns a data frame. This data frame contains the location information of the nodes to be placed in a plot. The function **print.treeplot** will connect the nodes and produce a sketch of the JSON tree. The tree plot in Figure 9 shows the structure of the JSON object, which is the same as what we read from the data.

```
{"menu": {
    "id": "file",
    "value": "File",
    "popup": {
        "menuitem": [
            {"value": "New", "onclick": "CreateNewDoc()"},
            {"value": "Open", "onclick": "OpenDoc()"},
            {"value": "Close", "onclick": "CloseDoc()"}
            ]
    }
}}
```

Figure 8: JSON document: “menu.json”

```
> treeplot(menu.json)

  ROOT
   |
   |     - - - -   menu
                    |
                    |     - - - -    id
                    |
                    |     - - - -   value
                    |
                    |     - - - -   popup
                                     |
                                     |     - - - -  menuitem
                                                      |
                                                      |     - - - -   value
                                                      |
                                                      |     - - - -  onclick
```

Figure 9: Tree plot for “menu.json” (shown in Figure 8)

```
<XML>
    <menu id="file" value="File">
        <popup>
            <menuitem value="New" onclick="CreateNewDoc()" />
            <menuitem value="Open" onclick="OpenDoc()" />
            <menuitem value="Close" onclick="CloseDoc()" />
        </popup>
    </menu>
</XML>
```

Figure 10: XML document: “menu.xml”

```
> treeplot(menu.xml)

  ROOT
   |
   |     - - - -    menu
                      |
                      |    - - - -   popup
                      |                 |
                      |                 |     - - - -  menuitem
                      |                                   |
                      |                                   |     - - - -   value
                      |                                   |
                      |                                   |     - - - -  onclick
                      |
                      |    - - - -   attrs
                                        |
                                        |     - - - -     id
                                        |
                                        |     - - - -   value
```

Figure 11: Tree plot for “menu.xml” (shown in Figure 10)

Alternatively, the same information as in Figure 8 can be expressed in an XML document, shown in Figure 10. The document contains the same nodes and the same parent-node relationships as the JSON document in Figure 8, except for “id” and “value”. “id” and “value” are the child nodes of “menu” in “menu.json”, but they are attributes of “menu” in “menu.xml”. Other than this difference in the structures, “menu.json” and “menu.xml” contain exactly the same data and information.

If we apply a tree representation to this XML document, we will end up with a tree plot similar to the one in Figure 9, except for the attribute nodes, as shown in Figure 11.

4. Conclusion

The functions **treeplot** and **print.treeplot** introduced in this article work in the same way for both tree-structured data formats, XML and JSON. The functions use a recursive algorithm to detect each branch of the tree, from the root node to the child nodes at the lowest level. Hence, if documents in these two formats contain the same nodes and parent-child relationships, the functions **treeplot** and **print.treeplot** will return the same tree plot. Although XML and JSON are using different notations, they can be used to express exactly the same data and information. Actually, a recursive algorithm can be applied to translate between these two types of data formats (Wang, 2011).

There have only been a few simple examples presented in this article. Nevertheless, the functions **treeplot** and **print.treeplot** have been developed for a general purpose. These functions can be used for any valid XML and JSON document. The same approach could also be applied to other tree-structured data formats, or, say, data formats with a hierarchical database model.

References

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F., 1997. Extensible Markup Language (XML). World Wide Web Journal 2 (4), 27–66.

Couture-Beil, A., 2012. rjson: JSON for R. R package version 0.2.8.

Crockford, D., 2006. JSON: The Fat-free Alternative to XML. URL: `http://www.json.org/fatfree.html`.

Dai, X., 2013. Processing and Manipulation of Data Collected from the Educational On-line Game Refraction. Master's thesis, Utah State University, Department of Mathematics & Statistics, Logan, Utah.

jQuery4u.com, 2013. Online JSON Tree Viewer Tool.
URL `http://www.jquery4u.com/demos/online-json-tree-viewer/`

JSON, 2013. JSON Example.
URL `http://json.org/example.html`

R Development Core Team, 2013. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, ISBN 3–900051–07–0 (`http://www.R-project.org/`).

Temple Lang, D., 2011. RJSONIO: Serialize R Objects to JSON, JavaScript Object Notation. R package version 0.95-0.

Temple Lang, D., 2012. XML: Tools for Parsing and Generating XML within R and S-plus. R package version 3.9-4.

w3schools, 2013. XML Attributes.
URL `http://www.w3schools.com/xml/xml_attributes.asp`

Wang, G., 2011. Improving Data Transmission in Web Applications via the Translation between XML and JSON. In: 2011 Third International Conference on Communications and Mobile Computing (CMC). IEEE, pp. 182–185.