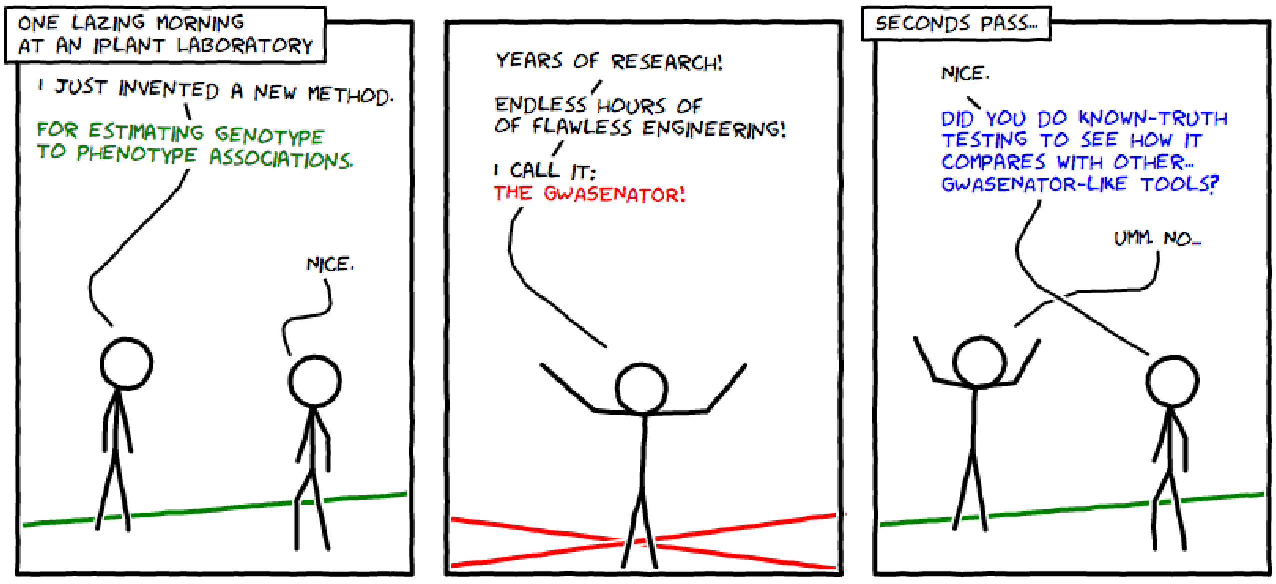
Validate: A Pipeline for Evaluating Performance for GWAS/QTL Tools Using Known-Truth Datasets

Ann Stapleton, Kurt Michels, and Dustin Landers
University of North Carolina Wilmington

Abstract

Understanding the effectiveness of Genome-Wide Association (GWAS) and Quantitative Trait Loci (QTL) analytical tools under various situations is crucial to deciding which tools are best given a particular problem. Validate provides a way to return classification and regression performance measures for large amounts of tool outputs generated using known-truth simulations. We also provide solutions for aggregating hundreds or thousands of outputs in to a single folder on the iPlant data store, so that Validate can be used.

Contents


ONE LAZING MORNING AT AN IPLANT LABORATORY
I JUST INVENTED A NEW METHOD.
FOR ESTIMATING GENOTYPE TO PHENOTYPE ASSOCIATIONS.
NICE.
YEARS OF RESEARCH!
ENDLESS HOURS OF OF FLAWLESS ENGINEERING!
I CALL IT:
THE GWASENATOR!
SECONDS PASS...
NICE.
DID YOU DO KNOWN-TRUTH TESTING TO SEE HOW IT COMPARES WITH OTHER... GWASENATOR-LIKE TOOLS?
UMM. NO...

1 Introduction

1.1 About the Developers

Project Lead and Principal Investigator	Dr. Ann Stapleton works at the interfaces, such as the junctions between research and teaching, individual research projects and large collaborative projects, the organization of international meetings and high-school teacher inquiry labs. Most recently, she has worked at the interface between plant biology and software engineering, leading the way to broad methods applicable to evaluating genotype-to-phenotype analytical methods.
Statistical Analyst and Developer	Dustin Landers received a BS from Appalachian State where he learned a lot about survey research and using statistics to solve problems and answer questions. He continued his education at UNCW by studying Applied Statistics. Moving from world of polling and survey statistics to the that of Big Data, he has an interest in bridging the gap between statistics and software engineering and seeing the combined discipline brought to bear on problems once considered impossible.

1.2 Why Validate?

To test known-truth data sets, we had the idea of looking at a bunch of ROC plots for each simulation run in testing a certain tool. Of course, this was never really feasible because any one simulation run could be atypical. We asked ourselves how could we run lots of simulations through a tool, and test the outputs in a way that gave us insight in to the performance of these GWAS and QTL tools?

It was obvious to us that iPlant gave us the resources to make this happen, but we had to make this as easy as possible if we expected people to use it.

We sought after summary measures of performance in order to break down the complexity of any individual simulation. We ultimately decided on several classification measures such as the area under the ROC curve, H, and the Kolmogorov-Smirnov statistic. For evaluating effect sizes, we include root mean squared error and mean absolute error.

But how do we run all these simulations? How do we store them all in a single location so we can calculate these measures? How do we visualize them?

These are the problems we sought to solve. For each problem, we have our own solution. We also left them divided. This way you can decide whether to use our solution or to use your own if you have a unique scenario.

1.3 Getting iPlant Credentials

1.4 Getting Your Application on an API

1.5 Where Can I Get These?

1.6 How to Use This Manual

Evaluating a GWAS/QTL tool with Validate is essentially four major steps after your application is installed on either the iPlant Foundation API or the Agave API:

→ Section 2

1. **Running the Tool With Simulations As Inputs.** This step involves deciding what simulations to use and then iterating over those simulations and submitting them as job requests through the API. We recommend using some sort of scripting method that you are comfortable with. At this point, we don't have a standalone application for submitting jobs. We use rPlant, which is freely available R package that allows you to connect with iPlant's API layer to submit job requests.

→ Section 3

2. **Aggregating the Outputs into a Single Folder.** This part is a bit of a logistical exercise. You need to put all the tool outputs (from Step 1) that you want to be analyzed using the same known-truth metadata in to aggregate folders. For example, say we are using simulations that are generated using varying levels of heritability. Since the varying heritability values in essence produce different SNP effects, we need to essentially run Validate three separate times. Validate requires an input on an entire folder, and then iterates over those tool outputs. So the first step here, is to decide how many different runs we need to do and then create that many folders. We provide a GUI tool, *Aggregate* that allows you to select files from multiple folders on your iPlant data store (multiple runs of a single tool) and move those (or aggregate them) in to a single folder.

→ Section 4

3. **Running *Validate* on the Aggregated Folder.** Once you have all your outputs in aggregated folders. You can simply log in to the iPlant Discovery Environment and run *Validate* on that folder. You must have at least two columns with header names in the outputs: The name of the SNP column, and the name of threshold column (such as P-value). Further, if you wish the get back effect size estimation errors, you must also know the column on the estimated effect size column (for example, PLINK's is BETA). Text files with known SNPs and effect sizes must also be included. Once you submit Validate, you will receive a notification when it is completed.

The *Validate* output will be columns of performance measures for each tool output.

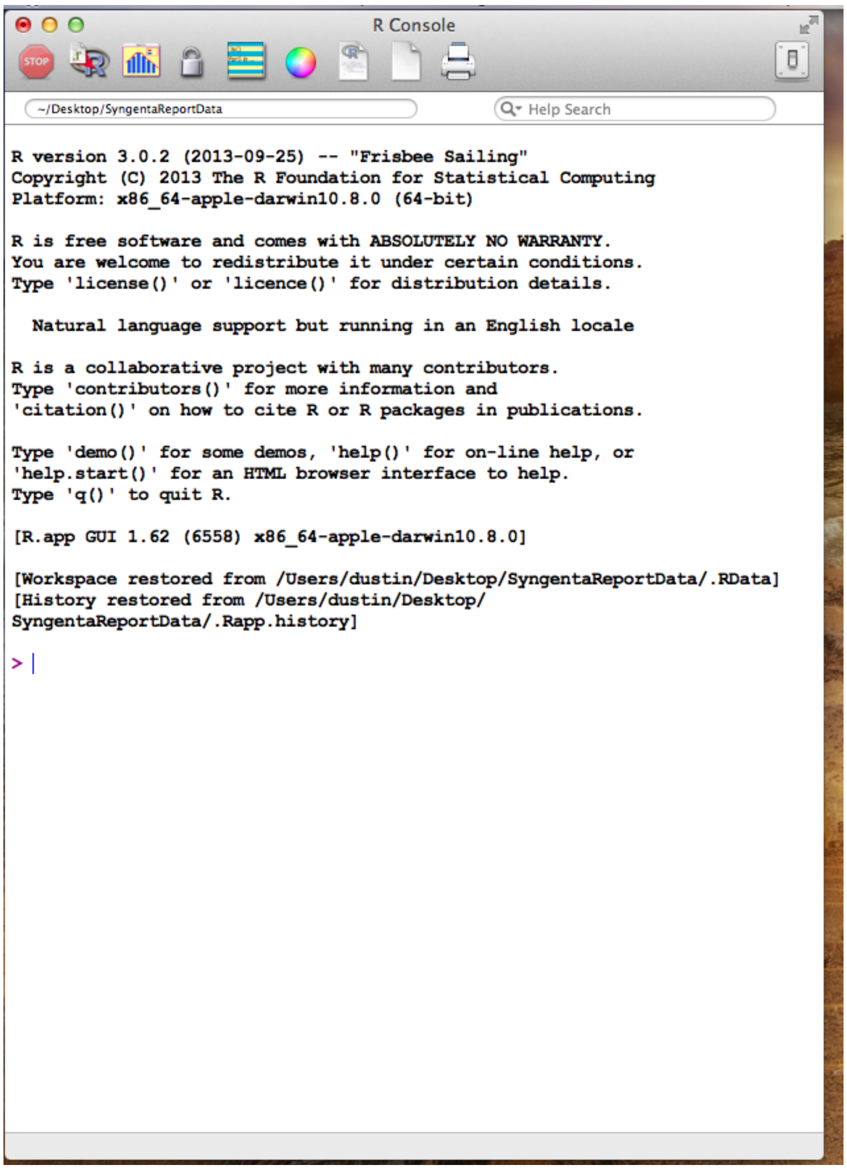
→ Section 5

4. **Running *Demonstrate* on the Validate Output.** Finally, once you have outputs from *Validate* (which may be more than one), you need to combine them and test the differences in your measures between your simulation's parameters. Demonstrate is an R package (to be installed on the iPlant DE) that quickly combines all your results files in a folder you specify and returns sciplot factorial graphs for parameters you specify. It also returns the combined results file so that you can perform your own analyses.

2 Running Simulations

2.1 How To Run Simulations with rPlant

We will first show you how to batch run simulations with rPlant. First, we are using the latest R version, 3.0.2. When you open the R console, it should look something like this (we are using a Mac, so yours may look slightly different, but it will function essentially the same).



```
R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin10.8.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[R.app GUI 1.62 (6558) x86_64-apple-darwin10.8.0]

[Workspace restored from /Users/dustin/Desktop/SyngentaReportData/.RData]
[History restored from /Users/dustin/Desktop/
SyngentaReportData/.Rapp.history]

> |
```

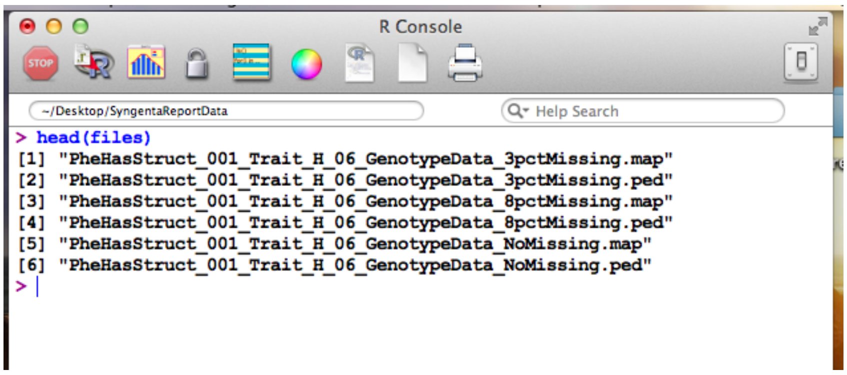
If you haven't already installed rPlant, we recommend that you do that. After that, you need to validate your iPlant credentials so that you can access your simulation files. Then, you will need to get a list of the names of your simulations files and iterate over them in a for loop, submitting jobs to your installed application.

The following code is to just an example of how the submission may go assuming your known-truth data sets are in something like a raw format, where each known-truth data set is stored in a single file. rPlant assumes you already include your iPlant data store username in the files path, so that "iplant/home/username/simulations" is just written as "simulations" or "iplant/home/username/analyses" is just written as "analyses". This is, of course, because validating your credentials allows you to not have to keep typing you user information over and over again.

```
> install.packages("rPlant")
> require(rPlant)
> Validate("username", "password")
> mydir <- ListDir("simulations")
> for (i in 1:nrow(mydir)) {
+         SubmitJob(application="myGWASapplication",
+         file.path="simulations", file.list=list(mydir[i,1]))
+         print(i)
+ }
```

First, you need to make sure of the specific application name assigned to your tool in the API and of any special parameters or inputs required in order to run. For example, PLINK requires a PEDMAP format. The PEDMAP format is essentially two files, the .PED and the .MAP file for each known-truth data set.

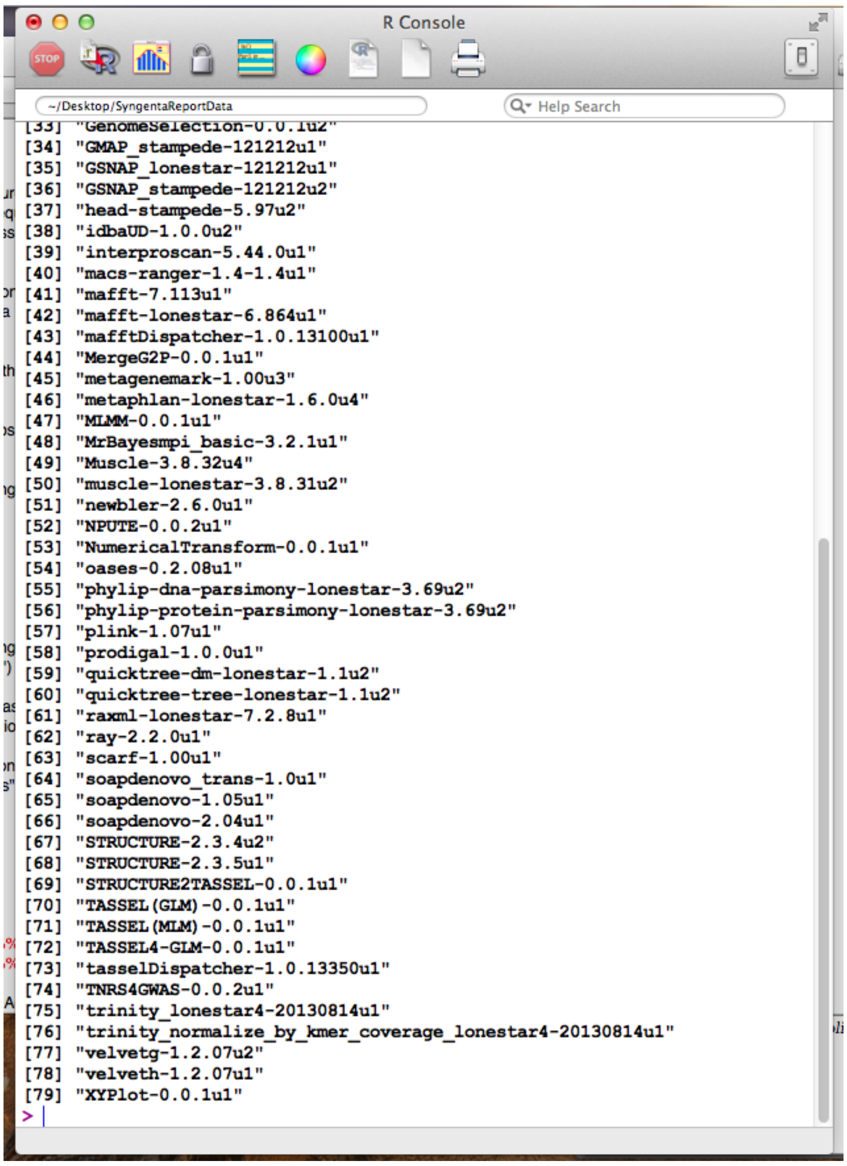
For example, our simulation files are stored in a shared folder, where each odd file number is a .PED and each even is a .MAP. Each odd is a .PED file and the subsequent file number is a .MAP like so:



```
> head(files)
[1] "PheHasStruct_001_Trait_H_06_GenotypeData_3pctMissing.map"
[2] "PheHasStruct_001_Trait_H_06_GenotypeData_3pctMissing.ped"
[3] "PheHasStruct_001_Trait_H_06_GenotypeData_8pctMissing.map"
[4] "PheHasStruct_001_Trait_H_06_GenotypeData_8pctMissing.ped"
[5] "PheHasStruct_001_Trait_H_06_GenotypeData_NoMissing.map"
[6] "PheHasStruct_001_Trait_H_06_GenotypeData_NoMissing.ped"
> |
```

As can be seen, for our particular structure we would need to submit two files for each run. So let's walk through it. We first can list all the apps in order to find ours (which we have already installed), so that we can get the exact name that we need to use for the SubmitJob function.

```
> ListApps()
```



```
[33] "GenomeSelection-0.0.1u2"
[34] "GMAP_stampede-121212u1"
[35] "GSNAP_lonestar-121212u1"
[36] "GSNAP_stampede-121212u2"
[37] "head-stampede-5.97u2"
[38] "idbaUD-1.0.0u2"
[39] "interproscan-5.44.0u1"
[40] "macs-ranger-1.4-1.4u1"
[41] "mafft-7.113u1"
[42] "mafft-lonestar-6.864u1"
[43] "mafftDispatcher-1.0.13100u1"
[44] "MergeG2P-0.0.1u1"
[45] "metagenemark-1.00u3"
[46] "metaphlan-lonestar-1.6.0u4"
[47] "MLMM-0.0.1u1"
[48] "MrBayesmpi_basic-3.2.1u1"
[49] "Muscle-3.8.32u4"
[50] "muscle-lonestar-3.8.31u2"
[51] "newbler-2.6.0u1"
[52] "NPUTE-0.0.2u1"
[53] "NumericalTransform-0.0.1u1"
[54] "oases-0.2.08u1"
[55] "phylip-dna-parsimony-lonestar-3.69u2"
[56] "phylip-protein-parsimony-lonestar-3.69u2"
[57] "plink-1.07u1"
[58] "prodigal-1.0.0u1"
[59] "quicktree-dm-lonestar-1.1u2"
[60] "quicktree-tree-lonestar-1.1u2"
[61] "raxml-lonestar-7.2.8u1"
[62] "ray-2.2.0u1"
[63] "scarf-1.00u1"
[64] "soapdenovo_trans-1.0u1"
[65] "soapdenovo-1.05u1"
[66] "soapdenovo-2.04u1"
[67] "STRUCTURE-2.3.4u2"
[68] "STRUCTURE-2.3.5u1"
[69] "STRUCTURE2TASSEL-0.0.1u1"
[70] "TASSEL(GLM)-0.0.1u1"
[71] "TASSEL(MLM)-0.0.1u1"
[72] "TASSEL4-GLM-0.0.1u1"
[73] "tasselDispatcher-1.0.13350u1"
[74] "TNRS4GWAS-0.0.2u1"
[75] "trinity_lonestar4-20130814u1"
[76] "trinity_normalize_by_kmer_coverage_lonestar4-20130814u1"
[77] "velvetg-1.2.07u2"
[78] "velveth-1.2.07u1"
[79] "XYPlot-0.0.1u1"
> |
```

We can see that the exact name of PLINK on the Foundation API is "plink-1.07ul". We will revisit the earlier explanation to see how we would submit a PLINK job using the PEDMAP format.

```
> install.packages("rPlant")
> require(rPlant)
> Validate("username", "password")
> mydir <- ListDir("simulations")
> for (i in 1:nrow(mydir)) {
+         SubmitJob(application="plink-1.07ul",
+         file.path="simulations", file.list=list(mydir[i,1],
+                   mydir[i+1,1]))
+         print(i)
+ }
```

This should iterate over each simulation in our data set and submit a PLINK job to the iPlant hardware. Since iPlant uses a queuing system, this effectively distributes your workload over many computer nodes as needed. This is obviously helpful in achieving a high volume of runs, even outside of known-truth testing. Now you know how!

2.2 Using rPlant to Sample From Your Known-Truth Data Sets

Let's say, for example, that we had 600 total known-truth data sets: Our known-truth datasets are generated from hundred different genotypes, with three different types of simulated missing values such as 0%, 3%, and 8%. In addition, we have data generated with and without population structure (meaning the stochastic processes generated the data have either a single population mean or varying group means).

Say that we didn't want to run all of them, but maybe just half of them. In many cases, you may have substantially more than 600 and thus sampling makes much more sense then it does in this example.

We can use some scripting in R and rPlant to accomplish the same thing we did earlier but with some random sampling. Here is the code we used to sample 300 known-truth PEDMAPs for PLINK.

```
> set.seed(192031099)
> require(rPlant)
> Validate("username", "password")
> mydir <- ListDir("simulations")
> genos <- matrix(nrow=1200,ncol=1)
> break.data <- function(x,y=2) {
+       broken <- unlist(strsplit(x,'_',TRUE))
+       return(broken[y])
+ }
> mydir <- mydir[,1]
> for (i in 1:length(mydir)) {
+       genos[i] <- break.data(files[i],2)
+       gens <- na.omit(unique(genos))
+ }
> oddsamp <- seq(1,11,2)
> totest <- list()
> for (i in 1:length(genoa)) {
+       myfiles <- list()
+       for (j in 1:length(files)) {
+               if (breakdat(files[j],2)==genos[i]) {
+                       myfiles <- append(myfiles, files[j])
+               }
+       }
>       myfiles <- myfiles[sample(oddsamp,2,FALSE)]
>       myfiles <- unlist(myfiles)
>       for (k in 1:length(files)) {
+               for (p in 1:length(myfiles)) {
+                       if (files[k]==myfiles[p]) {
+                               totest <- append(totest,k)
+                       }
+               }
+       }
+ }
> totest <- unlist(totest)
> for (i in 1:length(totest)) {
+       SubmitJob(application="plink-1.07ul", file.list=
+               list(files[totest[i]], files[totest[i]+1]),
+               file.path="simulations")
+       print(i)
+ }
```

Every situation is unique in this case, which is part of the reason we have yet to develop a graphical user interface method for submitting jobs. All that we did in the code above, is demonstrate how rPlant may be used to break up some files that had naming conventions that were separated by underscores. If you look back to the earlier image that showed what our known-truth data sets looked like, you would see that phenotype, heritability information, and population structure are all in the naming format for each data set.

This is an important piece of information, because without an appropriate naming structure, there would be no way for us to keep metadata on the known-truth data sets throughout this testing process.

We break up the names to grab the pieces about phenotype information in order to essentially stratify-sample our data sets by genotype. We then have six data sets for any one genotype (three missing values types and two population structure types), and then we just pick two at random.

This insures that we include all of our genotypes in our runs, but there are simpler ways to do this, that basically just random sample from all of our data sets. The following code again assumes you are using PEDMAPs to run PLINK, but that you just want to sample a random 300 (as opposed to a stratified-random sample).

```
> set.seed(45876572)
> require(rPlant)
> Validate("username", "password")
> mydir <- ListDir("simulations")
> mydir <- mydir[,1]
> oddsamp <- seq(1,599,2)
> totest <- sample(oddsamp, 300, FALSE)
> for (i in 1:length(totes)) {
+       SubmitJob(application="plink-1.07ul", file.list=
+               list(mydir[totest[i]], mydir[totest[i] + 1],
+               file.path="simulation"
+ }
```

3 Aggregating with Aggregate

3.1 How to Use the GUI Aggregate Tool

The second major step is accomplished once all the runs have finished. In our example, depending on the resources available, can finish in a range from ten seconds to two and half minutes per run. Let's say that

3.2 How to Use the Aggregate Function in rPlant

4 Validating with Validate

4.1 How to Use Validate on the DE

5 Visualizing the Validation

5.1 How to Use the Demonstrate R Package

6 Acknowledgements

hmeasure.net