Moving JuliaStats Forward

John Myles White

Base changes are having big effects on JuliaStats

Julia 0.4 introduces Nullable{T}

Future versions might offer more Nullable tools

Nullable Types and Null Values

- Nullable{T} is a nullable type
- You can apply the isnull() predicate to any value of type Nullable{T}
- But only a subset of values of type Nullable{T} satisfy the isnull() predicate
- We call those values null values

Possible Nullable Features

- Lifting:
  - Given $f(x_1::T_1, ..., x_2::T_n)$ -> O, automatically define $f(x_1::Nullable\{T_1\}, ..., x_n::T_n)$ -> Nullable{O}
- Syntactic sugar to make Nullable more concise:
  - T? ≅ Nullable{T} (cf. C#, Swift, Hack, Dart, ...)
  - 1? ≅ Nullable(1)

Coherent Nullable Semantics

- Ontological: A type T is a set and a null-value for Nullable{T} indicates the non-existence of a value from that set
- Epistemological: A type T is a set and a null-value for Nullable{T} indicates the existence of a value from that set, but the value is unknown
- More possibilities exist (and matter). For example, censoring:
    - Given sets T and T', a null-value indicates that a value from T' exists, but is unknown
    - Non-null values are always from set T

The Natural Semantics

- Given $f(x_1::T_1, ..., x_n::T_n)$ -> O, the natural lifting is $f(x_1::Nullable\{T_1\}, ..., x_n::Nullable\{T_n\})$ -> Nullable{O}
- The result is always null-valued when any of the inputs are null-valued
- The input arguments can never be a mixture of nullable and non-nullable types
- Semantics sustain an ontological interpretation

Preserving Invariants

- But the epistemological interpretation suggests that you might want to preserve invariants that hold for all inputs of type T when lifting functions

- For example, Nullable{Bool}() & Nullable{Bool}(false) might always evaluate to false because this operation is false for any value the left-hand side might take on

- This is called three-valued logic

- This kind of invariant preservation doesn't work for most problems

Failure to Propagate Invariants

- If x is of type Int, then exp(x) >= 0 no matter the value of x

- But exp(Nullable{Int}) surely won't propagate this invariant since that requires that exp(Nullable{Int}) return Nullable{NonNegativeInt}, breaking the natural model of lifting

- In general, three-valued logic is the rare case when this propagation of invariants is somewhat defensible

Lifting: Opt-In or Opt-Out?

- Given a chosen semantics for lifting, how do we apply lifting?
  - Never
  - Opt-In at Function Definition Site
    - @inline approach
  - Opt-In at Lifted Function Definite Site
    - @vectorize_1arg approach
  - Opt-In at Call Site
    - @lift
  - Automatic for Functions Matching Some Criteria
    - C# model -- +,-, / are lifted, but foo(x, y) is not lifted

Complex Lifting

- How to lift functions that depend upon multiple array arguments (e.g Array{Nullable} or NullableArray)?
- Much larger space of sensible semantics

What's Happening in JuliaStats Packages

The Data Pipeline

- Pulling Data In
- Preprocessing Data
- Exploring Data
- Create Data Structures for Modeling
- Model Data
- Summarize Results

Pulling Data In

- Where we pull it from:
  - CSV files
  - Fixed width files
  - Excel files
  - DB's
- What we represent it as:
  - Tuple
  - Dict{Vector}
  - DataFrame

Interacting with DB's

- Design database independent interfaces?
- What data structures do we use to pull and push data from DB's?
- What API do we use to communicate with the DB?
    - Hand-written SQL?
    - A DSL that mimics the semantics of SQL?
    - An API that can be translated into SQL, but not always losslessly?

Default Data Structures?

- "It is better to have 100 functions operate on one data structure than to have 10 functions operate on 10 data structures." - Alan J. Perlis

- Alan Perlis clearly didn't do a lot of modern sparse linear algebra

Rows vs Columns

- What is the fundamental unit of data representation?
- Long-standing debate in DB world
  - Relational model defined on rows
  - DataFrame model defined on columns

Possible Julia Representations

- Rows are tuples
- Rows are Dict{Any}
- Tables are Dict{Vector}

Best Representation?

- Amenable to type inference?
- Require users to call functions on columns, not DataFrames?
- Redundant information in memory?

Current Data Representation

- Assume we're using Dict{Vector} from now on
- This is basically what a DataFrame is

Preprocessing Data

- SQL embodies most of the core preprocessing activities one needs:
  - Subset selection
  - Grouping and aggregation functions
  - Joining
- Good high-level language implementations:
  - SQLAlchemy
  - dplyr

Preprocessing Data

- We have basic support in DataFrames, but we're not close to the performance of SQLite
- Biggest weakness is indexing
- Should we represent all data using SQLite in-memory tables?

The Array-Table Divide

- A table cannot guarantee ordering
- Implies there are no implicit primary keys
- Any operating dependent on ordering should happen outside of the table world, e.g.:
  - Autocorrelation within a column
  - Sum of columns from separate tables

Making Tables Less like Matrices

- We need a pure separation of tables and arrays
- Cannot depend upon array-like properties:
  - Sorting
  - Row indexing
  - Row deletion has non-local effects
- When possible, avoid materializing results in memory
- Caveat: Must carefully ensure alignment of columns

Canonical Translations Tables into Array-like Objects

- Obvious form is the existing Dict{Vector} form
- But ModelMatrix is often more interesting

The Formula DSL

- ModelMatrix(y ~ x, df) -> M::Matrix{Float64}

The Formula DSL

- y ~ x
  - Construct a vector y of outcomes to predict
  - Construct a matrix containing columns that represent x
  - If x is numeric, make a Vector{Float64}
  - If x is categorical, use dummy variable encoding

The Formula DSL

- y ~ x + y
    - Similar logic, but only add enough columns to provide the full span of those variables without creating linear dependence

The Formula DSL

- $y \sim x + x{:}y$
  - Construct an interaction column, which is the element-wise multiplication of appropriate columns
  - For Boolean variables (e.g. dummy variables), this is easily an AND encoding.

The Formula DSL

- What should the type of M be?

Summarize Results

- With some more work on visualization and package pre-compilation, we should have powerful tools for reporting