MARi Predictive Research Sandbox

Making Sense of Large Scale User Model Data

Summary

The MARi Personal Learning Cloud operates as a large user modeling server and contains thousands of personal attributes that describe individual learners. Virtually any learning application or assessment can share observations about individual learners with MARi, which in turn shares those observations with other applications that want to adapt their learning experiences to the specific needs of each user. However, with so many potentially observable attributes available, it is unlikely that any one individual will play enough educational games, complete enough courses, or take enough assessments to generate direct observations of all the attributes available in MARi user models. Luckily, it is often possible to predict the values of many attributes based on the observed values for others. For instance, if a learner has demonstrated mastery of basic algebra, it is fairly likely that that same learner has mastered simple mathematics operations like addition, subtraction, multiplication, and division. The MARi Research Sandbox provides all the tools necessary to support research organizations that wish to create new predictive models to determine appropriate values for unobserved attributes based on previously observed value associated with related attributes. Researchers can use the ubiquitous web-based programming language JavaScript to construct and test models in a private experimental workspace on MARi's servers using a web-based Integrated Development Environment (IDE). This document describes

 © 2013 MARi, LLC

Introduction

The MARi Research Sandbox is a place to explore and experiment using predictive analytics over a large dataset of real-time, real-world personal attribute values.

As a sandbox, each research team is provided with a separate logical area on the MARi servers in which to gather data, build executable programs, run experiments, and validate their results. MARi will provide sample data sets along with a series of tools that can manipulate that data, but the sample data remains on MARi servers at all times. Researchers may also upload and store external (non-MARi) files

What is a project?

A project is an organizational area in the Research Sandbox that allows a research team to develop predictive models and test them on anonymized, real-world sample data in a controlled environment.

Primarily a project is composed of five components:

1. A collection of anonymized user data spanning a selection of PAs (as defined by the model author); this data is frozen for use in testing
2. One (or more) ontological models for reasoning during prediction (as defined by the model author, or shared from another source)
3. An entry-point script (main) that must conform to a particular I/O specification, and optionally any number of additional scripts that are instantiated from main
4. A configuration definition file with accompanying default configuration values
5. A “walled-garden” of disk space, where any arbitrary files (presumably externally produced data files) can be stored and accessed by scripts

The components of a project “compile” into a model with an automatically iterated version number. The model can then be published or shared as desired by the author.

Starting a Project

When a new project is started, the user is presented with a form that requires basic information about the project (e.g., name) and importantly asks the user to specify a list of PAs that the model will interact with (either requires as input, or will be outputting predictions for note that these options are not mutually exclusive).

 © 2013 MARi, LLC

Starting a Project (Continued)

The selected list of PAs will be used to immediately generate a set of anonymized sample data from the live data currently found in MARi. This sample data will show the values of all selected PAs (where available) for a collection of users (ideally 1000+). The researcher can freely browse this sample data and approve it (or opt to change the PA list).

A walled-garden project space is then created for the new project, wherein the sample data is stored, along with an automatically generated pair of files: a blank config file and a main script file (containing a templated main function with no logical operations). The researcher is then launched into the Project Editor, wherein they can begin to edit scripts, acquire ontology, upload secondary data files and test their model.

Project Editor

Researchers will be spending the majority of their model development time in the Project Editor, an online IDE for defining and testing predictive models. The editor supports editing of much of the components that make up the model, and provides an easy way to inspect the sample data and to run a variety of tests. A rough concept of the Project Editor is shown in Figure 1.

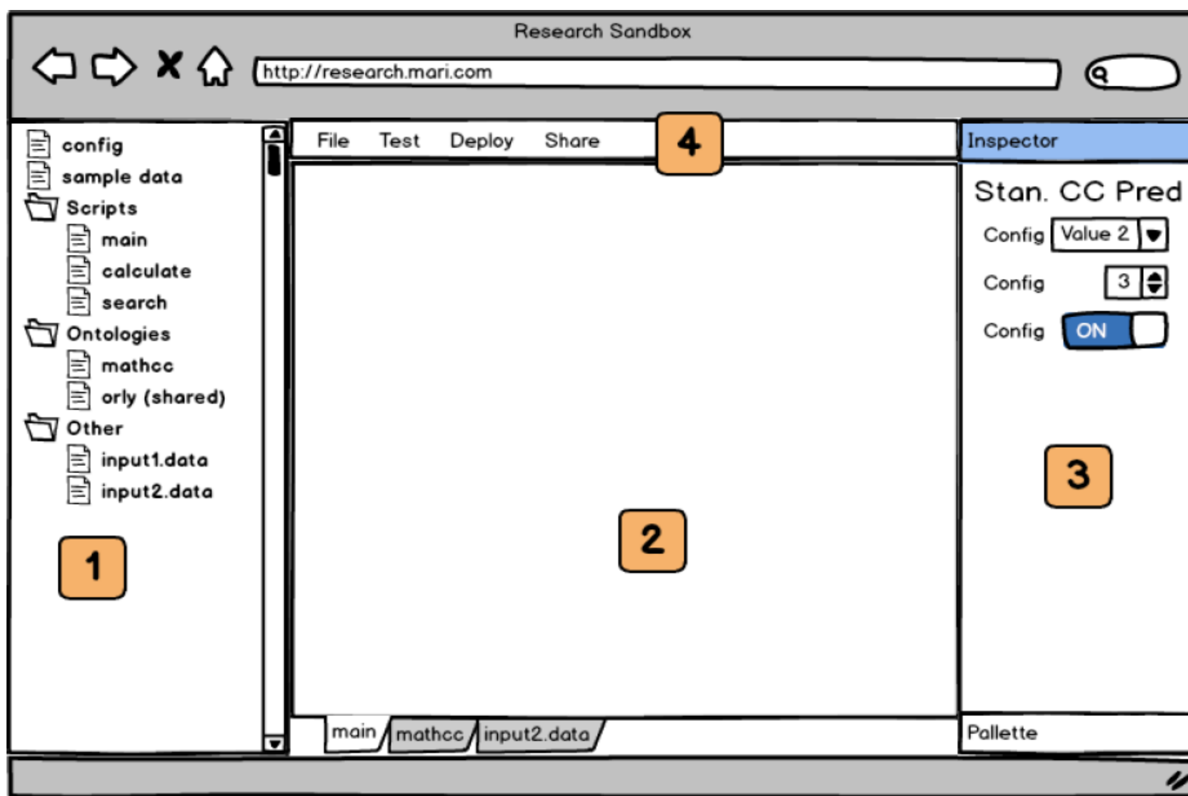

Figure 1: Project Editor Concept

© 2013 MARi, LLC

Project Editor (Continued)

The Project Editor window is divided into four parts:

1. The Project Navigator, anchored to the left of the screen provides a familiar top-down look at each asset in the project
2. The Asset Editor, anchored in the center, provides a context-sensitive editor view for the selected asset (and a convenient tabbed dock for fast switching between common assets)
3. The Detail Inspector, anchored to the right, provides a quick way to inspect and edit details of a selected component (asset, function, object, etc.) as well as a palette for new components
4. A menu runs along the top of the Asset Editor, providing saving and version control, testing options, release options and sharing options

Project Navigator

The Project Navigator is a top-down view of the project's walled garden. Assets can be added to the project and will be automatically categorized by their type: scripts (executables, including the required "main"), ontologies (created, imported or shared) and models (public or shared) will be sorted to their respective folders. Researchers can also directly upload assets to the sandbox (effectively a private file system that the researcher can manage as they see fit). These are filed under "Other" in the folder tree.

Config and Sample Data are both special cases and are treated as such by living in the root category. No other assets can be organized under root, and both Config and Sample Data (as well as Scripts/main) cannot be removed, relocated or renamed.

When selecting any asset in the navigator, a context-sensitive editor for that asset is opened in the Asset Editor panel.

 © 2013 MARi, LLC

Asset Editor

The Asset Editor displays the currently selected asset (from the Project Navigator) in a context-sensitive editor. For Release 1, the Asset Editor should cover the following unique asset types:

1. Config
2. Sample Data
3. Scripts
4. Ontologies

There should be no editor for any asset categorized as "Other" (assets uploaded by the user from an external source). However, there may be simple viewers for common asset types, such as text files, excel files, and XML files.

Editing Config

The config file for a project defines a collection of labeled triples: a data type, a range and a default value. While the default values are used when the model is instantiated without a custom config, the existence of the config file exposes to others the defined parameters and what they can be set to. (See the Detail Inspector for an example of overriding an external model's config file).

Due to the nature of the config file, the editor should be presented as an editable table:

Parameter ▲	Type ⇕	Range	Default ▼
weights.hasPrerequisite	float	0-1	0.8
weights.isStrongIndicatorOf	float	0-1	0.7
confidences.hasPreqrequisite	float	0-1	0.9
confidences.isStrongIndicatorOf	float	0-1	0.65

Figure 2: Config File Editor

 © 2013 MARi, LLC

Viewing Sample Data

NOTE: For now, we are assuming that sample data cannot be edited. This is debatable and may need to be changed in the future.

Sample data is automatically copied from the real-time data repository when the project is setup and PAs are selected (see Starting a Project). Sample data is taken from anonymized users for all PAs of interest to the project, including output PAs, and can then be presented for browsing in a table:

User ▲	abcd-efgh-ijkl-mnop Adding 2 digit... ▲	aaaa-bbbb-cccc-dddd Dividing single digit...	eeee-ffff-gggg-hhhh Multiplying two-digit...
anon001	0.85 / 0.90	0.65	0
anon002	0.80 / 0.95	0.65	00
anon003	0.75 / 0.92	0.60 / 0.90	0.60 / 0.90
anon004	0.75 / 0.85	0.50 / 0.80	0.60 / 0.90
anon005	0.75 / 0.50	0.44 / 0.75	0.44 / 0.75

aaaa-bbbb-cccc-dddd
Dividing single digit
numbers fluently

Figure 3: Sample Data Viewer - each PA displays both a score and a confidence

Beyond Release 1, the Sample Data Viewer should be extended to include a variety of convenient data investigation tools. These tools should include simple querying / filtering, and a collection of statistical functions to be run over the data. Tools to support ten-fold sampling techniques would also be a useful enhancement in subsequent iterations.

Editing Scripts

Scripts are editable in two ways: a traditional text-based IDE, and a graphical programming interface. For Release 1 we will focus only on the traditional text-based IDE.

Scripts are written in JavaScript. Each script file is effectively a JavaScript file, and can contain any number of declared variables, functions, etc. There must be a single script named “main” with a single function named “main” defined as:

- Exactly two inputs:
  - session – A session ID as passed in from the controller (this session ID allows the model to make anonymized requests of the data)
  - request – A request object from the controller that specifies, at least, a list of PAs for the model to produce predictions for

 © 2013 MARi, LLC

Editing Scripts (Continued)

- Exactly one output:
    - response – A response object that specifies, at least, the predicted scores and confidences for the requested PAs

Editing is done in the text-based IDE in a traditional manner. The text editor should be enabled with intellisense and auto-complete where possible, and should attempt to highlight syntax errors and type mismatches where possible. Type matching may not be completely possible since JavaScript is a late-binding language that allows variables to change their type at runtime, but it is worth investigating to see if some form of compile-time type checking can be offered.

Although the requested output PAs are passed as explicit parameters to the "main" function in order to identify which PAs are to be predicted, the underlying JavaScript code may access any of the input PAs simply by referencing them in the code. Input PAs are treated as global entities that are available at all levels of the predictive model's code structure.

Editing Ontologies

Ontologies are editable in two ways: a traditional top-down tree editor, and a drag-and-drop graphical editor. The graphical editor serves as a way to get a high-level grasp of a large ontology, or to quickly and easily edit a small ontology. The tree editor serves as a middle ground for doing fine-grained work on medium to large scale ontologies. Switching between the editors should be easily accessible.

Editing with the tree editor involves navigating to the desired PA in the tree oriented on the left. This will display the individual editor on the right, which includes the PA's label in the model (custom to the user), the PA's unique ID, and a form to select values for relations defined by the model.

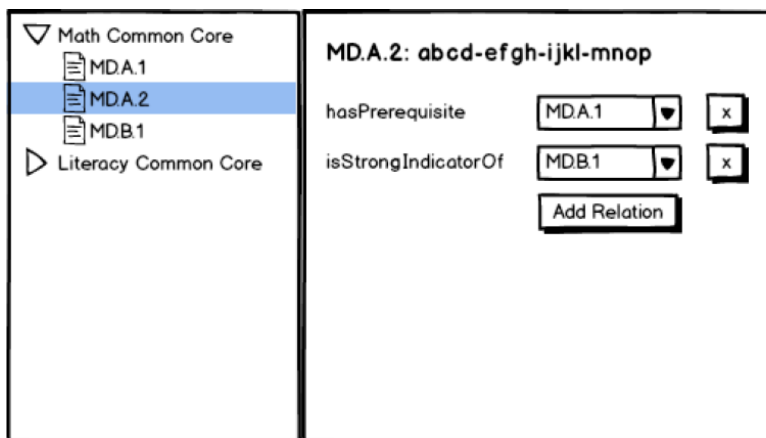

Figure 4: Ontology Tree Editor

© 2013 MARi, LLC

Editing Ontologies (Continued)

Editing with the graphical interface involves navigating to the desired PA by exploring the node cloud dragging the mouse between two PAs will form a relation that can be labeled immediately. Similarly, relations can be selected and deleted. Double clicking on a node will pop up the form editor used in the tree view.

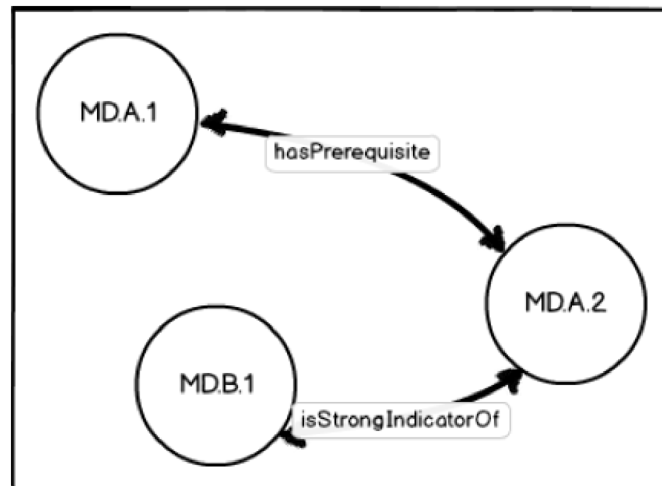

Figure 5: Ontology Graphical Editor

Detail Inspector/Palette

Located on the right side of the screen, the Detail Inspector and Palette are context-sensitive to the selected asset or component. Primarily used with the graphical script editor, the Detail Inspector allows the user to set the input parameters of a function or the config overrides of an externally imported model without having to dig into the code to do so.

For every parameterized value, an appropriate widget is displayed that allows the user to specify the input value (or in the case of external model configs, to override the default value). Under the hood, these values are represented in code and could be edited directly if desired those edits would then be reflected in the inspector.

The Palette is used to add new assets or components to the project- the contents displayed are context-sensitive: when working with the script editor, new function declarations or shared models can be dragged in. When working with the ontology editor, new (non-PA) concepts can be added, and when working with the Asset Editor, new assets can be created, imported or uploaded.

 © 2013 MARi, LLC

Main Menu

Spanning the top of the Project Editor is a menu whose contents contain:

- File
    - Save – saves the current asset being edited (unavailable for non-editable assets)
    - Revert – reverts the current asset to the last saved state (unavailable for non-editable assets)
    - New – adds a new asset to the project (a popup allows the user to select the asset type, and import / upload if desired or required)
- Test
    - Single Sandbox Sample – a single random sample is taken from the sample data (one row) and is processed through the existing model; the results are reported (or any runtime errors) and if the sample happens to have observed values for the PAs that are being predicted (e.g., it can work as an oracle) then a comparison between the predicted and "correct" values is also presented
    - Full Sandbox Sample – the entire sample data set is processed through the existing model; the results are reported (or any runtime errors); for each sample that has observed values for the PAs that are being predicted (e.g., it can work as an oracle) then a comparison between the predicted and "correct" values is collated and presented
    - Arbitrary Live – Sample a single random sample is taken from live data (non-sample data) and is processed through the existing model; the results are reported (or any runtime errors) and if the sample happens to have observed values for the PAs that are being predicted (e.g., it can work as an oracle) then a comparison between the predicted and "correct" values is also presented
- Deploy
    - New Major Version – the project is compiled into a fully enabled model, and is given the next available major version number; this does not affect privacy settings
    - New Minor Version – the project is compiled into a fully enabled model, and is given the next available minor version number; this does not affect privacy settings
    - Rollback Major – the currently deployed version is rolled back to the last major version
    - Rollback Minor – the currently deployed version is rolled back to the last minor version

© 2013 MARi, LLC

Main Menu (Continued)

- Share
    - Edit sharing for <filename> – a sharing dialog is presented for the currently selected asset
    - Edit sharing for Project – a sharing dialog is presented for the Project (this allows sharing editing for the Project as well as the compiled model)

The Share dialogues are intended to look a lot like Google Drive or Google Docs, with lists of people who can access each component.

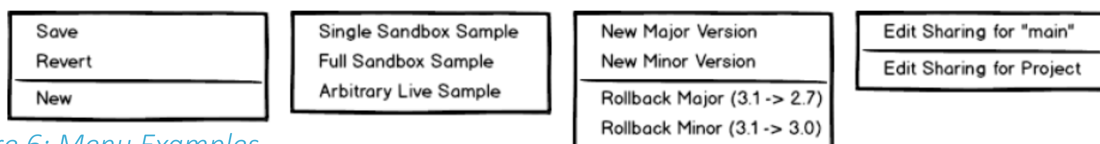

Figure 6: Menu Examples

Acquiring Ontologies

While the ontology editor is great for making changes to an existing ontology or building one completely from scratch, ontologies as resources can be partially (or completely) imported to a project from several external sources. When creating a new ontology asset in a project, the researcher will be prompted with several options:

1. Blank Ontology – a new blank ontology will be made, with, optionally, a node created for each PA declared as part of the project
2. Import Ontology – an ontology that is either publicly available through MARi, or has been shared with the user from another user's project space can be selected and added to the project (note that these ontologies are not editable unless a privately shared ontology has been flagged as editable by the owner)
3. Adopt Ontology – a new blank ontology will be made with a node created for each PA declared as part of the project

    a. Each publicly available or shared ontology that defines relations between the same PAs is presented to the user

    b. Selecting an external ontology displays a list of all of the relations defined between common PAs

    c. Users can opt to adopt relations one at a time into their ontology (effectively copying the semantics into their own, private, ontology)

4. Upload Ontology – an ontology (in a specified format) can be uploaded from the user's local machine

Existing ontologies can return the Adopt Ontology screen as often as desired, to continue to quickly pull in relations defined by external ontologies, even after local editing has occurred.

 © 2013 MARi, LLC

MARi Script SDK

When writing script code in a project, the user can benefit from a custom SDK that allows the scripts to access a variety of helpful (or necessary) functions. These functions range from accessing project files to advanced graph crawling algorithms for use in ontological reasoning. Below are a list of the functions that should be (at the minimum) included in Release 1.

Project SDK

project.file(category/filename): Gives the script a handle on the file in the absolute path of the project (the project is a walled garden, so "root" is the top-level of the project); this should primarily be used for accessing secondary data resource files

project.config(key): Returns the triple associated with the input config for the supplied key; note that the input config may be default, partially overridden or completely overridden this is handled externally by MARI (if the model is being tested then default is used, if the model is being run with no specified config, then

MARi SDK

mari.pa(session, id): Gives the script a handle on the PA object with the matching ID and session scoped (meaning an anonymized user, and any other session-level filtering)

pa.value(): Returns whatever single value is considered default for the PA

pa.average(): Returns an average of all values for the PA

pa.weightedAverage(): Return an average of all values for the PA, with a decay function applied to lessen the impact of old values

pa.lastValues(count): Returns a list of the last count values recorded for the PA

MARi SDK

model(name, config*): Gives the script a handle on the model found in the project (shared or public) matching the name; optionally instantiated with an overriding config

model.predict(session, id[]): Runs the main function of the model with the supplied session and requested list of PA ids, returning predicted scores (and confidences) for each PA

© 2013 MARi, LLC

Ontology SDK

ontology(filename): Gives the script a handle on an ontology object (found in the ontology category of the project by name these may be shared or public ontologies as well)

ontology.mariID(id): Gives the script a handle on the node in the ontology with the supplied mariID

ontology.node(name): Gives the script a handle on the node in the ontology with the supplied name

ontology.setWeight(relation, weight): Specifies the weight a particular relation has; this should be used in all path finding algorithms

ontology.shortestPath(node1, node2): Returns a path object representing the shortest path between the supplied nodes

ontology.shortestPath(node1, nodes2[]): Returns a path object representing the shortest path between the first input node and a single second input node (whichever is shortest)

ontology.shortestPath(nodes1[], nodes2[]): Returns a path object representing the shortest path between a single first input node and a single second input node (whichever is shortest)

node.parent(): Gives the script a handle on the parent node of the node

node.children(): Gives the script a handle on all child nodes of the node

node.ancestors(): Gives the script a handle on all ancestor nodes of the node

node.descendants(): Gives the script a handle on all descendant nodes of the node

node.depth(): Returns an integer representing the depth of the node in the ontology

path.score(): Returns the double value score representing the weights of all edges in the path

 © 2013 MARi, LLC

IDE Metrics Instrumentation

The Research Sandbox IDE is intended to be a living, frequently updated application that changes to match the identified needs of the research community. As such, it is critical to keep track of as many usage metrics as possible. So wherever possible, the underlying software structure must provide detailed log of user actions and click streams that can be used to analyze which features are used most often, which features are underutilized, task sequences are most effective, etc.

To accomplish this goal, the first release of the IDE must contain a click log that tracks the time-stamped event of every mouse click on every interactive element in the IDE. That log file can then be analyzed with different software to answer questions related to how researchers actually use the software.

Finally, there must be a method of allowing researchers to easily provide feedback to the IDE development team whenever they wish (i.e., the method should be available at all times, regardless of what state the IDE is in at the time.

© 2013 MARi, LLC