Rstats

Data Analysis with Rust

Presented at Brisbane Rust Group meeting on 28th July 2022

Libor Spacek

July 25, 2022

Overview

Introduction

My Crates

Rstats Usage

Implementation

New Concepts

Reflections on Rust in Science

Introduction

`Rstats` is primarily about characterising multidimensional sets of points (vectors), with applications to Machine Learning and Data Analysis.

Basic statistical measures and vector algebra provide self-contained tools for the multidimensional algorithms but can also be used in their own right. `Non analytical statistics` is used, whereby the 'random variables' are replaced by vectors of real data. Probabilities densities and other parameters are always obtained from the data, not from some assumed distributions.

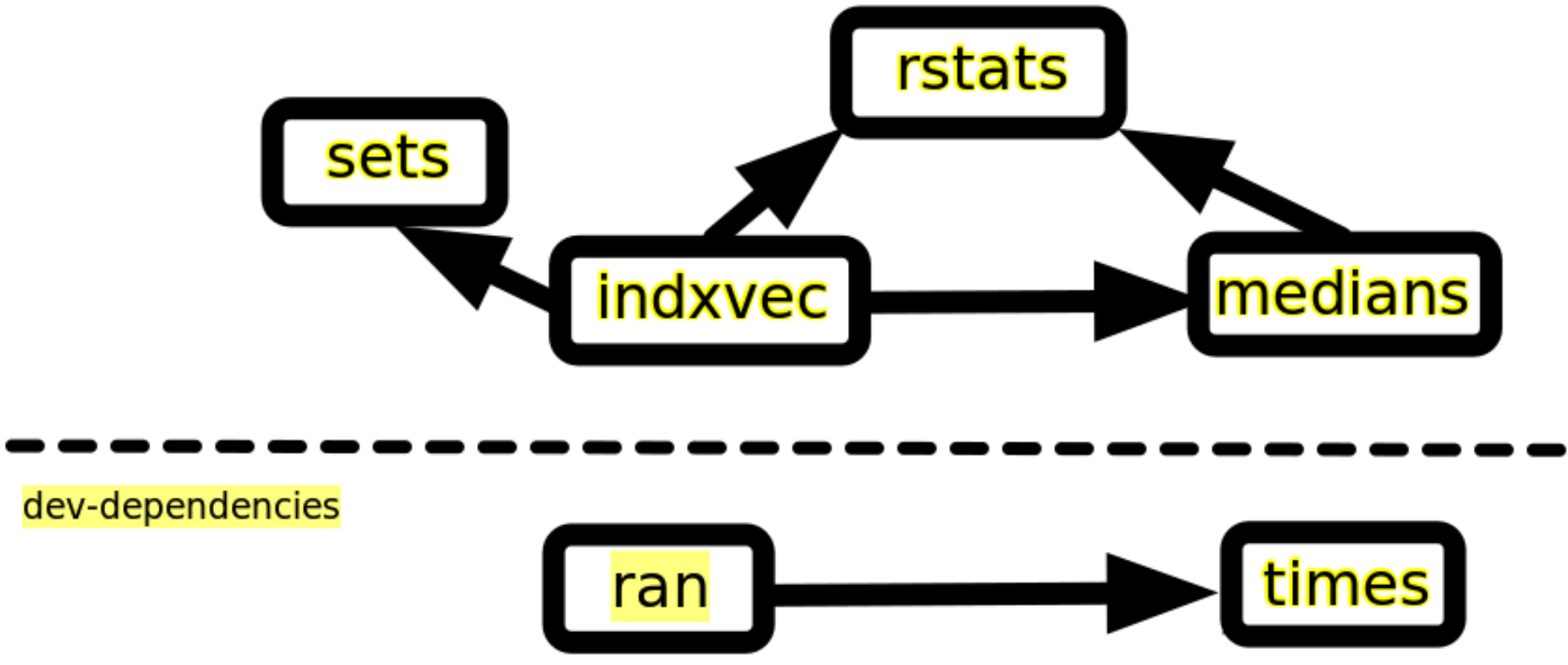
Our treatment of multidimensional sets of points is constructed from the first principles. Some original concepts, not found elsewhere, are introduced and implemented.

Example Application

Numer.ai competition (has a lot of numbers to crunch):

- Five classes to recognise
- Training set of 2,412,106 instances
- Tournament data of 1,412,933 instances
- Validation data of 539,659 instances
- Each instance has 1050 features. We view it as a point in 1050 dimensional space. Each outcome class is a cloud of hundreds of thousands of such points.
- Most data analysis and machine learning problems can be formulated in these terms, though not readily 'solved'.

My Crates



Rstats Usage

```
Insert into Cargo.toml file under [dependencies]
rstats = "^1"
```

Sructs:

```
use rstats::{Mstats,MinMax,Med,F64};
```

Traits:

```
use rstats::{Stats,Vecg,Vecu8,MutVecg,VecVec,VecVecg};
```

Testing and Benchmarking

It is recommended to read and run `tests/tests.rs`, with examples of usage. To run all the tests, use single thread in order to produce the results in the right order:

```
cargo test --release -- --test-threads=1 --nocapture
--color always
```

Timing comparisons use dev-dependencies crates:

```
times = "^0" # benchmark for comparing algorithms
ran = "^1" # random numbers, vectors and matrices
```

Implementation

The main constituent parts of Rstats are its generic traits. All data are Vecs of arbitrary length d (dimensionality). Vecs give fast random access. The different traits are determined simply by how many Vecs their methods handle: 1,2, or n.

Rstats Traits

- `Stats`: a single sample (of numbers)
- `Vecg`: methods of vector algebra and information theory operating on two vectors, e.g. scalar product
- `MutVecg`: some of the above methods, mutating self
- `Vecu8`: a few methods implemented more efficiently for u8
- `VecVec`: methods operating on n vectors in d dimensions
- `VecVecg`: as VecVec but take another generic argument, such as a vector of weights, e.g. to find the weighted geometric median. Where the weights express an unequal importance of points.

End Types

Vecs as arguments to most methods are defined over generic type parameter <T>. Thus the Vec items containing the actual data can be of any primitive numeric type.

Everything will work on user defined types as well, as long as PartialOrd, Copy and conversions to/from F64 are implemented for them.

End type `f64` is most commonly used for the results to maintain accuracy.

Documentation

For more detailed comments, plus some examples, see the source. You may have to unclick the ‘implementations on foreign types’ somewhere near the bottom of the page in the rust docs to get to it. Since these traits are implemented over the pre-existing Rust Vec type.

New Concepts

Statisticians talk about 'samples of random variables X,Y,..'
Vector algebraists talk about 'vectors **a,b,..**'
Set theoreticians talk about 'sets S,T,..'
Information theorists talk about 'data streams'.

All of the above are the same old Vecs !

Based on this important abstraction, Rstats is able to combine numerous methods from these four branches of mathematics in one compact and conceptually consistent crate.

Benefits of Geometric Median

Zero median vectors are generally preferable to the commonly used zero mean vectors.

In n dimensions (nd), many authors 'cheat' by using *quasi medians* (1-d medians along each axis). Quasi medians are a poor choice for characterisation of multidimensional data. In a highly dimensional space, they are slower to compute than our gm algorithm(s).

Specifically, all such 1d measures are sensitive to the choice of axis and thus are affected by rotation.

In contrast, analyses based on gm are axis (rotation) independent.

They are more stable, as medians have a 50% breakdown point (the maximum possible).

They are computed here by methods `gmedian` and its weighted version `wgmedian`, in traits `vecvec` and `vecvecg` respectively.

New Concepts in Stats Based Machine Learning

- `median correlation` - in one dimension (1d), our `mediancorr` method is to replace *Pearson's correlation*. We define *median correlation* as cosine of an angle between two zero median vectors (instead of Pearson's zero mean vectors).
- `gmedian` - fast multidimensional geometric median (gm) algorithm.
- `madgm` - generalisation of robust data spread estimator known as 'MAD' (median of absolute deviations from median), from 1d to nd.
- `contribution` - of a point to an nd set. Defined as gm displacement when the point is added. Not the same as radius, as it depends on positions of all the other points.
- `comediance` - is computed as a `covar` matrix from the *zero median feature vectors*, instead of the usual *zero mean vectors*.

Contribution

Key question of Machine Learning (ML) is how to quantify the contribution that an example point (typically a member of some large nd set) makes to the recognition concept, or outcome class, represented by that set.

In answer to this, we define the `contribution` of a point as the (vector) change Δgm, caused by adding that point. Outlying points generally make greater contributions. However, gm is far less sensitive to outliers than centroid.

`Contribution` is computed efficiently from an existing gm.

Reflections on Rust in Science

- \+ Speed and Parallelism
- \+ Practically no runtime errors
- +/- Invariance of difficulty: now lots of compiler errors
- - Unable to print any Vecs. Had to implement it myself in `indxvec::Printing`. It is non-trivial (for generic Vecs) and beginners should not have to do this.
- - Generic types: surprisingly hard battle just to use more than one standard numeric end type in Vecs.
- - Annotations needed in unpredictable ways (type inference bug?). Our gm is of concrete type `&[f64]`, instead of `&[T]`, so we have to annotate the generic methods calls with the turbofish, e.g.: s.vsub::<f64>(gm).

https://leanpub.com/bashforfun

Bash for Fun

Bash Programming: Principles and Examples

This book is 100% complete
COMPLETED ON 2022-01-08

Libor Spacek

$9.99
MINIMUM PRICE

$19.52
SUGGESTED PRICE

YOU PAY

$19.52

AUTHOR EARNS

$15.61

UNIT PRICE IN US $

$19.52

EU customers: Price excludes VAT.
VAT is added during checkout.

Add Ebook to Cart

Add to Wish List

Table Of Contents