lmutils

Tools for blazingly fast statistical analysis

Josef Graf

McMaster University

Table of contents

RARity Case Study

Context

- The original code was written in **148 lines** of pure R, and by no fault of the original author (purely that of R) was slow.
- A single phenotype and gene block took **42 minutes and 38 seconds** to run.
- For a full UKB run this would take days, if not weeks.

The Code

```
lmutils::set_num_worker_threads(32)
phenos <- lmutils::Mat$new("phenos.rkyv")
phenos$remove_column("eid")
genos <- lmutils::Mat$new("genos.rkyv")
genos$min_column_sum(2)
genos$na_to_column_mean()
genos$standardize_columns()
genos$remove_column("eid")
df <- lmutils::calculate_r2(genos, phenos)
```

- The new code is only **9 lines** long and makes use of a variety of functions from lmutils.
- A single phenotype and gene block can now be run in as little as **44 seconds**.

What!?! But how!

- Parallelism
  - Using 1 thread instead of 32 only adds about 4 and a half minutes.
- rkyv
  - Loading from `.RData` instead of `.rkyv` only adds about 10 seconds.
- So where does the rest of the time go?
- Rust
- faer
  - SIMD, parallelism, linear algebra magic.

Welcome to lmutils

Installation

First, you need to install the Rust toolchain. You can do this by running

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

Then, you can install lmutils by running the following in R. Once the package is in a more final state I'm going to submit it to CRAN.

```
install.packages(
"https://github.com/GMELab/lmutils.r/archive/refs/heads/master.tar.gz",
repos=NULL) # use .zip for Windows
# OR
devtools::install_github("GMELab/lmutils.r")
```

Introductory definitions

- **Matrix-convertible object:** a data frame, matrix, file name, numeric column vector, or a `Mat` object.
- **List of matrix-convertible objects:** a list of matrix-convertible objects, a character vector of file names, or a single matrix-convertible object.
- **Standard output file:** a character vector or list of file names matching the length of the inputs or `NULL` to return the output. If a single input (not in a list) was given, the output will not be in a list.
- **Core parallelism:** the number of primary operations that will be run in parallel (i.e. gene blocks processed at once, files read at once, etc.).

The `Mat` object

- The `Mat` object is a wrapper around lmutils's internal representation of a matrix.
- It allows for easy, efficient, and lazy manipulation of the matrix.
- It is the recommended way to interact with the newest version of lmutils.

```
mat <- lmutils::Mat$new("phenos.rkyv")
eids <- mat$col("eid")
r <- mat$r() # just like any other R matrix
colnames(r) == mat$colnames() # TRUE
r$eid == eids # TRUE
```

File Types

Imutils supports a variety of both compressed and uncompressed file formats.

- `.csv` (`.csv.gz`) comma-separated values, first row is the column header.
- `.tsv` (`.tsv.gz`) tab-separated values, first row is the column header.
- `.txt` (`.txt.gz`) space-separated values, first row is the column header.
- `.rkyv` (`.rkyv.gz`) a binary format that is both fast and space-efficient (**recommended**).
- `.RData` an R-specific binary format.

Reading and Writing Files

lmutils supports two methods of reading and writing files, using the `Mat` object or global functions.

```
mat <- lmutils::Mat$new("phenos.rkyv")
r <- mat$r()
mat$save("phenos.csv")

# both functions support a list of files
lmutils::load("phenos.rkyv")
lmutils::save(
    list("file1.csv", matrix(1:9, nrow=3), 1:3,
      data.frame(a=1:3, b=4:6), mat),
    c("file1.tsv", "file2.rkyv.gz", "file3.rkyv",
      "file4.rdata", "file5.tx.gz")),
)
```

Configuration

lmutils provides a number of global configuration options that can be set using environment variables or through various functions. Please note, these generally cannot be changed after calling any other lmutils functions.

- `LMUTILS_LOG` / `lmutils::set_log_level`: the log level, defaults to `info`. The valid levels (in order of increasing verbosity) are `off`, `error`, `warn`, `info`, `debug`, and `trace`.
- `LMUTILS_CORE_PARALLELISM` / `lmutils::set_core_parallelism`: the number of primary operations that will be run in parallel, defaults to 16.
- `LMUTILS_NUM_WORKER_THREADS` / `lmutils::set_num_worker_threads`: the number of worker threads, defaults to `num_cpus / 2`.
- `LMUTILS_ENABLE_PREDICTED` / `lmutils::disable_predicted` / `lmutils::enable_predicted`: whether to calculate and return predicted values in linear models, defaults to disabled for performance reasons.

Matrix Manipulation

lmutils provides more than 30 functions just for manipulating `Mat` objects, not to mention more than a dozen other global functions for matrices and data frames. Here are a few examples.

- `Mat$remove_column(col)`: remove a column by name.
- `Mat$min_column_sum(threshold)`: remove columns with a sum below a threshold.
- `Mat$na_to_column_mean()`: replace `NA` values with the column mean.
- `Mat$standardize_columns()`: standardize columns to have a mean of 0 and a standard deviation of 1.
- `Mat$transpose()`: transpose the matrix.
- `Mat$sort(col_idx)`: sort the matrix by a column.
- `Mat$dedup(col_idx)`: deduplicate the matrix by a column.

A full list of functions can be found in the documentation on GitHub at github.com/GMELab/lmutils.r.

Statistical Functions

lmutils also provides a number of lightning fast statistical functions.

- `lmutils::compute_r2(v1, v2)`: compute the R^2 between two vectors.
- `lmutils::mean(v)`: compute the mean of a vector.
- `lmutils::median(v)`: compute the median of a vector.
- `lmutils::sd(v)`: compute the standard deviation of a vector.
- `lmutils::var(v)`: compute the variance of a vector.

calculate_r2

The first kind of primary analysis is calculating the R^2 value between a list of matrix convertible objects (i.e. gene blocks) and a single matrix convertible object (i.e. phenotypes).

```
df <- lmutils::calculate_r2(list("genos1.rkyv", "genos2.rkyv",
"genos3.rkyv"), "phenos.rkyv")
```

The output is a data frame with the columns:

- `r2`: the R^2 value.
- `adj_r2`: the adjusted R^2 value.
- `data`: the name or index of the data.
- `outcome`: the name or index of the outcome.
- `n`: the number of samples, i.e. rows.
- `m`: the number of predictors, i.e. SNPs.
- `predicted`: the predicted values of the outcome for each sample, only returned if enabled.

column_p_values

The second kind of primary analysis is calculating the *p*-values of the each pair of data and outcome columns. In some testing it can do a 20 thousand column block in about 2-3 seconds.

```
df <- lmutils::column_p_values(list("genos1.rkyv",
"genos2.rkyv", "genos3.rkyv"), "phenos.rkyv")
```

The output is a data frame with the columns:

- `p_value`: the *p*-value of the regression.
- `beta`: the slope of the regression.
- `intercept`: the intercept of the regression.
- `data`: the name or index of the data.
- `data_column`: the index of the column in the data.
- `outcome`: the name or index of the outcome.

plots.r

plots.r

- `plots.r` is a small package for creating publication-ready plots.
- It currently only supports forest plots thanks to `metafor`, but I'm happy to add more upon request!

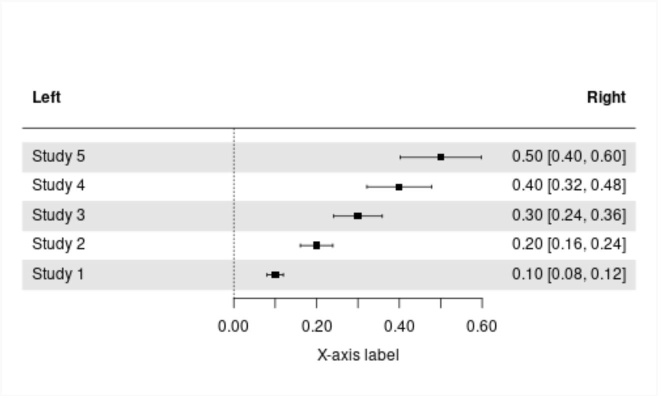
It can be installed by running the following in R.

```
install.packages(
"https://github.com/GMELab/plots.r/archive/refs/heads/master.tar.gz",
repos=NULL) # use .zip for Windows
# OR
devtools::install_github("GMELab/plots.r")
```

Basic Forest Plot

```r
plots.r::basic_forest_plot(
    x = c(0.1, 0.2, 0.3, 0.4, 0.5),
    se = c(0.01, 0.02, 0.03, 0.04, 0.05),
    width = 500,
    height = 300,
    name = "Example",
    header = c("Left", "Right"),
    slab = c("Study 1", "Study 2", "Study 3",
             "Study 4", "Study 5"),
    xlab = "X-axis label",
)
```

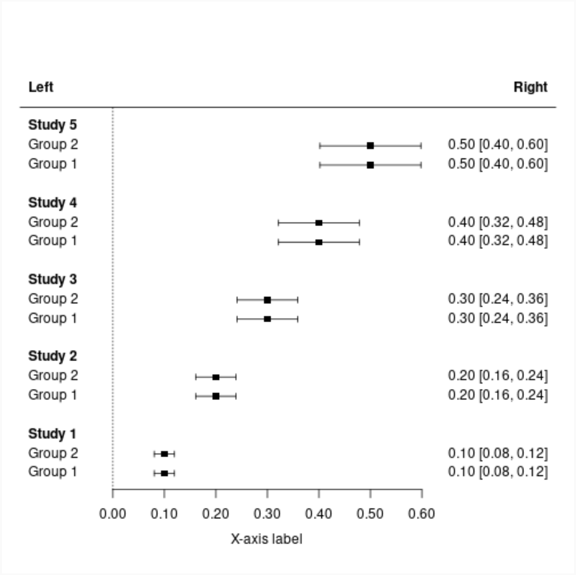
Basic Forest Plot



Grouped Forest Plot

```
plots.r::grouped_forest_plot(
    x = list(c(0.1, 0.2, 0.3, 0.4, 0.5),
             c(0.1, 0.2, 0.3, 0.4, 0.5)),
    se = list(c(0.01, 0.02, 0.03, 0.04, 0.05),
              c(0.01, 0.02, 0.03, 0.04, 0.05)),
    width = 500,
    height = 500,
    name = "Grouped_Example",
    header = c("Left", "Right"),
    slab = c("Study 1", "Study 2", "Study 3",
             "Study 4", "Study 5"),
    xlab = "X-axis label",
    glab = c("Group 1", "Group 2"),
)
```

Grouped Forest Plot



Conclusion

Conclusion

- Thank you for listening!
- Feel free to ask me questions anytime, I'm always happy to help or add new things. :)
- grafj1@mcmaster.ca or Josef Graf on Teams.