stats_in_julia_slides

April 6, 2017

Introduction to Julia for Statistics
Paul Stey, PhD
Brown Center for Biomedical Informatics
April 5, 2017

```
In [1]: using DataFrames
        using PlotlyJS
        using HypothesisTests
        using GLM
        using DecisionTree

        include("../utils/plot_utils.jl")
```

```
Out[1]: scatterplot (generic function with 2 methods)
```

1 Vectors, Matrices, and *N*-Dimensional Arrays

- All native types in Julia (part of `Base` Julia)
- Behave very similarly to counterparts in R, Python, and Matlab

1.1 Vectors

- Vectors in Julia are 1-dimensional arrays
- Arrays are containers
- Arrays (and therefore vectors) can store objects of any type
- For example: `Int`, `Float64`, `String`, `Dict`, `Function`
- This includes other arrays (i.e., array of arrays)

```
In [ ]: v = [4, 5, 6]
```

```
In [ ]: u = [1.2, 4.5, 6.5]
```

```
In [ ]: w = ["dog", "cat", "bird"]
```

1.1.1 Initializing and Growing Vectors

```
In [ ]: a = Int[]                          # initialize empty Int vector
In [ ]: a2 = Array{Int, 1}()               # identical to the above
In [ ]: a3 = Vector{Int}()                 # also identical to above
In [ ]: push!(a, 12)                       # inserts 12 in to a
In [ ]: push!(a, 1000)
In [ ]: append!(a, [9, 18, 27])
```

1.1.2 Types Matter

- Julia is a dynamic language like Python and R, but it has a nicer type system
- For example, the compiler infers the type of your variables, and optimizes for these

```
In [ ]: a = [3, 4, 5]                      # vector of Int64s
In [ ]: push!(a, 3.14)                     # Fail: trying to push Float into vector of
In [ ]: s = String[]                       # initialize empty vector of strings
In [ ]: push!(s, "this")
        push!(s, "is")
        push!(s, "a string vector")
In [ ]: push!(s, e)
```

1.1.3 Pre-Allocating Vector

- Almost always a good idea to pre-allocate vectors (if possible)
- Has performance advantages
- Growing a vector is 2 operations vs 1
- Forces you to think more about what your code is accomplishing
- Self-documenting final size of your vector

```
In [ ]: x1 = zeros(Int, 10)                 # create vector of ten 0s of type Int64
In [ ]: x2 = falses(5)                     # create vector of 5 Bool set to false
```

1.1.4 Indexing a Vector

- Nearly identical to indexing in R, Python, and Matlab
- Julia used 1-based indexing (like R and Matlab)

```
In [ ]: s = ["this", "is", "a string vector"]
In [ ]: s[2]                               # get second element
In [ ]: s[3]                               # third element
In [ ]: s[1] = "that"                      # assign to first element
In [ ]: println(s)
```

1.1.5 Slicing and Subsetting a Vector

```
In [ ]: s[1:2]                          # get subset of vector (1st and 2nd element
```

```
In [ ]: s[2:end]                        # subset from 2nd element to last element
```

```
In [ ]: animals = ["dog", "cat", "fish", "mouse", "potato"]
```

```
In [ ]: mammals_indcs = [1, 2, 4]       # vector for indexing
```

```
In [ ]: animals[mammals_indcs]          # identical to `animals[[1, 2, 4]]`
```

1.1.6 Element-Wise Operations

Element-wise operations on vectors in Julia are performed using the . operator, often called the broadcasting operator.

```
In [ ]: b = [1, 2, 3, 2]
```

```
In [ ]: b .== 2
```

```
In [ ]: b .< 3
```

1.1.7 Concatenating Vectors

```
In [ ]: x1 = [1, 2, 3]

        x2 = [4, 5, 6]

        x3 = [x1; x2]                   # concatenating with [] and ;
```

```
In [ ]: x4 = vcat(x1, x2)               # concatenate with vcat() is same as above
```

1.2 Matrices and *N*-Dimensional Arrays

```
In [ ]: A = Array{Float64}(5, 3)        # initialize 5-by-3 matrix (start as 0.0)
```

```
In [ ]: Z = zeros(5, 3)                 # same as above
```

```
In [ ]: B = [1 2 3;
             4 5 6;
             7 8 9]                     # semi-colon used to indicate end of row
```

1.2.1 Indexing and Slicing Matrix (or *N*-dimensional array)

```
In [ ]: B[3, 1]                         # gets entry in third row first column
```

```
In [ ]: B[2, :]                         # gets all of second row
```

```
In [ ]: B[:, 3]                         # gets all of third column
```

1.3 Reading Data from Flat File

- Julia has a few options for reading data from "flat files"
- There are some options in `Base` Julia
- The read-in data will generally be represented in 2-dimensional array

1.3.1 Using `readdlm()`

- Function in `Base` Julia
- Used for reading delimited plain-text data files
- CSV files, tab-delimited, etc.

```
In [ ]: d = readdlm("somedata.csv", ',')                # specify comma-delimited

In [ ]: d1 = readdlm("otherdata.tsv", '\t')              # specify tab is the delim
```

1.3.2 Using `readcsv()` and its Optional Arguments

- The `readcsv()` function is syntactic sugar for `readdlm()` with ',' argument
- Has many options for how data are read in
- Returns 2-dimensional array, or tuple with data array and header array

```
In [ ]: d2 = readcsv("./data/somedata.csv")                # equivalent to readdlm

In [ ]: d3 = readcsv("./data/somedata.csv", header = true) # treat first line as co

In [ ]: typeof(d3)

In [ ]: d3[1]                                             # 1st element in Tuple is

In [ ]: d3[2]                                             # 2nd element in Tuple is

In [ ]: d4 = readcsv("./data/somedata.csv", header = true, skipstart = 3)   # skip

In [ ]: d4[1]
```

Note: There are a variety of other optional arguments you can pass to the `readdlm()` family of functions. You can see more of these by using `?readdlm` at the Julia prompt, or (even better) you can go read the documentation online.

2 DataFrames

- *de facto* object for data analysis
- `NA` type for missing data
- In the tradion of R `data.frame` objects
- Similar to pandas `DataFrames` in Python
- Two-dimensional tabular array capable of handling mixed-type data
- Also capable of handling categorical (i.e., `factor`) variables

2.1 Using `DataFrame` Objects

- Interaction with `DataFrame` objects is similar to interaction with matrices
- Many additional features for reshaping, summarizing, and joining data
- Many packages play nicely with `DataFrames`, for example:
- DataArrays.jl
- Query.jl
- GLM.jl

2.1.1 Simple Read-in to `DataFrame`

```
In [ ]: using DataFrames                          # assumes DataFrames is installed

In [ ]: df1 = readtable("./data/somedata.csv")
```

2.1.2 Indexing and Subsetting

```
In [ ]: df1[2, 3]                                 # get second row, third column

In [ ]: df1[4, :]                                 # get all of 4th row

In [ ]: df1[3, :condition]                        # indexing with column name

In [ ]: df1[:condition]                           # get single column
```

2.1.3 Slightly more Interesting Subsetting

```
In [ ]: col_subset = [:gender, :condition]        # specify columns to extract

        df1[:, col_subset]                        # get subset of columns

In [ ]: df1[:, [:gender, :condition]]             # equivalent to above

In [ ]: df1[:, [3, 4]]                            # equivalent to above
```

2.1.4 Subsets of Rows

```
In [ ]: is_female = df1[:gender] .== "f"          # get boolean vector indicating fem

In [ ]: df1[is_female, :]                         # use boolean vector for subsetting

In [ ]: df1[df1[:gender] .== "f", :]              # equivalent to above
```

2.2 Options for Reading to `DataFrame`

- Handling "factor" variables
- Treatment of `NA` variables

2.2.1 Optional Arguments for `readtable()`

```
In [ ]: df2 = readtable("./data/somedata.csv", makefactors = true)        # treat st

In [ ]: df3 = readtable("./data/otherdata.tsv", makefactors = true)

In [ ]: df4 = readtable("./data/otherdata.tsv", nastrings = ["999"])

In [ ]: df5 = readtable("./data/somedata.csv", nastrings = ["", "NA", "999"])
```

Important Caveat

There is currently a debate among core developers in the Julia community regarding whether to use DataFrames or DataTables. The latter package very recently split off from DataFrames, and is very similar superficially. But `DataTable` objects use a different underlying array representation to handle `NA` values more efficiently. Because of this, they have better performance than `DataFrame` objects (in some cases, *much* better performance). However, for many applications, the performance penalty of DataFrames is inconsequential. Furthermore, the DataTables package has a few rough spots that make its interface slightly less elegant. Nonetheless, keep your eye on these two packages.

2.3 Challenge Question 1

Using the data in chronic_kidney_disease.csv, determine whether or not the oldest patient in the sample has chronic kidney disease. Note that the `class` variable indicates a patient's CKD status.

As a hint, you will probably want to use the `maximum()` function and the `find()` function.

And you'll need to consider how to handle `NA` values; there is a function `dropna()` that will be useful.

```
In [ ]: # Data can be read into Julia using the following

        ckd = readtable("../data/chronic_kidney_disease.csv")
```

3 Descriptive Statistics

Useful Packages: - StatsBase

```
In [ ]: x = randn(100)                # 100 draws from standard Gaussian

In [ ]: mean(x)
        median(x)
        mode(x)
        std(x)
        var(x)
        minimum(x)
        maximum(x);
```

3.0.1 Descriptives with `DataFrame`

```
In [ ]: x1 = describe(ckd)          # get many useful descriptive stats

In [18]: @show mean(ckd[:blood_pressure])          # returns NA (not what we want)

         @show mean(dropna(ckd[:blood_pressure]))  # dropna() removes NA values

mean(ckd[:blood_pressure]) = NA
mean(dropna(ckd[:blood_pressure])) = 76.46907216494846


Out[18]: 76.46907216494846
```

4 Inferential Statistics

Useful Packages:

- HypothesisTests (Binomial test, *t*-tests, χ^2-test, and many more...)
- GLM (linear and generalized linear models)
- MixedModels (Multi-level (or Mixed-effects) models)

4.1 Binomial Test

The binomial test is a statistical test of dichotomous data's (e.g., "success" and "failure") deviation from expected distribution. Binomial tests can be used to answer questions of this form: *If the true probability of success is P then what is the probability of the data we have observed?*

```
In [ ]: # Coin Tossing Example:
        # Simulate data from Binomial, test
        # hypothesis data came from fair coin

        using HypothesisTests
        using Distributions

        binom = Binomial(1, 0.6)            # initialize Binomial dist'n

        srand(137)                          # set seed for reproducibility
        x = rand(binom, 30)                 # 10 random draws from our dist'n

        xbool = convert(Array{Bool,1}, x)   # cast x to vector of Booleans
        BinomialTest(xbool, 0.5)            # test null hypothesis that p = 0.5
```

4.2 Student's *T*-test

Extremely common statistical test for differences in means between two groups on some continous variable. *T*-tests are often used to investigate the effects of some new treatment versus a control group.

```
In [ ]: # Life Expectancy Example:
        # Simulate data from Gaussian, test whether smokers
        # and non-smokers have same life expectancy

        non_smokers_gaussian = Normal(75, 7)
        smokers_gaussian = Normal(65, 7)

        srand(1137)

        n = 100
        non_smokers = rand(non_smokers_gaussian, n)      # n random draws from Gaus
        smokers = rand(smokers_gaussian, n)

        EqualVarianceTTest(smokers, non_smokers)         # two-sample t-test (assu
```

```
In [ ]: hist2(smokers, non_smokers)
```

4.3 Pearson's Correlation

Pearson's correlation is a measure of the linear relationship between two continous variables. The resulting correlation coefficient, *r*, ranges between 1 and -1. Positive values of *r* indicate a positive association between the variables (e.g., hours of studying and GPA), while negative values indicate a negative relation between variables (e.g., beers-per-week and GPA).

```
In [ ]: # correlation between two variables in vectors
        n = 1000
        x = randn(n)
        y = 1.5x .+ randn(n)

        scatterplot(x, y)
```

```
In [ ]: cor(x, y)
```

```
In [ ]: # pairwise correlation of all variables in a matrix
        A = randn(100, 5)
        cor(A)
```

4.4 Challenge Question 2

Using the `chronic_kidney_disease.csv` dataset from above, use an appropriate statistical test to determine if patients with chronic kidney disease (CKD) have significantly higher blood urea than patients without CKD.

As a hint, you will likely want to use the HypothesisTests package.

4.5 Linear and Generalized Linear Models

- LMs and GLMs are one of the most powerful and fundamental classes of models
- Tremendously useful for making inferences as well as for making predictions
- Ubiquitous across scientific disciplines
- Serve as foundation for some of the most promising advances in machine learning and artificial intelligence

4.5.1 Linear Regression Models

Useful Packages: - GLM - MixedModels (for multi-level models) - Mamba (for Bayesians)

```
In [ ]: using GLM

        ckd = readtable("../data/chronic_kidney_disease.csv")

        fm1 = lm(@formula(blood_pressure ~ 1 + age), ckd)     # fit linear model reg
```

```
In [ ]: coef(fm1)                              # get model coefficients (i.e., Betas)

        stderr(fm1)                           # standard error of coefficients

        confint(fm1)                          # confidence intervals for coefficients

        predict(fm1)                          # predicted value for each observation (i

        residuals(fm1)                        # residuals (i.e., y - y_hat)
```

```
In [ ]: # fit indices
        deviance(fm1)

        aic(fm1)

        bic(fm1)
```

```
In [ ]: # make predictions with fitted model
        new_data = DataFrame(age = [10, 20, 30, 40, 50])

        predict(fm1, new_data)
```

```
In [ ]: # Adding predictors
        fm2 = lm(@formula(blood_pressure ~ 1 + age + hemoglobin), ckd)
```

```
In [ ]: # Adding interaction terms
        fm4 = lm(@formula(blood_pressure ~ 1 + age + serum_creatinine + age*serum_c
```

Result above shows that `age` and `serum_creatinine` are both positive predictors of blood pressure. Additionally, we see a significant [negative] interaction indicating that as `age` increases, `serum_creatinine` becomes an increasingly poor predictor of `blood_pressure`.

4.5.2 Binomial Logistic Regression

Binomial logistic regression models are used when fitting models to dichotomous outcome variables (e.g., 0 and 1).

```
In [ ]: using DataFrames
        using GLM
```

```
thoracic = readtable("../data/thoracic_surgery.csv", makefactors = true)

fm5 = glm(@formula(died ~ 1 + tumor_size), thoracic, Binomial())          #
```

```
In [ ]: coef(fm5)

stderr(fm5)

confint(fm5)

deviance(fm5)

aic(fm5)

bic(fm5)
```

4.5.3 Poisson Regression

Poisson regression is useful for modeling outcome variables that are in the form of count data.

```
In [ ]: prussian = dataset("pscl", "prussian")                # load Prussian horse

In [ ]: fm6 = glm(@formula(Y ~ 1 + Year + Corp), prussian, Poisson())   # defaults t

In [ ]: coef(fm6)
        stderr(fm6)
        confint(fm6)
        deviance(fm6)
        aic(fm6)
        bic(fm6)
```

5 Challenge Question 3

Using the chronic_kidney_disease.csv data, determine which of the following predictors (if any) are related chronic kidney disease (CKD):

`blood_urea`,
`hemoglobin`,
`red_blood_cell_count`,
`white_blood_cell_count`.

Hints: - There are a few days this could be done, let's use a single regression model of some kind - Our outcome variable is `class` in the CKD data, this needs to be re-coded as 0/1

6 Statistical (Machine) Learning

Useful Packages: - DecisionTree - RandomForest - Lasso - GLMNet (for ridge regression, lasso, and elastic net) - LARS (lasso and elastic net) - Clustering - GradientBoosting - XGBoost - Mocha (deep neural nets)

6.1 Bagging

Bagging is "bootstrap aggregation", and involves fitting a many individual classification or regression trees. Thus, bagging can be used with both categorical and continous data. The use of many trees improves the prediction accuracy of your fitted model over a single tree by decreasing the chance of overfitting your data (see bias/variance tradeoff).

```
In [27]: # take only complete cases
         using RDatasets
         stagec = dataset("rpart", "stagec")

         stagec_comp = stagec[completecases(stagec), :];
```

```
In [47]: # Clean up data and fit model

         is_tetraploid = stagec_comp[:Ploidy] .== "tetraploid"

         stagec_comp[:tetra] = is_tetraploid

         # must convert to Array
         y = convert(Array{Float64,1}, stagec_comp[:G2])
         X = convert(Array{Float64,2}, stagec_comp[[:Age, :Grade, :Gleason, :EET, :

         fm7 = build_forest(y, X, size(X, 2), 500)
```

```
Out[47]: Ensemble of Decision Trees
         Trees:      500
         Avg Leaves: 31.294
         Avg Depth:  9.986
```

```
In [25]: apply_forest(fm7, [55.0, 3.0, 2.0, 1.0, 1.0])
```

```
Out[25]: 17.408657000000012
```

6.2 Random Forest

Random forests were developed after bagging, and are a generalization of the idea of taking bootstrap samples from your dataset and fitting many trees. Random forests differ from bagged trees in that for each split point in the tree, the algorithm only considers some subset of the predictors as candidates on which to split. This has the effect of further reducing the correlation between trees beyond what is already achieved by the bootstrapping. This reduces overfitting and improves prediction accuracy for new data.

```
In [ ]: fm8 = build_forest(y, X, 3, 500)          # 3rd param controls m_try
```

7 Question 4

Using the `stagec` data from above, fit several random forest models predicting the `G2` score. Try experimenting with different numbers of trees and different numbers of variable subsets for candidate splitting (i.e., `m_try`, the third argument to the `build_forest()` function).

In order to evaluate the quality of the models on training data, write a function that calculates the mean-squared error of the fitted model. The function should take 3 arguments: (1) the fitted model, (2) the vector with the outcome variable and (3) the matrix of predictors.

What was the mean-squared error of your best-fitting model?

Data can be loaded with the code below.

8 Question 5

Using the `aldh2` dataset from the `gap` package in R, try fitting a few random forest (or bagged tree) models using Julia to predict whether a given patient is an alcoholic using their genetic information.

What is the prediction accuracy of your best model? What were the meta-paremeters of your best-fitting model?

The data can be loaded using the code below.

```
In [ ]: aldh2 = dataset("gap", "aldh2")
```

8.1 Ridge Regression, the Lasso, and Elastic Net

- Ridge regression is a form of regularized linear and generalized linear model that penalizes the L2 (Euclidean) norm of the regression parameter estimates
- The lasso (Least Angle Shrinkage and Selection Operator) penalizes the L1 norm of the vector of regression coefficients. This has the effect of shrinking the least important regression coefficients to zero.
- The Elastic Net combine L1 and L2 norm penalties

```
In [ ]: using Lasso
        swiss = dataset("datasets", "swiss")
```

```
In [ ]: Xswiss = convert(Array{Float64, 2}, swiss[:, 2:5])

        yswiss = convert(Array{Float64, 1}, swiss[:, 6])

        fm9 = fit(LassoPath, Xswiss, yswiss,  = 1)          #  = 0 is ridge, 1 is la
```

```
In [45]: fieldnames(fm9)
```

```
Out[45]: 10-element Array{Symbol,1}:
          :m
          :nulldev
          :nullb0
          :
          :auto
          :Xnorm
          :pct_dev
          :coefs
          :b0
          :niter
```

```
In [46]: @show fm9.
         full(fm9.coefs)

fm9. = [23.6306,21.5314,19.6186,17.8757,16.2877,14.8407,13.5223,12.321,11.2265,10.2

Out[46]: 4Œ62 Array{Float64,2}:
          0.0   0.0        0.0        0.0          1.66185    1.66396    1.66589
          0.0   0.0        0.0        0.0           0.617038   0.618098   0.619064
          0.0  -0.265982  -0.508336  -0.729159     -2.95888   -2.95994   -2.9609
          0.0   0.0        0.0        0.0           3.40215    3.40709    3.4116
```

9 Calling R from Julia

- R has been around since early 90s or late 70s (depending on how you count S language)
- Over 10,000 R packages registered on CRAN
- R is very specialized for statistical programming

9.1 RCall.jl Package

- Happily, we can easily call R from Julia
- Simply install RCall.jl package and then load the installed package

```
In [ ]: Pkg.add("RCall")                    # only needed first time using RCall

        using RCall
```

9.2 Two Environments: R and Julia

- RCall package starts an R session using your existing R interpretter
- Objects in your Julia session need to be passed to R session
- And vice versa

9.2.1 Putting a Julia object in R

```
In [ ]: a = [2, 3, 4, 5]

        @rput a

        R"print(a)"

        R"print(typeof(a))"
```

9.2.2 Getting an object from R to Julia

```
In [ ]: R"b <- rnorm(10)"

        @rget b
```

```
println(b)
```

```
In [ ]: R"B <- matrix(rnorm(25), nrow = 5)"
```

```
@rget B
```

```
display(B)
```

10 Question 6

Use R and the randomForest package via the RCall.jl package in Julia to fit a random forest model on the `chronic_kidney_disease.csv` data set. In particular, fit a model with 5000 trees to predict whether or not patients have chronic kidney disease.

After fitting the model, extract the variable importance estimates (mean Gini decrease) from the fitted model. Pass a dataframe back to Julia that has two columns (1) name of the predictor, and (2) mean Gini decrease for that predictor.

Sort the returned data frame such that more important predictors (i.e., larger values) are at the top.

The following steps should serve as a general to complete this:

1. Read the data in to Julia
2. Ensure RCall.jl is loaded
3. Pass the dataframe from Julia to R
4. Fit the model in R using `randomForest()` function with argument `ntrees = 5000`
5. Use the `importance()` function to extract the estimates of variable importance from the fitted model
6. Pass the estimates back to Julia

Some Hints:

- The `importance()` function in R returns a data frame whose row names are the variable names, and the last column is mean Gini decrease
- The documentation for the randomForest package in R can be found here: https://cran.r-project.org/web/packages/randomForest/randomForest.pdf
- The `sortperm()` function will be useful for sorting in the last step

11 Recommended Resources

11.0.1 Statistical Inference

- Casella and Berger (2002) *Statistical Inference*
- Wasserman (2004) *All of Statistics*

11.0.2 Linear Models

- Gelman and Hill (2007) *Data Analysis Using Regression and Multilevel/Hierarchical*
- Rencher and Schaalje (2008) *Linear Models in Statistics*

11.0.3 Generalized Linear Models

- Agresti (2002) *Categorical Data Analysis*
- Hosmer and Lemeshow (2000) *Applied Logistic Regression*

11.0.4 Machine Learning

- Hastie, Tibshirani, & Friedman (2001) *Elements of Statistical Learning*
- James, Witten, Hastie, & Tibshirani (2015) *An Introduction to Statistical Learning*
- Kuhn and Johnson (2013) *Applied Predictive Modeling*