Arbor Workflows Documentation

Release None

Kitware, Inc.

February 22, 2017

Contents

Please visit the Arbor Workflows homepage or the GitHub repository for more information.

CHAPTER 1

User's Guide

Advanced Mode

See the following sections of the TangeloHub documentation for using the Advanced Mode interface available at https://arbor.kitware.com.

Creating an analysis

Creating a workflow

OpenTree Integration

Arbor contains a set of analyses designed to provide connectivity to the Open Tree of Life (opentreeoflife.org). These analyses provide taxonomy lookup and subtree access services. The goal is to provide techniques for users to access open tree data without having to explicitly program using the OpenTree API calls. Instead, the API calls are embedded inside Arbor workflow steps.

These operations are currently in prototype form and should be tested and adjusted as the OpenTree API and Arbor's functionality continue to evolve.

Taxonomy Analyses

Explore OpenTree species name completion

This analysis lets users provide a comma-separated list of partial names (name prefixes) that will be used to query the OpenTree taxonomy. A table is returned that provides all the values returned as attempts to complete the queries partial names. This uses the "autocomplete" function of OpenTree's API.

Verbose OpenTree TNRS return for names

This analysis is written to allow for exploring the OpenTree TNRS data return. It uses the OpenTree v2 API "match_names" interface. A single column table, or a table with the names in the first column, is expected. All names are extracted from the table and a single query of the name list is performed. A table is built containing the full taxonomy lookup results.

Lookup names using OpenTree auto completion

This operation assumes a single column table, or a table where the first column is the species of a matrix. Each name in the column is used to query the OpenTree "autocomplete" API function in order to determine a matching node in the Tree of Life. Currently, only the first taxonomic return is examined and used if it is indicated as a taxon (or leaf node) in the tree. Therefore, this analysis can be used to identify OpenTree taxa that correspond to a user's character matrix.

Lookup names using OpenTree Taxonomy

This operation is similar to the "Lookup with auto completion" described above, except this operation uses OpenTree's v2 "match_names" API call. The match_names API call does not currently delineate between taxa and non-taxa, so an attempt has been made, at the Arbor level, to filter returns for only taxa by examining the attributes returned by OpenTree. This methodology should be reviewed before extensive use.

Tree Operations

Return the OpenTree subtree from a node list

Given a list of OpenTree node IDs (OttIds), return a tree that contains these taxa, and only these taxa. This analysis calls OpenTree's SubtreeForNodes API endpoint. The output of this function is the returned tree in Newick format.

Table Operations

Arbor is designed to make it easier to perform comparative analysis on tabular and tree datasets without the need for custom programming. Users of Arbor are able to invoke operations on their datasets one at a time, or collect operations into a workflow to be executed by Arbor. The following is a list of table operations currently available in Arbor. Additional operations will be added as the Arbor system continues to develop.

Row Operations

Append Rows

This merges two tables which have the same column headers, but different content rows. The analysis should look for all headers in either input file and output the union of the individual column header fields. If values are sparse (missing some columns in some rows), let it be sparse initially. This can be used to "merge" tables together, when the attribute columns match.

Select Random Rows

Select a random subset of the input rows to pass through. Input is a table and an integer value indicating how many rows are desired in the output table. The algorithm selects output rows randomly from the entire length of the input table. This can be used to generate samples out of a large trait matrix, to test a workflow with a smaller matching tree, for example.

Drop Rows by Value

Pass all rows of an input table through the analysis unless a particular test criteria is met. If the test is met, this row will be "dropped" from the output table. The criteria is specified by specifying a column header name and a comma separated list of values to look for. For example, drop a row from a character matrix, where the "species" entry is "anolis occultus". This will drop the species from the matrix but pass the other species values through.

Aggregate table by average

A table algorithm that inputs a table and a column name to use as a "class" identifier. It is assumed that the class column will have a finite number of categorical values, even if they are numerical in nature. The algorithm will generate a table output with only one row per class value. All the attribute values for each individual row from a particular class will be aggregated. To illustrate, consider an example with a table containing 100 observations across a number of islands. An "Island" column is included, where rows have the the value "Cuba", "Puerto Rico", or "Hispaniola" – representing three "classes" of observations. If aggregated by "Island", the output table will have three rows and all the continuous attributes in the output table will be the average of all rows that contributed to each corresponding class. This is sometimes called a "Group By" operation.

Aggregate table by max

This is the same operation as "aggregate table by max", except the maximum value observed for each class is returned instead of the average value of all class members.

Column Operations

Append Columns

This analysis appends columns of two datasets together. One "index" column is named and this column is used as the primary key to bring together all column entries from each table together into a single row record. For example, lets say we have a 1st table indexed by a scientific name with 3 attribute columns and a 2nd table indexed by the same scientific name, with 2 additional attribute columns (lets call them attribute 4 & 5). This analysis will output a five column table, where the index column (scientific name) has been used to correctly merge attributes for each corresponding row together.

Extract Columns

Accept an input table and a list of column header names. Only the columns contained in the header list are passed through to the output. Therefore, this is a way of keeping only the most important columns in a table instance. The column selection input is organized as a single-column table, because it will be easy to read as a table, also.

Extract Columns by string

The same algorithm as above, but the user can enter a list of columns to extract as a comma separated list when the algorithm is run. This is useful for interactively working with a table a step at a time.

CHAPTER 2

Tutorials

Demonstration Videos

Here are some short demonstration videos to illustrate some of the features of the Arbor web interface:

Introduction

An Introduction to the Arbor Interface

Creating a New Analysis

Making a new analysis with the Arbor interface

Creating a Workflow

How to create and execute a multistep workflow in Arbor

OpenTree Integration

In the link below, we use Arbor analyses that invoke the OpenTree API. This integration technique makes it possible to query the OpenTree for taxonomy lookup and subtree extraction without having to write any program code. The algorithms performed in the Arbor analyses need to be yet refined to produce useful phylogenetic results:

Simple examples of OpenTree integration with Arbor

LifeMapper Integration

Below is a link to preliminary work to request species occurrence points from LifeMapper in order to examine phylogenetic and species range information together:

Prototype connections to LifeMapper for species occurrences

CHAPTER 3

Administrator's Guide

Installation

TangeloHub Installation

Arbor Workflows uses Flow, developed by Kitware, as its main interface. Follow the Flow Vagrant install instructions with the Ansible `arbor` setting set to `true` (the default) to spin up your own instance.

Easy Mode App Installation

The easy mode applications must be added through an additional repository checkout and a few symbolic links. To set up these links, first enter your Vagrant VM with:

```
vagrant ssh
```

Once inside your virtual machine, checkout the ArborWebApps repository to the `/vagrant` directory, which will be exposed on your host machine.

```
cd /vagrant
git clone https://github.com/arborworkflows/ArborWebApps.git
cd app
ln -s ../ArborWebApps/phylogenetic-signal .
ln -s ../ArborWebApps/ancestral-state  .
```

Now you will also need to import the "Phylogenetic signal" and "Ancestral state" analyses from the public Arbor at http://arborclassic.arborworkflows.com by selecting the analysis and clicking Download, then uploading them on your instance with the Upload button.

Indices and tables

- genindex
- search