LDpred2 Pipeline for Polygenic Risk Score Prediction

This notebook implements the pipepline for genome-wide PRS prediction using R package bigsnpr and PLINK v1.9 implementing the LDpred2 method. The pipeline is based on tutorials from Florian Privé and Shing Wan Choi.

Author: Mengyu Zhang, mengyu1307@gmail.com with input from Gao Wang

Overview

This pipeline is developed to integrate GWAS summary statistics data and LD reference panel to compute PRS using the LDpred2 model. The pipeline can also perform phenotypes prediction using PRS computed with other covariates, when individual sample genotype and covariates information are available. are available. When the true phenotypes are also available, the pipeline provides regression analysis for the association between PRS, covariates and phenotypes.

Methods

PRS model

Typically, phenotype Y for N individuals is modeled as a linear combination of M genetic effects, P covaritates and an independent random noise shown as formula (1).

$$Y = \sum_{i=j}^{M} X_j \beta_j + \sum_{j=1}^{P} Z_j \alpha_j + \varepsilon \tag{1}$$

Assuming genotype X is centered and scaled, the (marginal) least-squares estimate of an individual marker effect is $\hat{\beta}_j = X_j' Y / N$.

LDpred is a Bayesian PRS that account for the effects of linkage disequilibrium (LD). It estimates posterior mean causaul effect sizes from GWAS summary statistics by assuming **genetic *architecture* prior** and **LD information from a reference panel**.

LD reference panel

The choice of reference panel is crucial to the prediction performance of LDpred model. Population structure should ideally be the same (and in practice as similar as possible) between reference panel and training data that summary statistics are calculated from.

Also, a preliminary quality control on genotype reference data should be conducted. This includes but not limited to

1. Filter individuals with much (>10%) genotype calls missing and filter SNPs that had a missing rate more than 1% and a minor allele frequency (MAF) greater than 1%

2. Remove SNPs that have strand ambiguous nucleotides, i.e., A/T and G/C.

In addition to SNP filtering, SNP flipping may be necessary. It is of importance that GWAS summary stats has the same effect allele, and non-effect allele. Therefore, if the alleles of a SNP in the summary stats is the reverse of the alleles of the reference panel, the sign of z-scores (also effect size, log odds ratio, etc) is need to be flipped. At the end, summary statistics and reference panel will be matched on the basis of the SNP rsID.

Independent validation cohort or dataset can be used as LD reference panel. Here is an example in the literature: Bjarni J. Vilhja´lmsson (2015) used 1000 Genome, Hapmap imputed cohort validation dataset as reference panel. He analyzed six large summary-statistics datasets in his study, including schizophrenia with European and non-European ancestry, multiple sclerosis (MS), breast cancer (BC), coronary crtery cisease (CAD), type II diabetes (T2D) and height. For schizophrenia with European ancestry, they used the Psychiatric Genomics Consortium 2 (PGC2) SCZ summary statistics excluding the ISC (International Schizophrenia Consortium) cohorts and the MGS (Molecular Genetics of Schizophrenia) cohorts. ISC and MGS datasets are used as validation datasets. For non-European ancestry, MGS validation datasets was used as an LD reference. To coordinate the summary statistics from Asian population, they used overlap among 1000 Genomes imputed MGS genotypes and the 1000 Genomes imputed validation genotypes for the three Asian validation datasets (JPN1, TCR1, and HOK2), respectively. For African-American (AFAM) population, they used overlap among the 1000 Genomes imputed MGS genotypes and the HapMap 3 imputed AFAM genotypes.

The reference panel applied in this pipeline is 1000 genomes project (phase 3) data including 503 (mostly unrelated) European individuals and ~1.7M SNPs in common with either HapMap3 or the UK Biobank. EUR includes Utah Residents (CEPH) with Northern and Western European Ancestry, Toscani in Italia, Finnish in Finland, British in England and Scotland and Iberian Population in Spain.

LDpred prior effect size model

The **prior for effect sizes** is a point-normal mixture distribution with 2 hyper-parameters:

Heritability (parameter) explained by the genotypes

The heritability h_g^2 is estimated from LD score regression and is used as initial parameter for LDpred2 algorithm.

Fraction of causal markers (i.e., the fraction of markers with non-zero effects)

The distribution of effect size for variant j is given as

$$\beta_j \sim \begin{cases} \mathcal{N}\left(0, \frac{h_g^2}{Mp}\right) & \text{with probability p} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

LDpred-inf (infinitesimal model)

In this case, all markers are **causal** (p=1), and effect drawn from a Gaussian distribution, i.e., $\beta_{ij} \sim_{iid} N\left(0, \left(h_g^2/M\right)\right)$. The posterior mean can be derived analytically

$$E(\beta_j \mid \tilde{\beta}_j, D) \approx \left(\frac{M}{Nh_g^2} I + D\right)^{-1} \tilde{\beta}_j \tag{3}$$

where $\tilde{\beta}_j$ denotes a vector of marginally estimated least-squares estimates obtained from the GWAS summary statistics. D denotes the LD matrix between the markers in the training data.

LDpred-grid/auto (non-infinitesimal model)

Without considering LD, the posterior mean of effect size can be derived as

$$\mathrm{E}\left(\beta_j \mid \tilde{\beta}_j\right) = \left(\frac{h_g^2}{h_g^2 + \frac{Mp}{N}}\right) \bar{p}_j \tilde{\beta}_j \tag{4}$$

where $\bar{p}_j$ is the posterior probability that the j^{th} marker is causal.

However, it is very difficult to derive a analytical expression for the posterior mean under a non-infinitesimal Gaussian mixture prior. Therefore, LDpred approximates it numerically by using an approximate MCMC Gibbs sampler. Once posterior mean effect sizes are estimated, they will be applied to genotype data to obtain **PRSs**.

Grid model tries a grid of parameters that p ranges from 0 to 1 and three h^2 which are 0.7/1/1.4 times of initial h^2 estimated by LD score regression. The best combination of p and h^2 will be selected according to the t score associated with each variants and covariates in phenotype model. Auto model runs the algorithm for 30 different p values ranging from 10e-4 to 0.9. h^2 estimated from LD score regression is the initial value of algorithm.

LDpred2

LDpred2 is LDpred 2.0. It can estimate effect size without using validation data to tunning hyper-parameters. Plus, it provides better predictive performance when the causal varients in long-range LD regions and sparse.

LDpred2 algorithm relies on an assumption that

$$\mathrm{sd}(G_j) \approx \frac{\mathrm{sd}(Y)}{\mathrm{se}\left(\hat{\gamma}_j\right)\sqrt{n}} \tag{5}$$

where G_j the genotype vector for variant j, and $\hat{\gamma}_j$ is marginal effect of vairant j. For binary traits with logistic model, the approximation is

$$\mathrm{sd}(G_j) \approx \frac{2}{\mathrm{se}\left(\hat{\gamma}_j\right)\sqrt{n_{\text{eff}}}} \tag{6}$$

where

$$n_{\text{eff}} = \frac{4}{1/n_{\text{case}} + 1/n_{\text{control}}} \tag{7}$$

To ensure the validity of the assumption, quality control on summary statistics is highly recommanded: remove variants with $SD_{ss} < 0.5 \times SD_{val}$ or $SD_{ss} > 0.1 + SD_{val}$ or $SD_{ss} < 0.1$ or $SD_{val} < 0.05$. SD_{ss} is the standard deviations derived from the summary statistics (right-hand side of equation). SD_{val} is the standard deviations of genotypes of individuals in the validation set (training set) (left-hand side).

Other Methods for PRS Prediction

An intuitive and simple method for prediction is **unadjusted PRS**. Given linear model (1), the standard unadjusted polygenic ris score for i^{th} individual is $S_i = \sum_{j=1}^{M} X_{ij}\hat{\beta}_j$ under the assumption that X_j are uncorrelated.

Pruning/Clumping and thresholding (P $+$ T, C $+$ T) is a commonly used approach to preidct PRS. Variants are filterd based on an empirically determined P-value threshold. Then linked variants will be clumped into the same group. Within each group, calculate correlation among index variants and nearby variants within certain genetic distance and remove correlated nearby variants beyond a certain value. Finally, only SNPs with lowest P values in each group are selected into the prediction model.

Method for Phenotype Prediction

Target data is splited into train dataset (80%) and test dataset (20%). Fit linear/logistic model on train dataset and then predict the phenotype on testdata. MSE and R^2 are calculated to considered as potential metrics to evaluate model performance of prediction. Missing genotypes are imputed with the mean of the genotype dosage for that variant with `snp_fastImputeSimple()` according to Florian Privé.

Required software

Required R packages include `data.table`, `tidyverse`, `bigsnpr`, `bigsparser` and plot libraries.

```
for (pkg in c("data.table", "tidyverse", "bigsnpr", "bigsparser", "ggplot2",
"gplots"))
    if (!(pkg %in% rownames(installed.packages()))) install.packages(pkg,
repos = 'http://cran.rstudio.com')
```

and PLINK version 1.9. To install on a desktop with `sudo` privilege, for example,

```
wget http://s3.amazonaws.com/plink1-assets/plink_linux_x86_64_20201019.zip &&
\
    unzip plink_linux_x86_64_20201019.zip && \
    sudo mv plink prettify /usr/local/bin
```

Input

1. Reference panel

   `--reference_geno`

2. Target data: genotypes and phenotypes

   ```
   --target_geno
   ```

   File contians only covariate predictors:

   ```
   --covFile
   ```

   File contrians only traits one column (phenotype):

   ```
   --phenoFile
   ```

3. Summary statistics of base data

   ```
   --ss
   ```

   Summary statistics should includes columns: "chr", "pos", "rsid", "a1", "a0", "beta", "beta_se"

Output

The pipeline saves the results from every steps. Main outputs are

- Posterior effect size and PRS (inf/grid/auto)
- Regression model results for phenotype prediction
- Summary of models and evaluation of prediction performance
- Plots
  - Quality control plot
  - Convergence plot from grid (z socre) and auto model (heritability and proportions of causal variants)

General workflow

Preliminary quality control on reference panel

Filtered SNPs that had a missing rate less than 1% and a minor allele frequency (MAF) greater than 1% in the reference genotype data. Excludes individuals whoes genotype missingness rate higher than 2%.

Output

- Reference panel after QC saved to `work_dir/ref.work_dir.bed/bim/fam`

```
sos run ldpred.ipynb preprocess:1 \
    --cwd $work_dir \
    --genoFiles <path/to/ref.bed>
```

Intersect SNPs among summary stats, reference panel and target data

This step returns a file `xxx.intersect.snplist` recording the common SNPs among reference panel, base data and target data. Then filter variants in the reference panel, summary statistics and target data. Common variants in target data should be filtered after SNP matching or quality control.

Output

- Summary statistics data after filtering common SNPs saved to `work_dir/sumstats.intersect.rds`
- Reference panel after filtering common SNPs saved to `work_dir/ref.work_dir.snp_intersect.extracted.bed/bim/fam`
- Target data after filtering common SNPs saved to `work_dir/target.work_dir.snp_intersect.extracted.bed/bim/fam`
- Common SNPs saved to three data folders saved to `work_dir/intersect.snplist`

```
sos run ldpred.ipynb snp_intersect \
    --cwd $work_dir \
    --ss <path/to/sumstats.rds> \
    --genoFiles <path/to/ref.work_dir.bed> <path/to/target.bed>

cat $work_dir/gwas_hdl.intersect.stdout
```

SNP Matching

Perform SNP matching using `snp_match(sumstats, map)`. Match alleles between summary statistics `sumstats` and SNP information in `map` from reference panel.

Output

- Summary statistics after snp matching saved to `work_dir/sumstats.snp_matched.rds`.
- The SNPs left after matching and also the SNPs used to fit LDpred model saved to `work_dir/snp_matched.snplist`.

```
sos run ldpred.ipynb snp_match \
    --cwd $work_dir \
    --reference_geno <work_dir/ref.work_dir.snp_intersect.extracted.rds>
\
    --ss <path/to/sumstats.rds>
```

Quality Control on Summary Statistics

Output

- Summary statistics after quality control saved to `work_dir/sumstats.snp_matched.qc.rds`.
- Plot showing the removed SNPs saved to `work_dir/sumstats.snp_matched.qc.png`.

- The SNPs used to fit LDpred model, also the SNPs we need to predict PRS in target data saved to `work_dir/sumstats.snp_matched.qc.snplist` .

  ```
  sos run ldpred.ipynb sumstats_qc \
      --cwd $work_dir \
      --reference_geno <work_dir/ref.work_dir.snp_intersect.extracted.rds>
  \
      --ss <path/to/sumstats.snp_matched.rds> \
      --sdy 1
  ```

Calculate LD matrix and Fit LDSC Model

Calculate LD correlation using `snp_cor(Gna, size, infos.pos)`

- size: for one SNP, window size around this SNP to compute correlations. Window size of 3 cM is applied in this pipeline which is recommanded by the developer.
- infos.pos: specifying the physical position on a chromosome (in base pairs) of each SNP.

Fit LDSC model using `snp_ldsc()`

Output

- LD matrix and LDSC model saved to `work_dir/sumstats.snp_matched.ld.rds` or `work_dir/sumstats.snp_matched.qc.ld.rds` .

  ```
  sos run ldpred.ipynb ldsc \
      --cwd $work_dir \
      --ss <work_dir/sumstats.snp_matched.rds> \
      --reference-geno <work_dir/ref.work_dir.snp_intersect.extracted.rds>
  ```

Estimate posterior effect sizes and PRS

Three models can be applied to predict PRS which are infinitesimal, grid and auto models.

- Estimate effect size
  - Infinitesimal model: `snp_ldpred2_inf(corr, df_beta, h2)`
  - Grid model: `snp_ldpred2_grid(corr, df_beta, grid_param)`

    gird_param: grid of hyper parameters p and h^2

  - Auto model: `snp_ldpred2_auto(corr, df_beta, h2_init, vec_p_init)`

    h2_init: estiamted from LD score regression

    vec_p_init: with 30 different initial values for p. Ranges from 0 to 0.9.

- Derive PRS

  For grid and auto model, the best combination of p and h^2 was selected based on largest t score and mean of 3 times of median absolute deviation of predicted PRS.

Output

- Estimated effect size `adj_beta` and PRS prediction `prs_pred` saved to `work_dir/sumstats.snp_matched.<model>_prs.rds`.

For Inf and Auto model

```
sos run ldpred.ipynb <model>_prs \
    --cwd $work_dir \
    --ss <work_dir/sumstats.snp_matched.qc.rds> \
    --target-geno <work_dir/target.snp_intersect.extracted.rds> \
    --ldsc <work_dir/sumstats.snp_matched.qc.ld.rds>
```

For Grid model

```
sos run ldpred.ipynb grid_prs \
    --cwd $work_dir \
    --ss <work_dir/sumstats.snp_matched.qc.rds> \
    --target-geno <work_dir/target.snp_intersect.extracted.rds> \
    --ldsc <work_dir/sumstats.snp_matched.qc.ld.rds> \
    --phenoFile <path/to/phenofile> \
    --covFile <path/to/covariatefile> \
    --response <continuous/binary>
```

Predict phenotype

Predict phenotypes in target data and evaluate the performance of prediction using MSE and R^2.

Output

- Linear/logistic model results saved to `work_dir/pheno.baseline.rds` or `work_dir/pheno.sumstats.snp_matched.<model>.rds`.
- Goodness of fit R^2, MSE and residual plot saved to `work_dir/pheno.baseline.pdf` or `work_dir/pheno.sumstats.snp_matched.<model>.pdf`.

Baseline model: Traits ~ Sex + Age

```
sos run ldpred.ipynb pred_eval \
    --cwd $work_dir \
    --phenoFile <path/to/phenofile> \
    --covFile <path/to/covariatefile> \
    --response <continuous/binary>
```

Ldpred model: Traits ~ Sex + Age + PRS

```
sos run ldpred.ipynb pred_eval \
    --cwd $work_dir \
    --prs <work_dir/sumstats.snp_matched.model_prs.rds> \
    --phenoFile <path/to/phenofile> \
    --covFile <path/to/covariatefile> \
    --response <continuous/binary>
```

Example analysis

See notebook ldpred2_example.ipynb for demonstration of results of a simulated data-set using UK biobank variants and 1000 genomes reference panel.

Command Interface

In [ ]:

```
sos run ldpred.ipynb -h
```

Global Parameter Setting

In [ ]:

```
[global]
# the output directory for generated files
parameter: cwd = path
# A string to identify your analysis run
parameter: name = f"{cwd:b}"
# For cluster jobs, number commands to run per job
parameter: job_size = 1
# Wall clock time expected
parameter: walltime = "5h"
# Memory expected
parameter: mem = "16G"
# Number of threads
parameter: numThreads = 20
# use this function to edit memory string for PLINK input
from sos.utils import expand_size
cwd = path(f"{cwd:a}")
```

Workflow

Preliminary quality control and preprocessing for genotype data

In [ ]:

```
# Filter SNPs and select individuals
[preprocess_1, snp_qc (basic QC filters)]
# PLINK binary files
parameter: genoFiles = paths
# The path to the file that contains the list of samples to remove (format FID,
parameter: remove_samples = path('.')
# The path to the file that contains the list of samples to keep (format FID, I
parameter: keep_samples = path('.')
# The path to the file that contains the list of variants to keep
parameter: keep_variants = path('.')
# minimum MAF filter to use. Notice that PLINK default is 0.01
parameter: maf_filter = 0.01
# maximum MAF filter to use
parameter: maf_max_filter = 0.0
# Maximum missingess per-variant
parameter: geno_filter = 0.01
# Maximum missingness per-sample
parameter: mind_filter = 0.02
# HWE filter
parameter: hwe_filter = 5e-08
```

```
fail_if(not (keep_samples.is_file() or keep_samples == path('.')), msg = f'Canno
fail_if(not (keep_variants.is_file() or keep_variants == path('.')), msg = f'Car
fail_if(not (remove_samples.is_file() or remove_samples == path('.')), msg = f'(

input: genoFiles, group_by=1
output: f'{cwd}/{_input:bn}.{name}{".extracted" if keep_variants.is_file() else
task: trunk_workers = 1, walltime = walltime, mem = mem, cores = numThreads, tag
bash: expand= "${ }", stderr = f'{_output:n}.stderr', stdout = f'{_output:n}.std
    plink \
      --bfile ${_input:n} \
      ${('--maf %s' % maf_filter) if maf_filter > 0 else ''} \
      ${('--max-maf %s' % maf_max_filter) if maf_max_filter > 0 else ''} \
      ${('--geno %s' % geno_filter) if geno_filter >= 0 else ''} \
      ${('--hwe %s' % hwe_filter) if hwe_filter >= 0 else ''} \
      ${('--mind %s' % mind_filter) if mind_filter >= 0 else ''} \
      ${('--keep %s' % keep_samples) if keep_samples.is_file() else ""} \
      ${('--remove %s' % remove_samples) if remove_samples.is_file() else ""} \
      ${('--extract %s' % keep_variants) if keep_variants.is_file() else ""} \
      --make-bed \
      --out ${_output:n} \
      --threads ${numThreads} \
      --memory ${int(expand_size(mem) * 0.9)}
```

In [ ]:

```
[preprocess_2 (convert PLINK to bigsnpr format with missing data mean imputed)]
input: group_by = 1, concurrent = False
output: f"{cwd:a}/{_input:bn}.bk", f"{cwd:a}/{_input:bn}.rds"
task: trunk_workers = 1, walltime = walltime, mem = mem, cores = numThreads, tag
bash: expand = "${ }"
    rm -f ${_output}
R: expand= "${ }", stderr = f'{_output[0]:n}.stderr', stdout = f'{_output[0]:n}.
    library(bigsnpr)
    # generate .bk and .rds file for R code
    dat = snp_readBed(${_input:r}, backingfile=${_output[0]:nr})
    obj.bigSNP <- snp_attach(dat)
    # get the CM information from 1000 Genome
    # will download the 1000G file to the current directory (".")
    obj.bigSNP$map$genetic.dist <- snp_asGeneticPos(obj.bigSNP$map$chromosome, c
    obj.bigSNP$genotypes = snp_fastImputeSimple(obj.bigSNP$genotypes, method = '
    saveRDS(obj.bigSNP, file = "${_output[1]}")
```

Find common SNPs among summary statistics, reference panel and test genotypes.

1. Find common SNPs
2. Get subsets.

In [ ]:

```
# SNP intersect of summary stats and genotype data
[snp_intersect_1]
# PLINK binary files
parameter: genoFiles = paths
# summary stats file
parameter: ss = path
input: ss, [x.with_suffix(".bim") for x in genoFiles]
output: substats = f"{cwd:a}/{_input[0]:bn}.intersect.rds",
        snp = f"{cwd:a}/{_input[0]:bn}.intersect.snplist"
```

```
task: trunk_workers = 1, walltime = walltime, mem = mem, cores = numThreads, tag
R: expand= "${ }", stderr = f'{_output[0]:n}.stderr', stdout = f'{_output[0]:n}.
    suppressMessages(library(tidyverse))
    sumstats <- readRDS(${_input[0]:r})
    geno_snps = lapply(c(${paths(_input[1:]):r,}), function(x) read.table(x, str
    common_snp <- Reduce(intersect, c(list(sumstats$rsid), geno_snps))
    # filter snps in sumstat
    new_sumstats <- sumstats %>%
        filter(rsid %in% common_snp)
    print(paste("There are", length(common_snp), "shared SNPs."))
    saveRDS(new_sumstats, file = "${_output["substats"]}")
    write.table(common_snp, file = "${_output["snp"]}", sep = " ",
                row.names = FALSE, col.names = FALSE, quote=FALSE)

[snp_intersect_2]
# PLINK binary files
parameter: genoFiles = paths
output: [(f"{cwd}/{x:bn}.snp_intersect.extracted.bed", f"{cwd}/{x:bn}.snp_inters
sos_run("preprocess", genoFiles=genoFiles, keep_variants=_input['snp'],
                    maf_filter=0, geno_filter=1, mind_filter=1, hwe_filter=-9,
                    cwd=cwd, name="snp_intersect")
```

In [ ]:

```
[snp_subset]
parameter: genoObj = path
parameter: keep_variants = path
input: genoObj, keep_variants
output: f"{cwd}/{_input[0]:bn}.{name}.subset.rds"
R: expand= "${ }", stderr = f'{_output:n}.stderr', stdout = f'{_output:n}.stdout
    library(bigsnpr)
    dat <- readRDS("${_input[0]}")
    snps <- as.vector(unlist(data.table::fread("${_input[1]}",header=F)))
    snp_subset(dat,  ind.col = match(snps, dat$map$marker.ID), backingfile = ${_
```

Convert reference genotype to bigsnpr format and get the genetic distance cM information

Will take a while to download the genetic distance database.

Allele harmonizing

In [ ]:

```
[snp_match]
# summary stats file
parameter: ss = path
parameter: reference_geno = path
input: ss, reference_geno
output: match = f'{cwd:a}/{_input[0]:bn}.snp_matched.rds',
        snplist = f'{cwd:a}/{_input[0]:bn}.snp_matched.snplist'
task: trunk_workers = 1, walltime = walltime, mem = mem, cores = numThreads, tag
R: expand= "${ }", stderr = f'{_output[0]:n}.stderr', stdout = f'{_output[0]:n}.
    library(bigsnpr)
    sumstats <- readRDS("${_input[0]}")
    # now attach the genotype object
    obj.bigSNP <- snp_attach("${_input[1]}")
    # extract the SNP information from the genotype
    map <- obj.bigSNP$map[-(2:3)]
    names(map) <- c("chr", "pos", "a1", "a0")
    # perform SNP matching
    updated_ss <- snp_match(sumstats, map)
    write.table(updated_ss$rsid, file = "${_output["snplist"]}", sep = " ",
            row.names = FALSE, col.names = FALSE,quote=FALSE)
    saveRDS(updated_ss, file = "${_output["match"]}")
```

Summary statistics quality Control

In [ ]:

```
[sumstats_qc]
# standard deviation of y; set it to 2 for binary traits
parameter: sdy = float
# summary stats file, snp matched
parameter: ss = path
# reference data geno object previously generated
parameter: reference_geno = path
input: ss, reference_geno

output: qc_plot = f'{cwd}/{_input[0]:bn}.qc.png',
        snplist = f'{cwd}/{_input[0]:bn}.qc.snplist',
        qc_res = f'{cwd}/{_input[0]:bn}.qc.rds'

task: trunk_workers = 1, walltime = walltime, mem = mem, cores = numThreads, tag
R: expand= "${ }", stderr = f'{_output[0]:n}.stderr', stdout = f'{_output[0]:n}.
    library(bigsnpr)
    suppressMessages(library(tidyverse))
    attach(readRDS("${_input[1]}"))
    info_snp = readRDS(${_input[0]:r})
    NCORES = bigparallelr::nb_cores()
    NCORES = tryCatch({bigparallelr::assert_cores(NCORES); NCORES }, error = fun
    ind.val = 1:nrow(genotypes)
    sd <- sqrt(big_colstats(genotypes, ind.val, ncores = NCORES)$var)
    sd_val <- sd[info_snp$`_NUM_ID_`]

    sdy = ${sdy}
    sd_ss <- with(info_snp, sdy / sqrt(n_eff * beta_se^2))

    is_bad <- sd_ss < (0.5 * sd_val) |
            sd_ss > (sd_val + 0.1) |
            sd_ss < 0.1 |
            sd_val < 0.05

    p = qplot(sd_val, sd_ss, color = is_bad, alpha = I(0.5)) +
      theme_bigstatsr() +
      coord_equal() +
      scale_color_viridis_d(direction = -1) +
      geom_abline(linetype = 2, color = "red") +
      labs(x = "Standard deviations in the validation set",
           y = "Standard deviations derived from the summary statistics",
           color = "Removed?")
    png("${_output["qc_plot"]}")
    print(p)
    dev.off()

    n = nrow(info_snp)
    print(paste(length(which(is_bad=="TRUE")), "over", n, "were removed in summa

    info_snp = info_snp[!is_bad, ]
    write.table(info_snp$rsid, file = ${_output["snplist"]:r}, sep = " ",
            row.names = FALSE, col.names = FALSE,quote=FALSE)
    saveRDS(info_snp, file=${_output["qc_res"]:r})
```

Calculate LD matrix and perform LD score regression

In [ ]:

```
[ldsc]
# summary stats file
parameter: ss = path
parameter: reference_geno = path
input: ss, reference_geno
output: f'{cwd}/{_input[0]:bn}.ld.rds'
task: trunk_workers = 1, walltime = walltime, mem = mem, cores = numThreads, tag
R: expand= "${ }", stderr = f'{_output[0]:n}.stderr', stdout = f'{_output[0]:n}.
    library(bigsnpr)
    library(data.table)
    library(bigsparser)
    suppressMessages(library(tidyverse))
    info_snp = readRDS("${_input[0]}")
    attach(readRDS("${_input[1]}"))
    NCORES = bigparallelr::nb_cores()
    NCORES = tryCatch({bigparallelr::assert_cores(NCORES); NCORES }, error = fun
    # Initialize variables for storing the LD score and LD matrix
    corr = NULL
    ld = NULL
    # Open a temporary file
    tmp = tempfile(tmpdir = "${cwd}/ld-cache")
    on.exit(file.remove(paste0(tmp, ".sbk")), add = TRUE)

    for (chr in 1:22) {
      # Extract SNPs that are included in the chromosome
      ind.chr <- which(info_snp$chr == chr)
      ind.chr2 <- info_snp$`_NUM_ID_`[ind.chr]
      # Calculate the LD
      corr0 <- snp_cor(
        genotypes,
        ind.col = ind.chr2,
        ncores = NCORES,
        infos.pos = map$genetic.dist[ind.chr2],
        size = 3 / 1000
      )
      if (chr == 1) {
        ld <- Matrix::colSums(corr0^2)
        corr <- as_SFBM(corr0, tmp)
      } else {
        ld <- c(ld, Matrix::colSums(corr0^2))
        corr$add_columns(corr0, nrow(corr))
      }
    }

    ldsc <- snp_ldsc(ld,
                    length(ld),
                    chi2 = (info_snp$beta / info_snp$beta_se)^2,
                    sample_size = info_snp$n_eff,
                    blocks = NULL)
    saveRDS(list(ld=ld,corr=corr,ldsc=ldsc), file = "${_output}")
```

Get adjusted betas and PRS

```
In [ ]:
[prs_core]
# rds file for summary stats
parameter: ss = path
# rds file of target data generated from bed file
parameter: target_geno = path
# ldsc file
parameter: ldsc = path
# method: choose from inf, grid, and auto
parameter: method = str
# phenotype file, must have a header
parameter: phenoFile = path
# covariates file, must have a header
parameter: covFile = path
# either continuous or binary
parameter: response = str
parameter: suffix = "prs"
input: ss, ldsc, target_geno
output: f'{cwd}/{_input[0]:bn}.{suffix}.rds'
task: trunk_workers = 1, walltime = walltime, mem = mem, cores = numThreads, tag
R: expand= "${ }", stderr = f'{_output:n}.stderr', stdout = f'{_output:n}.stdout
    library(bigsnpr)
    library(data.table)
    suppressMessages(library(tidyverse))

    ## load correlation data ##
    ldsc = readRDS("${_input[1]}")
    ## load summary stats ##
    info_snp = readRDS("${_input[0]}")
    ## load test data ##
    obj.test <- snp_attach("${_input[2]}")
    ind.test = 1:nrow(obj.test$genotypes)
    map2 <- obj.test$map[-3]
    names(map2) <- c("chr", "rsid", "pos", "a0", "a1")
    info_snp_test = snp_match(info_snp[, -which(names(info_snp) %in% c("_NUM_ID_
                              join_by_pos = FALSE)
    rsid = intersect(info_snp_test$rsid,info_snp$rsid)
    index = which(info_snp$rsid %in% rsid)
    print(paste(length(index), "SNPs are used for PRS calculations"))
    NCORES = bigparallelr::nb_cores()
    NCORES = tryCatch({bigparallelr::assert_cores(NCORES); NCORES }, error = fun
    df_beta <- info_snp[,c("beta", "beta_se", "n_eff", "_NUM_ID_")]

    if ("${method}" == 'inf') {
        ## adjusted beta ##
        adj_beta <- snp_ldpred2_inf(ldsc$corr, df_beta, h2 = ldsc$ldsc['h2'])
        ## Predict PRS ##
        prs_pred <- big_prodVec(obj.test$genotypes, adj_beta[index],
        ind.row = ind.test, ind.col = info_snp_test$`_NUM_ID_`)
        saveRDS(list(beta = adj_beta, prs = prs_pred), file = "${_output}")
    } else if ("${method}" == 'grid') {
        response = "${response}"
        `%notin%` <- Negate(`%in%`)
        if (response %notin% c("continuous", "binary")) stop("--response variabl
        # Prepare data for grid model
        p_seq <- signif(seq_log(1e-4, 1, length.out = 10), 2)
        h2_seq <- round(ldsc$ldsc['h2'] * c(0.7, 1, 1.4), 4)
        grid.param <-
```

```
            expand.grid(p = p_seq,
                        h2 = h2_seq,
                        sparse = c(FALSE, TRUE))

        # Get adjusted beta from grid model
        gird_beta <- snp_ldpred2_grid(ldsc$corr, df_beta, grid.param, ncores = N
        gird_beta[is.na(gird_beta)] = 0

        # Prediction
        grid_pred <- big_prodMat(obj.test$genotypes, gird_beta[index,],
        ind.row = ind.test, ind.col = info_snp_test$`_NUM_ID_`)

        ## find best betas
        # load covariates data
        covariates = read.table("${covFile}", header = T)
        y = read.table("${phenoFile}", header = T)
        data = cbind(covariates, y)

        # split train (80%) and test data (20%)
        set.seed(2021)
        train.ind = sample(nrow(data), 0.8*nrow(data))
        test.ind = setdiff(rows_along(data), train.ind)

        # find best p and h2
        response = "${response}"
        reg.formula <- paste(colnames(covariates), collapse = '+') %>%
            paste0(colnames(y),"~PRS+", .) %>%
            as.formula
        if(response == "continuous"){
            grid.model = big_univLinReg(as_FBM(grid_pred[train.ind,]),
                              y[train.ind,1], covar = as.matrix(covariates[train.ind,]
        }
        if(response == "binary"){
            grid.model = big_univLogReg(as_FBM(grid_pred[train.ind,]),
                              y[train.ind,1], covar = as.matrix(covariates[train.ind,]
        }

        # find best betas according to z score
        grid.param$score = grid.model$score
        adj_beta <- grid.param %>%
          mutate(id = row_number()) %>%
          arrange(desc(abs(score))) %>%
          slice(1) %>%
          pull(id) %>%
          gird_beta[, .]
        prs_pred <- big_prodVec(obj.test$genotypes, adj_beta[index],
            ind.row = ind.test, ind.col = info_snp_test$`_NUM_ID_`)

        library(ggplot2)
        ggplot(grid.param, aes(x = p, y = score, color = as.factor(h2))) +
          theme_bigstatsr() +
          geom_point() +
          geom_line() +
          scale_x_log10(breaks = 10^(-5:0), minor_breaks = grid.param$p) +
          facet_wrap(~ sparse, labeller = label_both) +
          labs(y = "Z-Score", color = "h2") +
          theme(legend.position = "top", panel.spacing = unit(1, "lines"))
        ggsave("${_output:n}.png")
      saveRDS(list(beta = adj_beta, prs = prs_pred, grid.param = grid.param), fi
    } else if ("${method}" == 'auto') {
        # Get adjusted beta from the auto model
        multi_auto <- snp_ldpred2_auto(
            ldsc$corr,
            df_beta,
            h2_init = ldsc$ldsc['h2'],
            vec_p_init = seq_log(1e-4, 0.9, length.out = 30),
            ncores = NCORES
        )
        beta_auto <- sapply(multi_auto, function(auto)
            auto$beta_est)
        pred_auto <- big_prodMat(obj.test$genotypes, beta_auto[index,],
        ind.row = ind.test, ind.col = info_snp_test$`_NUM_ID_`)

        ## Find best beta (take average)
        sc <- apply(pred_auto, 2, sd)
        keep <- abs(sc - median(sc)) < 3 * mad(sc)
        adj_beta <- rowMeans(beta_auto[, keep])
        prs_pred <- rowMeans(pred_auto[, keep])

        auto <- multi_auto[[1]]
        plot_grid(
          qplot(y = auto$path_p_est) +
            theme_bigstatsr() +
            geom_hline(yintercept = auto$p_est, col = "blue") +
            scale_y_log10() +
            labs(y = "p"),
          qplot(y = auto$path_h2_est) +
            theme_bigstatsr() +
            geom_hline(yintercept = auto$h2_est, col = "blue") +
            labs(y = "h2"),
          ncol = 1, align = "hv"
        )
        ggsave("${_output:n}.png")
        saveRDS(list(beta = adj_beta, prs = prs_pred), file = "${_output}")
    } else {
      stop("Wrong --method specified")
    }
```

Infinitesimal model

In [ ]:

```
[inf_prs]
# rds file for summary stats
parameter: ss = path
# rds file of target data generated from bed file
parameter: target_geno = path
# ldsc file
parameter: ldsc = path
input: ss, target_geno, ldsc
output: f'{cwd}/{_input[0]:bn}.inf_prs.rds'
sos_run("prs_core", ss=ss,target_geno=target_geno, ldsc=ldsc, method="inf", pher
```

Grid model

In [ ]:

```
[grid_prs]
# rds file for summary stats
parameter: ss = path
# rds file of target data generated from bed file
parameter: target_geno = path
```

```
# ldsc file
parameter: ldsc = path
# phenotype file, must have a header
parameter: phenoFile = path
# covariates file, must have a header
parameter: covFile = path
# either continuous or binary
parameter: response = str
input: ss, target_geno, ldsc
output: f'{cwd}/{_input[0]:bn}.grid_prs.rds'
sos_run("prs_core", ss=ss,target_geno=target_geno, ldsc=ldsc, method="grid", phe
```

Auto model

In [ ]:

```
[auto_prs]
# rds file for summary stats
parameter: ss = path
# rds file of target data generated from bed file
parameter: target_geno = path
# ldsc file
parameter: ldsc = path
input: ss, target_geno, ldsc
output: f'{cwd}/{_input[0]:bn}.auto_prs.rds'
sos_run("prs_core", ss=ss,target_geno=target_geno, ldsc=ldsc,method="auto", pher
```

Predict phenotype

Note: currently the prediction evaluation is based on one random split of train and test data. A possible improvement should be performing K-fold cross validation and summarize average results

In [ ]:

```
[pred_eval]
# rds file for PRS
parameter: prs = path(".")
# phenotype file, must have a header
parameter: phenoFile = path
# covariates file, must have a header
parameter: covFile = path(".")
# either continuous or binary
parameter: response = str

fail_if(not (covFile.is_file() or covFile == path('.')), msg = f'Cannot find ``{
fail_if(not (prs.is_file() or prs == path('.')), msg = f'Cannot find ``{prs}``')
fail_if(not (prs.is_file() or covFile.is_file()), msg = 'At least one of ``--prs

if not prs.is_file():
    # fake a PRS file name to get proper filename for later
    prs = path("baseline.out")
input: phenoFile
output: f'{cwd}/{_input:bn}.{prs:bn}.rds'
task: trunk_workers = 1, walltime = walltime, mem = mem, cores = numThreads, tag
R: expand= "${ }", stderr = f'{_output:n}.stderr', stdout = f'{_output:n}.stdout
    options(digits=7)
    library(bigsnpr)
    suppressMessages(library(gplots))
    suppressMessages(library(tidyverse))
    dat = read.table("${_input}", header = T)
    model = paste0(colnames(dat),"~")
    if (${"T" if covFile.is_file() else "F"}) {
        covariates = read.table("${covFile}", header = T)
        dat = cbind(dat,covariates)
        model = paste(model, paste(colnames(covariates), collapse = '+'))
    }
    if (${"T" if prs.is_file() else "F"}) {
      dat$PRS = readRDS("${prs}")$prs
      model = paste(model, "+PRS")
    }
    # split train (80%) and test data (20%)
    set.seed(2021)
    train.ind = sample(nrow(dat), 0.8*nrow(dat))
    test.ind = setdiff(rows_along(dat), train.ind)
    # fit model
    response = "${response}"
    if(response=="continuous"){
      fitted = model %>%
        as.formula %>%
        lm(.,data = dat[train.ind,])
    } else if(response=="binary"){
      fitted = model %>%
        as.formula %>%
        glm(.,data = dat[train.ind,], family = binomial)
    } else {
      stop("response parameter should be continuous or binary")
    }

    # Predict
    pheno_pred = predict(fitted, dat[test.ind,])
    residual = pheno_pred-dat[test.ind,1]
    tbl = tibble(model = c("model${prs:bnx}"),
                   R2 = round(summary(fitted)$adj.r.squared,5),
                   MSE = round(mean(residual^2),5))
    # save output
    pdf(file = "${_output:n}.pdf",width=10, height=10,)
    textplot(print(tbl))
    title("Goodness of fit and MSE")
    hist(residual,prob = T)
    plot(residual, xlab = "individuals", main = "Residual Plot")
    abline(h=0, lty = 2)
    dev.off()

    saveRDS(list(fitted = fitted, summary = tbl, predicted = pheno_pred, residua
```