# **runPRS script overview:**

**Description:**

**By Adrian Campos v1.5 22/10/18.**

**This script performs automatically a classical PRS pipeline on the genotype data from the genepi group (for more info on this script refer to its specific documentation). It should be readily modifiable to be run on other genotype datasets and clusters. It was developed on python 3.6 and requires the following libraries (all available on hpc):**

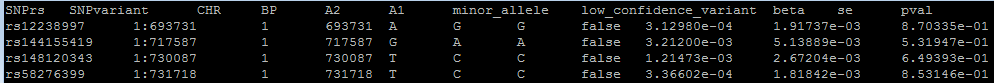
* **Pandas**
* **re**
* **argparse**
* **subprocess**
* **os**
* **glob**
* **time**
* **sys**
* **datetime**
* **socket**

**The code is available at (feel free to copy or add to your path):**

**/mnt/backedup/home/adrianC/bin/runPRS**

**If permission denied then use the copy provided and add it to your bin. Feel free to copy it, edit it or just link it to your path to be able to execute it from wherever (you have to edit your $HOME/.bashrc file and add the line export PATH="$PATH:/mnt/backedup/home/adrianC/bin/"). From now on this documentation assumes that you have either added “/mnt/backedup/home/adrianC/bin/” to your path, or copied the scripts to a place discoverable by your own path environmental variable (and made them executable etc etc).**

**To run this script you require a working directory (accessible by the nodes {not in labdata}) that contains the following files:**

1. **yfp (your favourite phenotype) summstats:**

**This is a tab or space delimited file containins at least the following columns (extras do not matter, and it is capslock insensible so SNP==snp):**

* **SNPrs [snp,snpid,rsnumber,rs,snprs] – rsnumber**
* **CHR [chr,chrom,chromosme]- chromosome**
* **BP [bp,pos]– basepair**
* **A2 [a2,ref,reference,oa] – reference allele (non-effect allele CAREFUL WITH THIS ONE)**
* **effect\_allele [A1,a1,minor\_allele,alt,effect\_allele,ea] – alt, effect or minor allele (CAREFUL WITH THIS ONE)**
* **beta [BETA,effsize,beta,or]– effect size or OR**
* **pval [p,pvalue,pvalues]– pvalues**

**NOTE: Column order doesn’t matter, the headers should be identical or one of the [synonyms]. Extra columns will be ignored**

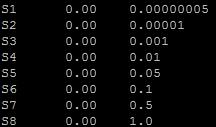
**If you do not have rsnumber you can match your data to our rsnumbers by using a script like this:**

**awk 'NR==FNR{id[$1]=$2;next}($1 in id) {print id[$1]"\t"$0}' allchr.markerlist.IDs GWASUMSTATS > GWASUMSTATS\_rs**

**Can be found in the release metadata**

1. **pvalue.ranges file:**

**Name lower upper**



**These two files are the only input needed and currently accepted. The pvalue.ranges file must lie in your working directory (where your input sumstats are). This script requires the python module to be loaded (module load python) and at least 15Gb of memory so it should be ran either interactively or as a Job using qsub or a PBS script. If you try to run it in the login node you will receive the following error:**

**RuntimeError: You should run this script interactively or via qsub, not in login nodes**

**How to run interactively:**

**Start interactive session:**

**$ qsub -I -l mem=32gb,walltime=10:00:00**

**Load python module**

**$ module load python**

**Move to the working directory (either as you would normally do or):**

**$ cd $PBS\_O\_WORKDIR**

**Submit the command**

**$ runPRS INFILE JOBNAME**

**How to run it with a PBS script (more reproducible):**

**Create the script file (where the input files are):**

**$ touch PRS\_SUBMIT.PBS**

**Edit the script file:**

**$ nano PRS\_SUBMIT.PBS**

**Add the following lines:**

**#!/bin/bash**

**##PBS -N JOBNAME**

**#PBS -r n**

**#PBS -l mem=20GB,walltime=5:00:00**

**module load python**

**cd $PBS\_O\_WORKDIR**

**runPRS.py INFILE JOBNAME**

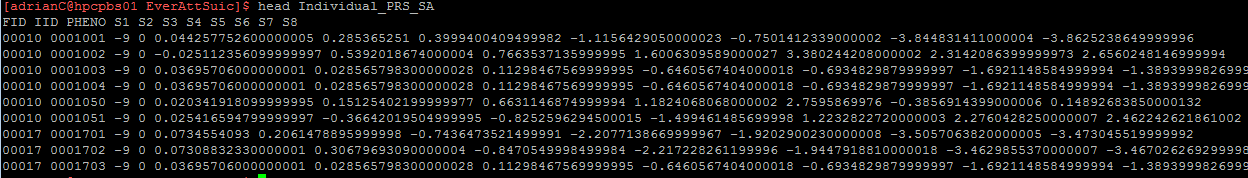
**To submit it run the following:**

**$ qsub PRS\_SUBMIT.PBS**

**Important Note: This script requires the submission to called from inside the working directory (where your input and pvalue.ranges files are located), as it will use the environmental variable $PBS\_O\_WORKDIR when it submits ‘children’ jobs.**

**A log named *RunPRS.log* contains all of the output and can be used to debug. If, for example, you forget to put in that working directory the pvalue ranges then the plink error message will be stored there (as well as in the job.o\*\*\*\* files generated by hpc).**

**Output:**

* **Individial\_PRS\_JobName – Final compiled output with the following columns:**
  + **FID**
  + **IID**
  + **PHENO – usually NULL unless you modify this script to include it**
  + **PRS1 – with the name you placed on your pvalue.ranges (e.g. S1)**
  + **PRS2**
  + **…**
  + **PRSN – the last cutoff you used (ordered based on the name)**

**Summary:**

* **This calculates, with a clumping procedure, a PRS on our genotype data based on summary statistics from other population.**
* **It is a python script that should not be run on login nodes.**
* **To run it you need two files and choose a job name:**
  + **GWAsumstats - with the format specified above**
  + **pvalue.ranges – currently with this exact name and format (range may vary)**
  + **Job name – it is required**
* **An example of how the script is to be ran (from within the working directory):**
  + **$ runPRS.py infile jobname**
  + **Be sure to run it from an interactive session or via qsub**
* **Several outputs created for debugging/reproducibility, but the important is called: *Individual\_PRS\_jobname* (or the standard hpc errors \*.e\*)**

**The next steps involve assessing the variance explained by this calculated PRS in the Twin data. Because they are related individuals a lmm analysis using GCTA should be performed. (see PRSlmm Documentation) below**

**PRSlmm Overview (refer to its documentation for more info)**

*PRSlmm* is a follow up script for runPRS. It performed a linear mixed effects model to assess the variance explained of a PRS on related individuals. It receives the following four required positional arguments (for optional arguments see its detailed docs):

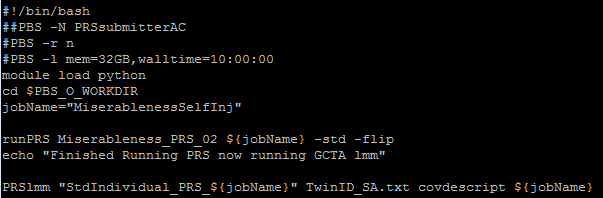
* prsfile PRS file: FID IID PHENO S1 S2 .... (don’t worry about PHENO)
* phenofile Phenotypes file: FID IID Age Sex PC1 ... PCX ... Pheno1 Pheno2 Pheno3 ... PhenoK
* covdescript Describe the variables on phenofile: ColumnName [covar|qcovar]
* jobname Job Name

Example run:

*$PRSlmm.py StdIndividual\_PRS\_MiserablenessSelfInj TwinID\_SA.txt covdescript MiserablenessSelfInj*

Where the inputs are (position important):

* PRS results file (can be results from runPRS)
* Phenotype file
* Covariate description file
* Job name (again) can be same as before

It is all designed so that it can be run right after the runPRS script, for example in a PBS:

**In the example above (should be provided to you) the input GWA sumstats are named Miserableness\_PRS\_02, and the job name is MiserablenessSelfInj. Setting the jobname above allows you to run both scripts one after the other as in this example.**

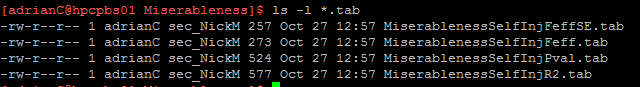
**The PRS lmm the output consists of four files:**

1. **${jobname}Feff.tab -> fixed effects (regression coefficients)**
2. **${jobname}FeffSE.tab -> standard error**
3. **${jobname}R2.tab -> Variance axplained**
4. **${jobname}Pval.tab -> Pvalue**

**plotPRS Documentation**

This script makes it very easy to plot the PRS results, at least in an exploratory quick way. It is also done in python and using *seaborn*. As you can imagine plotting requires tweaking many parameters, so it might be recommendable to take this script as a base and modify it each time to fit your purposes. The example is similar to most cases, where a PRS is assessed against a small set (n<10) of phenotypes. Then, the plots might be made to look fine with the default parameters. When several phenotypes are to be assessed, the script contains comments explaining where changes can be made to tweak parameters. Also on a recent update one can specify a *–kwrgs* flag that allows you to set virtually any parameter (such as xlimit, xticks, xticklabels, ylimit , title, google “matplotlib axes properties” to see which properties and the syntax).

This script takes as a single mandatory input the jobname (or base of the PRSlmm output). In our example the GCTA output looked like this:



**Therefore, the input for *plotPRS* would be: *MiserablenessSelfInj* the script automatically detects the R2 and Pval files to generate the plots using python.**

**The flags that control plotPRS are:**

**-plt If this is set to ‘bars’ then for each phenotype barplots will be created. If set to ‘barsjoint’ it will automatically create a *joint barplot***

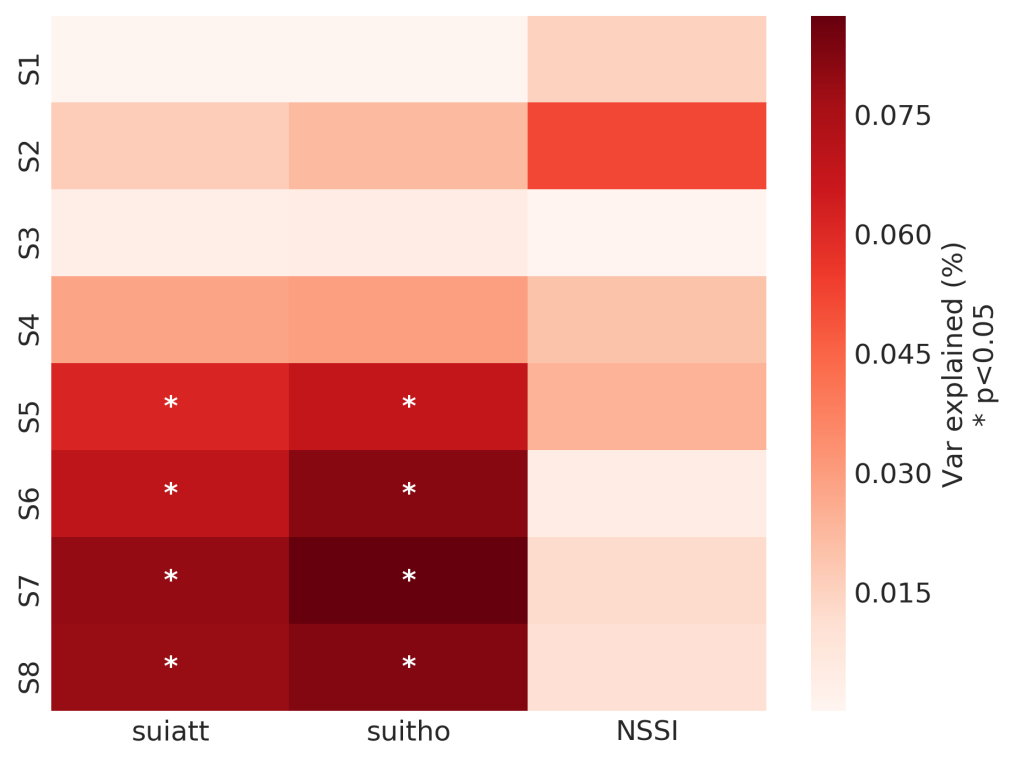
**-nvar Number of variables to correct for multiple testing (number of phenotypes studied)**

**-fontsize The size of the font for everything in heatmaps (x and y labels, asterisks etc)**

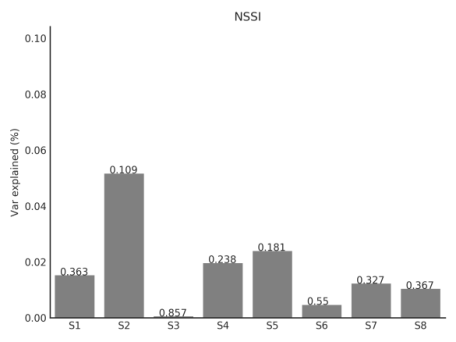
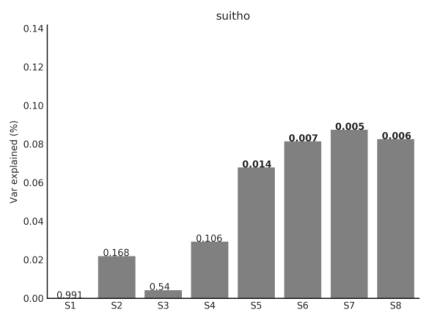
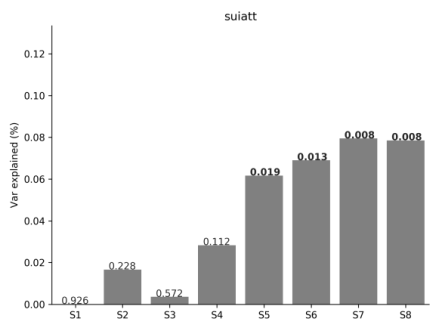
**-transpose Whether to transpose the heatmap (just for aesthethics).**

**A run example:**

*$plotPRS.py MiserablenessSelfInj* this generates a heatmap:



**On the other hand running:**

*$plotPRS.py MiserablenessSelfInj –plt bars* Would generate barplots for each phenotype:

*$* *plotPRS.py MiserablenessSelfInj -plt barsjoint -nvar 3 -kwrgs "{'xlabel':'phenotype'}" -heatCol 'red’*

Will generate a plot like the one below:

**

This script will automatically add an \* to the significant associations (corrected using nvar variable):

# 

# Running the whole pipeline:

The whole pipeline actually needs the following input data:

* GWAsumstats – Summary statistics
* pvalue.ranges – This file with this name is needed for the PRS to be calculated
* phenofile – The file with phenotypes (FID IID PHENO COV1 COV2 QCOV1 QCOV2)
* covdescritp – Description of covariates for the GCTA lmm

If these three four files exist the following sequence of codes should run a prs:

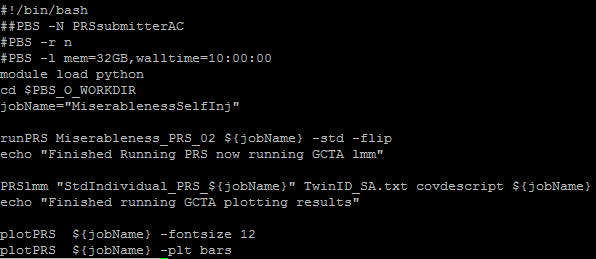
*$runPRS.py GWAsumstats ${jobname}*

*$PRSlmm.py “Individual\_PRS\_${jobname}” phenofile covdescript ${jobname}*

*$plotPRS.py ${jobname} -fontsize 12*

*$plotPRS.py ${jobname} -plt barsjoint*

Following this logic on the example provided, the final code would look like this:



This all goes on a script that can be submitted using *qsub*. After reading all this you are ready to run your first PRS. For testing you can copy the “PRSpipelineExample” into your hpc working directory and run the example *as is*  by running

*$qsub SUBMIT\_PRS.PBS*

# Debugging the different steps

Ideally the code searches for errors and mistakes and outputs them to the terminal. The main part where something might fail is using the runPRS script. An output named RUNPRS.log will be the strating point to check for errors, most common mistakes include:

* Header interpretation error (missing a column name, or having two names that match the same interpretation)
* Missing the pvalue.ranges file
* Trying to run the script on the login node (it will fail to avoid using all the login node memory)
* Not having loaded the python module before trying to run it
* Missing a positional input, or putting them in the wrong order
* Selecting an inexistent dataset such as HRC1 (instead of HRCr1)

Checking the log and any intermediate output and error files can give an idea of possible errors (XXX.oXXXXX and XXX.eXXXX). Finally all the intermediate PBS scripts are kept and may be run using qsub or interactively to see why they fail.

NOTE: As GCTA outputs errors on its log and not the .e sometimes it may fail without showing errors to the pipeline’s output (e.g. if a covariate has just one level).