
Benchmarking Polygenic Risk Scores vs. Machine Learning

Release 1.0

Muhammad Muneeb and Samuel F. Feng

Jan 12, 2022

CONTENTS

| | | |
|----------|---|----------|
| 1 | Directory structure and description of files generated in the whole process. | 3 |
| 2 | Content | 5 |
| 2.1 | Step 0 - Generate Data | 5 |
| 2.2 | Step 1 - Divide Data into Base and Target sets | 9 |
| 2.3 | Step 2 - CalculatePRS | 14 |
| 2.4 | Step 2.0 - QCTarget.R | 18 |
| 2.5 | Step 3 - Pvaluethreshold | 25 |
| 2.6 | Step 4 - MachineLearning | 26 |
| 2.7 | Step 5 - GetResults | 30 |

Sample files, code, the dataset used for this study are available on [Google Drive](#).

DIRECTORY STRUCTURE AND DESCRIPTION OF FILES GENERATED IN THE WHOLE PROCESS.

```
# "finalized" is the main directory, which contains all the code required for the
↳ execution.
0. finalized
0.1 finalized/ (snptest, plink, gtool, PRSice, PRSice.R, lib) - These all are the tools
↳ or files used by those tools.
0.2 finalized/dividedata.py - Contains code for Step - 1
0.3 finalized/ (QCtarget.R, CalculatePRS.py) - Contains code for Step - 2 and 2.0
0.4 finalized/pvalue.py - Contains code for Step - 3
0.5 finalized/pmodel.py - Contains code for Step - 4
0.6 finalized/getresults.py - Contains code for Step - 5
```

"finalized/1" means iteration 1.

```
# Test directory
1. finalized/1/test
"1" here represents the iteration number.

1.1 finalized/1/test/YRI.covariate (Covariate files)
1.2 finalized/1/test/phenotype.csv (Contains phenotype for the test set)
1.3 finalized/1/test/YRI.pheno (Contains phenotype for the test set in a particular
↳ format)
1.4 finalized/1/test/YRIs.sample (Contains phenotype for the test set)
1.5 finalized/1/test/ (test.ped, test.map, test.bed, test.bim, test.fam, test.vcf)
1.6 finalized/1/test/test_control_id.txt (Contains control IDs)
1.7 finalized/1/test/test_case_id.txt (Contains cases IDs)
1.8 finalized/test/test_id.txt (Contains test samples IDs)
```

```
#Train directory
2. finalized/1/train
"1" here represents the iteration number.

2.1 finalized/1/train/YRI.covariate (Covariate files)
2.2 finalized/1/train/phenotype.csv (Contains phenotype for the train set)
2.3 finalized/1/train/YRI.pheno (Contains phenotype for the train set in a particular
↳ format)
2.4 finalized/1/train/YRIs.sample (Contains phenotype for the train set)
2.5 finalized/1/train/ (train.ped, train.map, train.bed, train.bim, train.fam, train.vcf,
↳ trains.gen.gz, trains.gen)
```

(continues on next page)

(continued from previous page)

```
2.6 finalized/1/train/train_control_id.txt (Contains control IDs)
2.7 finalized/1/train/train_case_id.txt (Contains cases IDs)
2.8 finalized/train/train_id.txt (Contains train samples IDs)
2.9 finalized/train/train.assoc.fisher (GWAS generated by Plink)
2.10 finalized/train/train.sum (GWAS generated by Snptest)
2.11 finalized/train/Data.txt (Combined GWAS containing necessary columns)
```

```
#files directory
3. finalized/1/files
```

This directory contains the files generated when running this [Tutorial](#). We skipped this part because it is already explained there.

```
#result directory
4. finalized/1/result
4.1 finalized/1/result/PLINK.bar.png (Plink result)
4.2 finalized/1/result/PRScice_PRS.best (PRScice best PRS score)
4.3 finalized/1/result/test.txt (Lassosum result)
4.4 finalized/1/result/output.png (PRS distribution for machine learning and other
↳tools)
```

```
#pvalues directories
5. finalized/1/pv_(p-value)
5.1 finalized/1/pv_(p-value)/ML_probability (Deep learning classifier probabilities)
5.2 finalized/1/pv_(p-value)/ptest.raw (Test data containing SNPs above the p-value
↳threshold)
5.3 finalized/1/pv_(p-value)/ptrain.raw (Training data containing SNPs above the p-
↳value threshold)
```


CONTENT

2.1 Step 0 - Generate Data

Generated dataset is available on [Google Drive](#).

2.1.1 Dataset

Extract generatedata/CEU_merge.tar.gz file to generatedata/CEU_merge and delete CEU_merge.tar.gz

It contains the following files.

1. Genotypes_snptest.gen/
2. Ysim_snptest.sample (files generated by PhenotypeSimulator)
3. X.map / 4. X.ped (Same data in Ped and Map file)
5. X.log / 6. X.nosex (Ignore these files)
7. X.vcf (Same data in VCF file format so, bcftools can easily process it)

2.1.2 Dataset generation process

Following files and directories are required to generate the dataset.

1. CEU_merge (Directory which contains the merged files)
2. generate.py (Python script to generate data from Hapgen2)
3. gtool/plink (Tools for processing)
4. merge.py (Code for merging in python)
5. merge.sh (Bash script to deploy on HPC)
6. people.txt (List of directories in which sub-datasets should be stored. Numbered as 1, 2, 3, 4, 5 etc.)
7. rawdata (Contains Hapgen2 files)
8. sample.R (Rscript to generate PhenotypeSimulator data)
9. sample.sh (Script to generate Hapgen2 and PhenotypeSimulator)
- 10 CEU_1, CEU_2, CEU_3, and CEU_4 (Files that contain PhenotypeSimulator data)

2.1.3 Dataset generation - sample.R

```
# Required package. Kindly download the dependent packages
require("PhenotypeSimulator")
args = commandArgs(trailingOnly=TRUE)
start_time <- Sys.time()

# Number of people for which data is generated
numberOfPeople = 1500
filem <-paste("CEU",strtoi(args[1]),sep="_")
filem <-paste(filem,"gen",sep=".")

#Path to rawdata which contains the Hapgen2 data.
filem <-paste("/l/proj/kuin0009/MuhammadMuneeb/mlvsprs/generatedata/rawdata/",filem,sep=
↪"/")

genotypefile <- system.file("extdata/genotypes/hapgen",
                           filem,
                           package = "PhenotypeSimulator")

genotypefile <- gsub("\\.gen","", filem)

# Ensure correct file is read.
print(genotypefile)

# Paramters to change the genetic variation.
genVar <- 0.6
noiseVar <- 1 - genVar
totalSNPeffect <- 0.01
h2s <- totalSNPeffect/genVar
phi <- 0.6
rho <- 0.1
delta <- 0.3
shared <- 0.8
independent <- 1 - shared

phenotype <- runSimulation(N = numberOfPeople, P = 1, genotypefile = genotypefile,
                          format = "oxgen", cNrSNP = 5, genVar = genVar, h2s = h2s,
                          phi = 0.6, delta = 0.3, distBetaGenetic = "unif",
↪mBetaGenetic = 0.5,
                          sdBetaGenetic = 1, NrFixedEffects = 4, NrConfounders = c(1, 2,
↪ 1, 2), pIndependentConfounders = c(0, 1, 1, 0.5),
                          distConfounders = c("bin", "cat_norm", "cat_unif", "norm"),
                          probConfounders = 0.2, catConfounders = c(3, 4), pcorr = 0.8,
                          verbose = FALSE, seed = 3000)

result <-paste("CEU",strtoi(args[1]),sep="_")

# Save output in a specific format.
out <- savePheno(phenotype, directory=getwd(),
                outstring=result,
                format=c("csv", "snptest"), verbose=FALSE)
```

2.1.4 Dataset generation - generate.py

```
import os
import sys
import time

# Directory in which files will be saved.
filename = str(sys.argv[1])

# Place 1000 Genome + Hapmap3 (CEU, Chromosome 21) data in R/library/PhenotypeSimulator/
↳extdata/genotypes/hapgen
ps = "/home/kunet.ae/ku500519/anaconda3/envs/R_test/lib/R/library/PhenotypeSimulator/
↳extdata/genotypes/hapgen"

# Destination folder
dest = "/l/proj/kuin0009/MuhammadMuneeb/mlvsprs/generatedata/rawdata"

# Read information on this link to generate data from Hapgen2 https://mathgen.stats.ox.
↳ac.uk/genetics_software/hapgen/hapgen2.html
os.system(ps+os.sep+"/hapgen2 -m "+ps+os.sep+"CEU.map -l "+ps+os.sep+"CEU.leg -h "+ps+os.
↳sep+"CEU.haps -o "+dest+os.sep+"CEU_"+str(filename)+" -dl 43351827 0 0 0 -n 1500 0")

# Rename the files to process them easily
os.rename(dest+os.sep+"CEU_"+str(filename)+".controls.gen",dest+os.sep+"CEU_
↳"+str(filename)+".gen")
os.rename(dest+os.sep+"CEU_"+str(filename)+".controls.sample",dest+os.sep+"CEU_
↳"+str(filename)+".sample")
```

2.1.5 Dataset generation - sample.sh

This file contains the code to deploy the data generation process on High-performance computing (HPC)

```
#!/bin/sh
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=52
#SBATCH --job-name=PSBenchmark
#SBATCH --time=24:00:00
#SBATCH --mem=300000MB
#SBATCH --partition=prod
#SBATCH --account=kuin0009
#SBATCH --output=PSBenchmark.%A_%a.out
#SBATCH --error=PSBenchmark.%A_%a.err
#SBATCH --array=1-5

module purge
module load miniconda/3

# Read the directory name from the file, "people.txt"
radius=$(head -n $SLURM_ARRAY_TASK_ID people.txt | tail -n 1)

# Generate data from HAPGEN2
```

(continues on next page)

(continued from previous page)

```
python generate.py $radius  
  
#PASS data to PhenotypeSimulator  
Rscript sample.R $radius
```

2.1.6 Dataset generation - people.txt

```
1  
2  
3  
4  
5
```

2.1.7 Dataset generation - sample.R

```
#!/bin/bash  
#SBATCH --nodes=1  
#SBATCH --ntasks-per-node=20  
#SBATCH --job-name=Merge-90000  
#SBATCH --time=48:00:00  
#SBATCH --mem=100000MB  
#SBATCH --partition=prod  
#SBATCH --account=kuin0009  
#SBATCH --output=merge.%A_%a.out  
#SBATCH --error=merge.%A_%a.err  
#SBATCH --array=1-1  
  
module purge  
module load miniconda/3  
  
# Merge files  
python merge.py  
  
# Process the final files and save them in VCF file format.  
./gtool -G --g ./CEU_merge/Genotypes_snptest.gen --s ./CEU_merge/Ysim_snptest.sample --  
→ped ./CEU_merge/X.ped --map ./CEU_merge/X.map --phenotype pheno  
./plink -file ./CEU_merge/X --recode vcf --out ./CEU_merge/X
```

2.1.8 Dataset generation - merge.py

```
import os
import pandas as pd

files = os.listdir("./")
os.mkdir("CEU_merge")
data = pd.DataFrame()
gendata = pd.DataFrame()
count=0
for direc in files:
    # Read directories on in this format, CEU_NUMBER like CEU_1
    if "CEU_merge" not in direc and "rawdata" not in direc and os.path.isdir(direc) and
    ↪ "CEU" in direc:
        df2 = pd.read_csv(direc+os.sep+"Ysim_snptest.sample", sep=" ")
        sampletop = df2.head(1)
        samplebottom = df2.tail(len(df2)-1)
        data = pd.concat([samplebottom,data],axis=0)
        gen = pd.read_csv(direc+os.sep+"Genotypes_snptest.gen", sep = " ",header=None)
        frontgen = gen[[0, 1, 2, 3 , 4]]
        gen = gen.drop(0, axis=1)
        gen = gen.drop(1,axis=1)
        gen = gen.drop(2,axis=1)
        gen = gen.drop(3,axis=1)
        gen = gen.drop(4,axis=1)
        gendata = pd.concat([gendata,gen],axis=1,ignore_index=True)
        print("Merged",str(count))
        count=count+1
        os.mkdir(str(count))
        gendata = pd.concat([frontgen,gendata],axis=1)
        gendata.to_csv("CEU_merge"+os.sep+"Genotypes_snptest.gen",index=False,header=False,
        ↪ sep= " ")
        data['ID_1'] = range(1,len(data)+1)
        data['ID_2'] = range(1,len(data)+1)

data = pd.concat([sampletop,data],axis=0)
data.to_csv("CEU_merge"+os.sep+"Ysim_snptest.sample",index=False, sep=" ")
```

2.2 Step 1 - Divide Data into Base and Target sets

When calculating PRS, the GWAS summary statistic file is the Base file, a training set in Machine learning, whereas the Target file is a test set in machine learning.

2.2.1 Code execution

python dividedata.py (DirectoryName in which files will be stored)
 For example: python dividedata.py 1
 This function extracts the data from the previous step, divides it into training and
 ↪ test sets, and calculates the GWAS.

2.2.2 Actual Code in dividedata.py

```
import pandas as pd
import os
from sklearn.model_selection import train_test_split
import sys

#def reformat():
#    pass

#def subsubsection(pheno,direc,name):
#    case = pheno.loc[pheno["phenotype"]==1]
#    control = pheno.loc[pheno["phenotype"]==0]
#    case.to_csv(direc+os.sep+name+"_case_id.txt", index=False, columns=['user_id'],
#    ↪ header=False)
#    control.to_csv(direc+os.sep+name+"_control_id.txt", index=False, columns=['user_id',
#    ↪ ],header=False)
#    pheno.to_csv(direc+os.sep+name+"_id.txt", index=False, columns=['user_id'],
#    ↪ header=False)

def saveandformatsamplefile(top,bottom,direc):
    ### Split test cases/controls and train cases/controls
    ###make a phenotype file
    #bottom['ID_2'] = 'sample_' + bottom['ID_2'].astype(str)
    #bottom['ID_1'] = 'sample_' + bottom['ID_1'].astype(str)

    phenotype = pd.DataFrame()
    phenotype["user_id"] = bottom["ID_1"].values
    phenotype["phenotype"] = bottom["pheno"].values
    phenotype.to_csv(direc+os.sep+"phenotype.csv",sep="\t",index=False)

    ###make covarite file
    cov = pd.DataFrame()
    cov["FID"] = bottom["ID_2"].values
    cov["IID"] = bottom["ID_1"].values
    cov["Sex"] = 1
    cov["cov1"] = bottom["sharedConfounder1_bin1"].values
    cov["cov2"] = bottom["independentConfounder2_cat_norm1"].values
    cov["cov3"] = bottom["independentConfounder2_cat_norm2"].values
    cov["cov4"] = bottom["independentConfounder3_cat_unif1"].values
    sampletop = top.copy()
```

(continues on next page)

(continued from previous page)

```

samplebottom = bottom.copy()

samplebottom["pheno"] = samplebottom["pheno"].apply(pd.to_numeric)
samplebottom.pheno[samplebottom['pheno']<0]=0
samplebottom.pheno[samplebottom['pheno']>0]=1
samplebottom["pheno"] = pd.to_numeric(samplebottom["pheno"],downcast='integer')
sample = pd.concat([sampletop, samplebottom], axis=0)
sample= sample.astype(str)

if "test" in direc:
    subsubsection(phentype,direc,"test")
    sample.to_csv(direc+os.sep+"test_snptest.sample",index=False,sep=" ")

if "train" in direc:
    subsubsection(phentype,direc,"train")
    sample.to_csv(direc+os.sep+"train_snptest.sample",index=False,sep=" ")

# Modify the cases/controls information because plink considers 1 as a control and 2_
↪as a case, whereas other tools consider 0 as control and 1 as a case.
sample.pheno[sample['pheno']=='1']='2'
sample.pheno[sample['pheno']=='0']='1'

data = sample[["ID_1","ID_2","missing","pheno"]]
if "test" in direc:
    data.to_csv(direc+os.sep+"test.sample",index=False,sep=" ")

if "train" in direc:
    data.to_csv(direc+os.sep+"train.sample",index=False,sep=" ")

samplebottom.pheno[samplebottom['pheno']==1]='2'
samplebottom.pheno[samplebottom['pheno']==0]='1'

cov["cov5"] = bottom["sharedConfounder4_norm1"].values
cov["cov6"] = bottom["independentConfounder4_norm1"].values
cov.to_csv(direc+os.sep+"YRI.covariate",index=False,sep="\t")
###PRS phenotype
phenotype = pd.DataFrame()
phenotype["FID"] = bottom["ID_2"].values
phenotype["IID"] = bottom["ID_1"].values
phenotype["phenotype"] = bottom["pheno"].values
phenotype.to_csv(direc+os.sep+"YRI.pheno",sep="\t",index=False)
###NewSample file
sample = pd.concat([top, bottom], axis=0)
sample = sample[["ID_1","ID_2", "missing","pheno"]]
sample.to_csv(direc+os.sep+"YRIs.sample",index=False,sep=" ")
return phenotype['FID'].values

```

(continues on next page)

(continued from previous page)

```
def splitsample(sample,direc):
    # Extract the first row because it does not contain the sample information.
    sampletop = sample.head(1)
    samplebottom = sample.tail(len(sample)-1)

    #Modify the sample ID's.
    samplebottom['ID_1'] = samplebottom['ID_1'].astype(str)+str("_") + samplebottom['ID_1']
    samplebottom['ID_2'] = samplebottom['ID_2'].astype(str)+str("_") + samplebottom['ID_2']
    #samplebottom['ID_1'] = samplebottom['ID_1'].astype(str)
    #samplebottom['ID_2'] = samplebottom['ID_2'].astype(str)

    samplebottom["pheno"] = samplebottom["pheno"].apply(pd.to_numeric)

    #PhenotypeSimulator generates continuous phenotype, which we converted to binary
    phenotype by thresholding on 0.
    samplebottom["pheno"].values[samplebottom["pheno"] < 0] = 0
    samplebottom["pheno"].values[samplebottom["pheno"] > 0] = 1
    samplebottom["pheno"] = pd.to_numeric(samplebottom["pheno"],downcast='integer')

    # Spit the samples. The default is 75 percent training and 25 percent test sets.
    x_train, x_test, y_train, y_test = train_test_split(samplebottom, samplebottom["pheno"]
    ].values)
    sampletop.iloc[0,9]="B"

    trainsample = saveandformatsamplefile(sampletop, x_train,direc+os.sep+"train")
    testsample = saveandformatsamplefile(sampletop, x_test,direc+os.sep+"test")
    return trainsample,testsample

def commit(direc,name):
    sample = pd.read_csv(direc+os.sep+name+".sample",sep=" ")
    samplebottom = sample.tail(len(sample)-1)
    fam = pd.read_csv(direc+os.sep+name+".fam",sep="\s+",header=None)
    fam[5] = samplebottom['pheno'].values
    fam.to_csv(direc+os.sep+name+".fam",header=False,index=False, sep=" ")

# Directory name in which files will be stored.
# Create four directories to contain train, test, and intermediate files.

direc = sys.argv[1]
if not os.path.isdir(direc):
    os.mkdir(direc)
if not os.path.isdir(direc+os.sep+"test"):
    os.mkdir(direc+os.sep+"test")
if not os.path.isdir(direc+os.sep+"train"):
    os.mkdir(direc+os.sep+"train")
if not os.path.isdir(direc+os.sep+"files"):
```

(continues on next page)

(continued from previous page)

```

os.mkdir(direc+os.sep+"files")

testdirec = direc+os.sep+"test"
traindirec = direc+os.sep+"train"
filesdirec = direc+os.sep+"files"

# Read the sample files, and ensure path is correct.
originalsamples = pd.read_csv("/l/proj/kuin0009/MuhammadMuneeb/mlvsprs/generatedata/CEU_
↳merge/Ysim_snptest.sample", sep=" ")

# This function splits the samples into training and test sets.
train,test = splitsample(originalsamples,direc)

# Extract test samples using bcftools
os.system("bcftools view -S ./"+testdirec+os.sep+"test_id.txt /l/proj/kuin0009/
↳MuhammadMuneeb/mlvsprs/generatedata/CEU_merge/X.vcf > ./"+testdirec+os.sep+"test.vcf")
os.system(" ./plink --vcf ./"+testdirec+os.sep+"test.vcf --make-bed --out ./
↳"+testdirec+os.sep+"test")
os.system("./plink --bfile ./"+testdirec+os.sep+"test --recode --tab --out ./
↳"+testdirec+os.sep+"test")

# Extract training samples using bcftools
os.system("bcftools view -S ./"+traindirec+os.sep+"train_id.txt /l/proj/kuin0009/
↳MuhammadMuneeb/mlvsprs/generatedata/CEU_merge/X.vcf > ./"+traindirec+os.sep+"train.vcf
↳")
os.system("./plink --vcf ./"+traindirec+os.sep+"train.vcf --make-bed --out ./
↳"+traindirec+os.sep+"train")

# Modify the fam file.
commit(testdirec,"test")
commit(traindirec,"train")

os.system("./plink --bfile ./"+traindirec+os.sep+"train --recode --tab --out ./
↳"+traindirec+os.sep+"train")

# Calculate GWAS using plink
os.system("./plink --bfile ./"+traindirec+os.sep+"train --allow-no-sex --fisher --out ./
↳"+traindirec+os.sep+"train")

# Calculate GWAS using Snptest
os.system("./gtool -P --ped ./"+traindirec+os.sep+"train.ped --map ./"+traindirec+os.sep+
↳"train.map --og ./"+traindirec+os.sep+"train.gen --os ./"+traindirec+os.sep+"train_
↳fake.sample")
os.system("bcftools convert ./"+traindirec+os.sep+"train.vcf -g ./"+traindirec+os.sep+
↳"trains")
os.system("./snptest -data ./"+traindirec+os.sep+"trains.gen.gz ./"+traindirec+os.sep+
↳"train_snptest.sample -o ./"+traindirec+os.sep+"train.sum -frequentist 1 -method score_
↳-pheno pheno")

```

(continues on next page)

(continued from previous page)

```
snpteststats = pd.read_csv(trainindirec+os.sep+"train.sum",sep=" ",low_memory=False,
    ↪skiprows = 10)
snpteststats = snpteststats.head(len(snpteststats)-1)
plinkstats = pd.read_csv(trainindirec+os.sep+"train.assoc.fisher",sep="\s+",low_
    ↪memory=False)
gwasstats = pd.DataFrame()

# Change the column names as required by lassosum and plink.
# Ensure chromosome number is correct. We changed it in the later step.

gwasstats['CHR'] = ['0']*len(plinkstats)
gwasstats['BP'] = plinkstats['BP'].values
gwasstats['SNP'] = plinkstats['SNP'].values
gwasstats['A1'] = snpteststats['alleleA'].values
gwasstats['A2'] = snpteststats['alleleB'].values
gwasstats['N'] = snpteststats['all_total'].values
gwasstats['SE'] = snpteststats['frequentist_add_se_1'].values
gwasstats['P'] = plinkstats['P'].values
gwasstats['OR'] = plinkstats['OR'].values

# In simulated data, the imputation score is 1 as we have not used any imputation_
    ↪technique.
gwasstats['INFO'] = 1
gwasstats['MAF'] = snpteststats['all_maf'].values
#gwasstats.to_csv(trainindirec+os.sep+"Data.txt.gz",index=False, sep="\t",compression='gzip
    ↪')
gwasstats.to_csv(trainindirec+os.sep+"Data.txt",index=False, sep="\t")
```

2.3 Step 2 - CalculatePRS

2.3.1 Code execution

```
python CalculatePRS.py (DirectoryName in which files will be stored)
For example: python CalculatePRS.py 1
```

2.3.2 Actual Code in CalculatePRS.py

```
import pandas as pd
import os
import glob
import numpy as np
import shutil
import sys
```

(continues on next page)

(continued from previous page)

```
import numpy as np
import scipy as sp
from scipy import stats
import matplotlib.pyplot as plt
import os
import pandas as pd
import numpy as np
from sklearn.metrics import roc_auc_score
import math
direc = sys.argv[1]

traindirec = direc+os.sep+"train"
testdirec =direc+os.sep+"test"
result = direc+os.sep+"result"
fdirec = direc+os.sep+"files"

if not os.path.isdir(direc+os.sep+"files"):
    os.mkdir(direc+os.sep+"files")
if not os.path.isdir(direc+os.sep+"result"):
    os.mkdir(direc+os.sep+"result")

def changeIDS(direct):
    data = pd.read_csv(direct, sep="\s+",index_col=False)
    data['FID'] = data['FID'].str.split('_').str[0]
    data['IID'] = data['IID'].str.split('_').str[0]
    data.to_csv(direct,sep="\t",index=False)

# Modify the bim file for the test data.
bimfile = pd.read_csv(testdirec+os.sep+"test.bim",header=None,sep="\s+")
bimfile[0] = 21
bimfile[2] = list(range(1,len(bimfile)+1))
bimfile.to_csv(testdirec+os.sep+"test.bim",header=False, index=False, sep="\t")

# Modify the GWAS file, and use the correct chromosome number for each SNP.
data = pd.read_csv(traindirec+os.sep+"Data.txt",sep="\s+")
data['CHR']=21
data.to_csv(traindirec+os.sep+"Data.txt.gz", index=False, sep="\t",compression='gzip')

changeIDS(testdirec+os.sep+"YRI.covariate")
changeIDS(testdirec+os.sep+"YRI.pheno")

changeIDS(traindirec+os.sep+"YRI.covariate")
changeIDS(traindirec+os.sep+"YRI.pheno")
```

The code segments afterward are taken from this [Tutorial](#). At this point, the dataset is ready such that the code provided in the tutorial can be applied to it. We just automated all the steps, and we strongly recommend looking at that tutorial for understanding.

```
os.system("gunzip -c ./"+traindirec+os.sep+"Data.txt.gz | awk 'NR==1 || ($11 > 0.01) && (
↪$10 > 0.8) {print}' | gzip > ./"+fdirec+os.sep+"Data.gz")
```

(continues on next page)

(continued from previous page)

```
os.system("gunzip -c ./"+fdirec+os.sep+"Data.gz | awk '{seen[$3]++; if(seen[$3]==1){
↳ print}}' | gzip -> ./"+fdirec+os.sep+"Data.nodup.gz")
os.system("gunzip -c ./"+fdirec+os.sep+"Data.nodup.gz | awk '!( ($4=="A\" && $5=="T\")
↳ || ($4=="T\" && $5=="A\") || ($4=="G\" && $5=="C\") || ($4=="C\" && $5=="G\"))
↳ {print}' | gzip > ./"+fdirec+os.sep+"Data.QC.gz")

###QC-TargetFile
os.system("./plink --bfile ./"+testdirec+os.sep+"test --maf 0.01 --hwe 1e-6 --geno 0.01 -
↳ -mind 0.01 --write-snpList --make-just-fam --allow-no-sex -out ./"+fdirec+os.sep+"test.
↳ QC")

###This will work for alot of variants.
os.system("./plink --bfile ./"+testdirec+os.sep+"test --keep ./"+fdirec+os.sep+"test.QC.
↳ fam --extract ./"+fdirec+os.sep+"test.QC.snpList --allow-no-sex --indep-pairwise 200
↳ 50 0.25 --out ./"+fdirec+os.sep+"test.QC")

os.system("./plink --bfile ./"+testdirec+os.sep+"test --extract ./"+fdirec+os.sep+"test.
↳ QC.prune.in --keep ./"+fdirec+os.sep+"test.QC.fam --het --out ./"+fdirec+os.sep+"test.
↳ QC")

os.system("Rscript QCtarget.R "+direc+" 1")
print("Rscript QCtarget.R 1")

os.system("./plink --bfile ./"+testdirec+os.sep+"test --extract ./"+testdirec+os.sep+
↳ "test.QC.prune.in --keep ./"+testdirec+os.sep+"test.valid.sample --out ./
↳ "+testdirec+os.sep+"test.QC")
#exit(0)

###If sex information is present then use this. Simulated data does not contain the sex
↳ information so, we ignored it.

'''
os.system("./plink --bfile ./"+testdirec+os.sep+"test --extract ./"+fdirec+os.sep+"test.
↳ QC.prune.in --keep ./"+fdirec+os.sep+"test.valid.sample --check-sex --out ./
↳ "+fdirec+os.sep+"test.QC")
os.system("Rscript QCtarget.R "+direc+" 2")
os.system("./plink --bfile ./"+testdirec+os.sep+"test --extract ./"+fdirec+os.sep+"test.
↳ QC.prune.in --keep ./"+fdirec+os.sep+"test.QC.valid --rel-cutoff 0.125 --out ./
↳ "+fdirec+os.sep+"test.QC")
os.system("./plink --bfile ./"+testdirec+os.sep+"test --make-bed --allow-no-sex --allow-
↳ no-vars --keep ./"+fdirec+os.sep+"test.QC.rel.id --out ./"+fdirec+os.sep+"test.QC --
↳ extract ./"+fdirec+os.sep+"test.QC.snpList --exclude ./"+fdirec+os.sep+"test.mismatch -
↳ a1-allele ./"+fdirec+os.sep+"EUR.a1")
'''
##Mismatch information
#When information is correct.
#os.system("./plink --bfile ./"+testdirec+os.sep+"test --make-bed --allow-no-sex --out .
↳ ./"+fdirec+os.sep+"test.QC --extract ./"+fdirec+os.sep+"test.QC.snpList --exclude ./
↳ "+fdirec+os.sep+"test.mismatch --a1-allele ./"+fdirec+os.sep+"test.a1")
```

(continues on next page)

(continued from previous page)

```
os.system("./plink --bfile ./"+testdirec+os.sep+"test --make-bed --allow-no-sex --out ./
↳"+fdirec+os.sep+"test.QC --extract ./"+fdirec+os.sep+"test.QC.snplist --a1-allele ./
↳"+fdirec+os.sep+"test.a1")

###Make final files

os.system("Rscript QCtarget.R "+direc+" 3")
os.system("./plink --bfile ./"+testdirec+os.sep+"test --clump-p1 1 --clump-r2 0.1 --
↳clump-kb 250 --clump ./"+fdirec+os.sep+"Data.QC.Transformed --clump-snp-field SNP --
↳clump-field P --out ./"+fdirec+os.sep+"test")

os.system("awk 'NR!=1{print $3}' ./"+fdirec+os.sep+"test.clumped > ./"+fdirec+os.sep+
↳"test.valid.snp")
os.system("awk '{print $3,$8}' ./"+fdirec+os.sep+"Data.QC.Transformed > ./"+fdirec+os.
↳sep+"SNP.pvalue")

os.system("echo \"0.001 0 0.001\" > ./"+fdirec+os.sep+"range_list")
os.system("echo \"0.05 0 0.05\" >> ./"+fdirec+os.sep+"range_list")
os.system("echo \"0.1 0 0.1\" >> ./"+fdirec+os.sep+"range_list")
os.system("echo \"0.2 0 0.2\" >> ./"+fdirec+os.sep+"range_list")
os.system("echo \"0.3 0 0.3\" >> ./"+fdirec+os.sep+"range_list")
os.system("echo \"0.4 0 0.4\" >> ./"+fdirec+os.sep+"range_list")
os.system("echo \"0.5 0 0.5\" >> ./"+fdirec+os.sep+"range_list")

os.system("./plink --bfile ./"+testdirec+os.sep+"test --score ./"+fdirec+os.sep+"Data.QC.
↳Transformed 3 4 9 header --q-score-range ./"+fdirec+os.sep+"range_list ./"+fdirec+os.
↳sep+"SNP.pvalue --extract ./"+fdirec+os.sep+"test.valid.snp --out ./"+fdirec+os.sep+
↳"test")

os.system("./plink --bfile ./"+fdirec+os.sep+"test.QC --indep-pairwise 200 50 0.25 --out.
↳./"+fdirec+os.sep+"test")
os.system("./plink --bfile ./"+fdirec+os.sep+"test.QC --extract ./"+fdirec+os.sep+"test.
↳prune.in --pca 6 --out ./"+fdirec+os.sep+"test")

# Plink PRS
os.system("Rscript QCtarget.R "+direc+" 4")

#PRScice PRS
os.system("Rscript PRScice.R --prsice PRScice --base "+fdirec+os.sep+"Data.QC.gz --target
↳"+fdirec+os.sep+"test.QC --thread 1 --print-snp --stat OR --binary-target T --out
↳"+result+"/"+os.sep+"PRScice_PRS")

#Lassosum PRS
os.system("Rscript QCtarget.R "+direc+" 6")
```

2.4 Step 2.0 - QCtarget.R

2.4.1 Code execution

This is supplementary file and used by CalculatePRS.py to calculate the PRS values. The code segments in this section are taken from this `Tutorial <<https://choishingwan.github.io/PRS-Tutorial/base/>>`.

At this point, the dataset is ready such that the code provided in the tutorial can be applied to it.

We just automated all the steps, and we strongly recommend looking at that tutorial for understanding.

2.4.2 Actual Code in QCtarget.R

```
args <- commandArgs(trailingOnly = TRUE)
print(args)
if (args[2]=="1"){
  #args<-c("CEU_5_1_prsData")

  result <-paste(".",args[1],"files",toString("test.QC.het"),sep="/")
  dat <- read.table(result, header=T) # Read in the EUR.het file, specify it has header
  m <- mean(dat$F) # Calculate the mean
  s <- sd(dat$F) # Calculate the SD
  valid <- subset(dat, F <= m+3*s & F >= m-3*s) # Get any samples with F coefficient
  within 3 SD of the population mean
  result <-paste("./",args[1],"files",toString("test.valid.sample"),sep="//")
  write.table(valid[,c(1,2)], result, quote=F, row.names=F) # print FID and IID for
  valid samples
  result <-paste("./",args[1],"test",toString("test.bim"),sep="//")
  bim <- read.table(result)

  colnames(bim) <- c("CHR", "SNP", "CM", "BP", "B.A1", "B.A2")

  # Read in QCed SNPs
  result <-paste("./",args[1],"files",toString("test.QC.snplist"),sep="//")

  qc <- read.table(result, header = F, stringsAsFactors = F)
  # Read in the GWAS data

  result <-paste("./",args[1],"files",toString("Data.QC.gz"),sep="//")

  height <-read.table(gzfile(result),
    header = T,
    stringsAsFactors = F,
    sep="\t")
  # Change all alleles to upper case for easy comparison

  height$A1 <- toupper(height$A1)
  height$A2 <- toupper(height$A2)
  bim$B.A1 <- toupper(bim$B.A1)
```

(continues on next page)

(continued from previous page)

```

bim$B.A2 <- toupper(bim$B.A2)
info <- merge(bim, height, by = c("SNP", "CHR", "BP"))
# Filter QCed SNPs

info <- info[info$SNP %in% qc$V1,]

# Function for finding the complementary allele

complement <- function(x) {
  switch (
    x,
    "A" = "T",
    "C" = "G",
    "T" = "A",
    "G" = "C",
    return(NA)
  )
}

# Get SNPs that have the same alleles across base and target
info.match <- subset(info, A1 == B.A1 & A2 == B.A2)
# Identify SNPs that are complementary between base and target
info$C.A1 <- sapply(info$B.A1, complement)
info$C.A2 <- sapply(info$B.A2, complement)
info.complement <- subset(info, A1 == C.A1 & A2 == C.A2)
# Update the complementary alleles in the bim file
# This allow us to match the allele in subsequent analysis

complement.snps <- bim$SNP %in% info.complement$SNP
bim[complement.snps,]$B.A1 <-
  sapply(bim[complement.snps,]$B.A1, complement)
bim[complement.snps,]$B.A2 <-
  sapply(bim[complement.snps,]$B.A2, complement)

# identify SNPs that need recoding
info.recode <- subset(info, A1 == B.A2 & A2 == B.A1)
# Update the recode SNPs
recode.snps <- bim$SNP %in% info.recode$SNP
tmp <- bim[recode.snps,]$B.A1
bim[recode.snps,]$B.A1 <- bim[recode.snps,]$B.A2
bim[recode.snps,]$B.A2 <- tmp

# identify SNPs that need recoding & complement
info.crecode <- subset(info, A1 == C.A2 & A2 == C.A1)
# Update the recode + strand flip SNPs
com.snps <- bim$SNP %in% info.crecode$SNP
tmp <- bim[com.snps,]$B.A1
bim[com.snps,]$B.A1 <- as.character(sapply(bim[com.snps,]$B.A2, complement))
bim[com.snps,]$B.A2 <- as.character(sapply(tmp, complement))
result <- paste(".", args[1], "files", toString("test.a1"), sep="//")

# Output updated bim file

```

(continues on next page)

(continued from previous page)

```

write.table(
  bim[,c("SNP", "B.A1")],
  result,
  quote = F,
  row.names = F,
  col.names = F,
  sep="\t"
)
mismatch <-
  bim$SNP[!(bim$SNP %in% info.match$SNP |
            bim$SNP %in% info.complement$SNP |
            bim$SNP %in% info.recode$SNP |
            bim$SNP %in% info.crecode$SNP)]
result <-paste("./",args[1],"files",toString("test.mismatch"),sep="//")

write.table(
  mismatch,
  result,
  quote = F,
  row.names = F,
  col.names = F
)
}
if (args[2]=="2"){
  result <-paste("./",args[1],"files",toString("test.valid.sample"),sep="//")
  valid <- read.table(result, header=T)
  result <-paste("./",args[1],"files",toString("test.QC.sexcheck"),sep="//")
  dat <- read.table(result, header=T)
  valid <- subset(dat, STATUS=="OK" & FID %in% valid$FID)
  result <-paste("./",args[1],"files",toString("test.QC.valid"),sep="//")
  write.table(valid[,c("FID", "IID")], result, row.names=F, col.names=F, sep="\t",
  ↪quote=F)
}
if (args[2]=="3"){
  result <-paste("./",args[1],"files",toString("Data.QC.gz"),sep="//")

  dat <- read.table(gzfile(result), header=T)
  dat$BETA <- log(dat$OR)
  result <-paste("./",args[1],"files",toString("Data.QC.Transformed"),sep="//")

  write.table(dat, result, quote=F, row.names=F)
}
if (args[2]=="4"){

  result <-paste("./",args[1],"test",toString("YRI.pheno"),sep="//")

  p.threshold <- c(0.001,0.05,0.1,0.2,0.3,0.4,0.5)
  # Read in the phenotype file
  phenotypes <- read.table(result, sep="\t",header=T)
  # Read in the PCs

```

(continues on next page)

(continued from previous page)

```

result <-paste("./",args[1],"files",toString("test.eigenvec"),sep="//")

pcs <- read.table(result, header=F)
# The default output from plink does not include a header
# To make things simple, we will add the appropriate headers
# (1:6 because there are 6 PCs)

colnames(pcs) <- c("FID", "IID", paste0("PC",1:6))
# Read in the covariates (here, it is sex)

#pcs$FID <- as.character(pcs$FID)
#pcs$IID <- as.character(pcs$FID)

#pcs$FID <- paste(pcs$FID, pcs$FID,sep="_")
#pcs$IID <- paste(pcs$IID, pcs$IID,sep="_")

result <-paste("./",args[1],"test",toString("YRI.covariate"),sep="//")
covariate <- read.table(result, header=T)

#print(head(phenotypes))
#print(head(covariate))
#print(head(pcs))
# Now merge the files
pheno <- merge(merge(phenotypes, covariate, by=c("FID", "IID")), pcs, by=c("FID","IID
→"))

# We can then calculate the null model (model with PRS) using a linear regression
# (as height is quantitative)

null.model <- lm(phenotype~., data=pheno[,!colnames(pheno)%in%c("FID","IID")])

# And the R2 of the null model is
null.r2 <- summary(null.model)$r.squared

prs.result <- NULL
for(i in p.threshold){
  # Go through each p-value threshold
  result <-paste("./",args[1],"files",paste0("test.",i,".profile"),sep="//")
  prs <- read.table(result, header=T)
  #prs$FID <- as.character(prs$FID)
  #prs$IID <- as.character(prs$FID)

  #prs$FID <- paste(prs$FID, prs$FID,sep="_")
  #prs$IID <- paste(prs$IID, prs$IID,sep="_")
  # Merge the prs with the phenotype matrix
  # We only want the FID, IID and PRS from the PRS file, therefore we only select the
  # relevant columns
  print(head(prs))
  pheno.prs <- merge(pheno, prs[,c("FID","IID", "SCORE")], by=c("FID", "IID"))
  # Now perform a linear regression on Height with PRS and the covariates
  # ignoring the FID and IID from our model
  model <- lm(phenotype~., data=pheno.prs[,!colnames(pheno.prs)%in%c("FID","IID")])

```

(continues on next page)

(continued from previous page)

```

# model R2 is obtained as
model.r2 <- summary(model)$r.squared
# R2 of PRS is simply calculated as the model R2 minus the null R2
prs.r2 <- model.r2-null.r2
# We can also obtain the coefficient and p-value of association of PRS as follow
prs.coef <- summary(model)$coeff["SCORE",]
prs.beta <- as.numeric(prs.coef[1])
prs.se <- as.numeric(prs.coef[2])
prs.p <- as.numeric(prs.coef[4])
# We can then store the results
prs.result <- rbind(prs.result, data.frame(Threshold=i, R2=prs.r2, P=prs.p,
↪BETA=prs.beta, SE=prs.se))
}
# Best result is:
print(prs.result[which.max(prs.result$R2),])
#args<-c("CEU_5_1_prsData")

#result <-paste(strsplit(args[1], "_prsData"), "_results", sep="")
result <-paste("./", args[1], "result", toString("PLINK.bar.png"), sep="//")

png(result,
     height=10, width=10, res=300, unit="in")
# First, obtain the colorings based on the p-value
col <- suppressWarnings(colorRampPalette(c("dodgerblue", "firebrick"))))
# We want the color gradient to match the ranking of p-values
prs.result <- prs.result[order(-log10(prs.result$P)),]
prs.result$color <- col(nrow(prs.result))
prs.result <- prs.result[order(prs.result$Threshold),]
# generate a pretty format for p-value output
prs.result$print.p <- round(prs.result$P, digits = 3)
prs.result$print.p[!is.na(prs.result$print.p) & prs.result$print.p == 0 ] <-
  format(prs.result$P[!is.na(prs.result$print.p) & prs.result$print.p == 0 ], digits_
↪= 2)
prs.result$print.p <- sub("e", "%*10^", prs.result$print.p)
# Generate the axis labels
xlab <- expression(italic(P) ~ value ~ threshold ~ (italic(P)[T]))
ylab <- expression(paste("PRS model fit: ", R ^ 2))
# Setup the drawing area
layout(t(1:2), widths=c(8.8,1.2))
par( cex.lab=1.5, cex.axis=1.25, font.lab=2,
     oma=c(0,0.5,0,0),
     mar=c(4,6,0.5,0.5))
# Plotting the bars
b<- barplot(height=prs.result$R2,
            col=prs.result$color,
            border=NA,
            ylim=c(0, max(prs.result$R2)*1.25),
            axes = F, ann=F)
# Plot the axis labels and axis ticks
odd <- seq(0,nrow(prs.result)+1,2)
even <- seq(1,nrow(prs.result),2)
axis(side=1, at=b[odd], labels=prs.result$Threshold[odd], lwd=2)

```

(continues on next page)

(continued from previous page)

```

axis(side=1, at=b[even], labels=prs.result$Threshold[even],lwd=2)
axis(side=1, at=c(0,b[1],2*b[length(b)]-b[length(b)-1]), labels=c("", "", ""), lwd=2,
↪lwd.tick=0)
# Write the p-value on top of each bar
text( parse(text=paste(
  prs.result$print.p)),
  x = b+0.1,
  y = prs.result$R2+ (max(prs.result$R2)*1.05-max(prs.result$R2)),
  srt = 45)
# Now plot the axis lines
box(bty='L', lwd=2)
axis(2,las=2, lwd=2)
# Plot the axis titles
title(ylab=ylab, line=4, cex.lab=1.5, font=2 )
title(xlab=xlab, line=2.5, cex.lab=1.5, font=2 )
# Generate plot area for the legend
par(cex.lab=1.5, cex.axis=1.25, font.lab=2,
    mar=c(20,0,20,4))
prs.result <- prs.result[order(-log10(prs.result$P)),]
image(1, -log10(prs.result$P), t(seq_along(-log10(prs.result$P))), col=prs.result
↪$color, axes=F,ann=F)
axis(4,las=2,xaxs='r',yaxs='r', tck=0.2, col="white")
# plot legend title
title(bquote(atop(-log[10] ~ model, italic(P) - value), ),
    line=2, cex=1.5, font=2, adj=0)
# write the plot to file
dev.off()
}
if (args[2]=="6"){
  #install.packages(c("devtools","RcppArmadillo", "data.table", "Matrix"),
↪dependencies=TRUE)
  #library(devtools)
  #install_github("tshmak/lassosum")
  library(lassosum)
  # Prefer to work with data.table as it speeds up file reading
  library(data.table)
  library(methods)
  library(magrittr)
  # For multi-threading, you can use the parallel package and
  # invoke cl which is then passed to lassosum.pipeline
  library(parallel)
  # This will invoke 2 threads.
  cl <- makeCluster(2)
  result <-paste("./",args[1],"files","Data.QC.gz",sep="//")

  sum.stat <- result
  result <-paste("./",args[1],"files","test.QC",sep="//")
  bfile <- result
  # Read in and process the covariates
  result <-paste("./",args[1],"test","YRI.covariate",sep="//")
  covariate <- fread(result)
  result <-paste("./",args[1],"files","test.eigenvec",sep="//")

```

(continues on next page)

(continued from previous page)

```

pcs <- fread(result) %>% setnames(., colnames(.), c("FID","IID", paste0("PC",1:6)))
# Need as.data.frame here as lassosum doesn't handle data.table
# covariates very well

#pcs$FID <- as.character(pcs$FID)
#pcs$IID <- as.character(pcs$FID)

#pcs$FID <- paste(pcs$FID, pcs$FID,sep="_")
#pcs$IID <- paste(pcs$IID, pcs$IID,sep="_")
print(head(covariate))
print(head(pcs))

cov <- merge(covariate, pcs)

# We will need the EUR.hg19 file provided by lassosum
# which are LD regions defined in Berisa and Pickrell (2015) for the European
↪population and the hg19 genome.
ld.file <- "EUR.hg19"
# output prefix
prefix <- "EUR"
# Read in the target phenotype file
result <-paste("./",args[1],"test","YRI.pheno",sep="//")
bfile <-paste("./",args[1],"files","test.QC",sep="//")
target.pheno <- fread(result)[,c("FID", "IID", "phenotype")]
print(head(target.pheno))

# Read in the summary statistics
ss <- fread(sum.stat)
# Remove P-value = 0, which causes problem in the transformation
ss <- ss[!P == 0]
# Transform the P-values into correlation
cor <- p2cor(p = ss$P,
             n = ss$N,
             sign = log(ss$OR)
)
result <-paste("./",args[1],"test","test.fam",sep="//")

fam <- fread(result)
#fam$V1 <- as.character(fam$V1)
#fam$V2 <- as.character(fam$V2)

#fam$V1 <- paste(fam$V1, fam$V1,sep="_")
#fam$V2 <- paste(fam$V2, fam$V2,sep="_")
#fam$V6 <- paste(fam$V2, fam$V2,sep="_")
fam$V6[fam$V6==1]<-0
fam$V6[fam$V6==2]<-1

fam[,ID:=do.call(paste, c(.SD, sep=":")),.SDcols=c(1:2)]

# Run the lassosum pipeline
# The cluster parameter is used for multi-threading

```

(continues on next page)

(continued from previous page)

```
# You can ignore that if you do not wish to perform multi-threaded processing
out <- lassosum.pipeline(
  cor = cor,
  chr = ss$CHR,
  pos = ss$BP,
  A1 = ss$A1,
  A2 = ss$A2,
  ref.bfile = bfile,
  test.bfile = bfile,
  LDblocks = ld.file,
  cluster=cl
)
# Store the R2 results

result <-paste("./",args[1],"result","YRI.result",sep="//")

target.res <- validate(out, pheno = as.data.frame(target.pheno), covar=as.data.
↪frame(cov))
# Get the maximum R2
help(validate)
result <-paste("./",args[1],"result","test.txt",sep="//")

lapply(target.res[["best.pgs"]], write, result, append=TRUE, ncolumns=1000)
r2 <- max(target.res$validation.table$value)^2
print(r2)
}
```

2.5 Step 3 - Pvaluethreshold

2.5.1 Code execution

```
python pvalue.py "Directory" "P-value Threshold"
For example: python pvalue.py 1 5e-50
```

2.5.2 Actual Code in Pvaluethreshold.py

```
directory = sys.argv[1]

# Read the QC GWAS.
pvalues = pd.read_csv(directory+os.sep+"files/Data.QC.gz",compression='gzip',sep="\t")
print(pvalues.head())

# The selection of p-value is explained in the manuscript. We started from the lower p-
↪value threshold and then moved to significant p-value thresholds.
pvalue = float(sys.argv[2])
pvalues['P']=pd.to_numeric(pvalues['P'],errors='coerce')
```

(continues on next page)

(continued from previous page)

```
subpvalues = pvalues[pvalues['P']<=float(sys.argv[2])]

# Make a directory in which sub-dataset will be stored.
if not os.path.isdir(directory+os.sep+"pv_"+str(float(sys.argv[2]))):
    os.mkdir(directory+os.sep+"pv_"+str(float(sys.argv[2])))

subpvalues.to_csv(directory+os.sep+"pv_"+str(float(sys.argv[2]))+os.sep+str(float(sys.
↳argv[2]))+'.txt', columns=['SNP'],index=False,header=False)

# Extract selected SNPs from the test set.
os.system("./plink --bfile ./"+directory+os.sep+"test/test --extract ./"+directory+os.
↳sep+"pv_"+str(float(sys.argv[2]))+os.sep+str(float(sys.argv[2]))+".txt --recodeA --out
↳"+directory+os.sep+"pv_"+str(float(sys.argv[2]))+os.sep+"ptest")

# Extract selected SNPs from the training set.
os.system("./plink --bfile ./"+directory+os.sep+"train/train --extract ./
↳"+directory+os.sep+"pv_"+str(float(sys.argv[2]))+os.sep+str(float(sys.argv[2]))+".txt -
↳-recodeA --out "+directory+os.sep+"pv_"+str(float(sys.argv[2]))+os.sep+"ptrain")
```

2.6 Step 4 - MachineLearning

2.6.1 Code execution

```
python MachineLearning.py (DirectoryName in which files will be stored)
For example: python MachineLearning.py 1
```

2.6.2 Actual Code in MachineLearning

Helper Functions and Imports.

```
from __future__ import absolute_import, division, print_function
import argparse
from tensorflow.keras import Model, layers
import numpy as np

import pandas as pd
from sklearn.preprocessing import StandardScaler
from numpy import genfromtxt
from sklearn import svm

import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import warnings
warnings.filterwarnings('ignore')
```

(continues on next page)

(continued from previous page)

```

from datetime import datetime
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import StratifiedKFold
scaler = StandardScaler()
import tensorflow.keras
from sklearn.metrics import precision_score, recall_score, accuracy_score
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Dropout, Activation, ActivityRegularization
from tensorflow.keras.utils import to_categorical
from sklearn import metrics
import tensorflow as tf
from tensorflow.keras.layers import LSTM, GRU, Bidirectional
from sklearn import preprocessing
from tensorflow.keras.layers import Reshape
import seaborn as sn
import matplotlib.pyplot as plt
from pylab import rcParams
import sys
import os
from sklearn import tree, ensemble
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import make_pipeline
from sklearn.metrics import roc_auc_score

def metric(name, best_model, x_train, y_train, x_test, y_test):

    y_pred = best_model.predict(x_test)
    sn.set(font_scale=2)
    rcParams['figure.figsize'] = 7, 7
    confusion_matrix = pd.crosstab(y_test.argmax(axis=1), y_pred.argmax(axis=1),
    ↪ rownames=['Actual'], colnames=['Predicted'])
    sn.heatmap(confusion_matrix, annot=True)

    plt.savefig(args.path+os.sep+name+"Test.png")
    plt.clf()
    confusion_matrix = pd.crosstab(y_train.argmax(axis=1), best_model.predict(x_train).
    ↪ argmax(axis=1), rownames=['Actual'], colnames=['Predicted'])
    sn.heatmap(confusion_matrix, annot=True)

    plt.savefig(args.path+os.sep+name+"Train.png")
    plt.clf()

def plotting(history):
    fig = plt.figure()
    history_dict = history.history
    print(history_dict.keys())
    plt.subplot(2,1,1)
    plt.plot(history_dict['accuracy'])

```

(continues on next page)

(continued from previous page)

```
plt.plot(history_dict['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Training Set', 'Validation Set'], loc='lower right')

plt.subplot(2,1,2)

plt.plot( history_dict['loss'])
plt.plot( history_dict['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['Training Set', 'Validation Set'], loc='upper right')

plt.tight_layout()
```

Machine learning models. This code is designed so that it can be used for hyper-parameter optimization.

```
import pandas as pd
from os import makedirs

def model1(name,activationfunction=["sigmoid"],dropout=[5],optimizer = ["Adam"],batch_
↪size =[5],epochs=[30],l1=[0.01],l2=[0],validationsplit=[0.3]):
    for activation in activationfunction:
        for drop in dropout:
            for opt in optimizer:
                for b in batch_size:
                    for e in epochs:
                        for v in validationsplit:
                            model = Sequential()
                            model.add(Dense(100, input_shape=(x_train.shape[1],)))
                            model.add(Activation(activation))
                            model.add(Dropout(drop))
                            model.add(Dense(50))
                            model.add(Activation(activation))
                            model.add(Dropout(drop))
                            model.add(Dense(1))
                            model.add(Activation("sigmoid"))
                            model.compile(loss='binary_crossentropy', metrics=['accuracy'],
↪optimizer=opt)

                            history = model.fit(X_train, Y_train,batch_size=b, epochs=e,
↪validation_split=v,verbose=0)
                            loss, acc = model.evaluate(X_test, Y_test)
                            Y_pred = model.predict(X_test)

                            print(str(activation),str(drop),str(opt),str(b),str(e),str(v))
                            print("train AUC",roc_auc_score(Y_train, model.predict(X_train)))
                            print("test AUC",roc_auc_score(Y_test, model.predict(X_test)))
```

(continues on next page)

(continued from previous page)

```
# Return class prediction probabilities.
return np.amax(Y_pred,axis=1)
```

Training model for all p-value thresholds.

```
direc = sys.argv[1]
trainpath= "./"+direc+os.sep+"train/"
testpath = "./"+direc+os.sep+"test/"
allsnps = os.listdir("./"+direc)

from xgboost import XGBClassifier
for files in allsnps:
    #print(files)
    if "_snps" not in files and "pv_" in files:
        x_train = pd.read_csv("./"+direc+os.sep+files+os.sep+'ptrain.raw', sep="\s+")
        x_test = pd.read_csv("./"+direc+os.sep+files+os.sep+'ptest.raw', sep="\s+")
        y_train = pd.read_csv(trainpath+'YRI.pheno', sep="\s+")
        y_test= pd.read_csv(testpath+'YRI.pheno', sep="\s+")
        x_train = x_train.iloc[:,6:].values
        x_test = x_test.iloc[:,6:].values
        y_train = y_train.iloc[:,2:].values
        y_test = y_test.iloc[:,2:].values

        scaler = StandardScaler()
        std_scale = preprocessing.StandardScaler().fit(x_train)
        x_train = std_scale.transform(x_train)
        x_test = std_scale.transform(x_test)

        X_train = x_train
        X_test = x_test
        Y_train = y_train
        Y_test = y_test

        activationFunctions = ["relu"]
        ddropout = [0.2]
        optimizer = ["Adam"]
        batchsize = [50]
        epochsnumber = [100]
        validation = [0.3]
        data = model1("ANN1",activationFunctions,ddropout,optimizer,batchsize,epochsnumber,
        l1=[0],l2=[0],validationssplit=validation)
        x = pd.DataFrame()
        x['0'] = data
```

(continues on next page)

(continued from previous page)

```
x.to_csv(direc+os.sep+files+os.sep+"ML_probability",header=False, index=False)
```

2.7 Step 5 - GetResults

2.7.1 Code execution

```
python GetResults.py (DirectoryName in which files will be stored)
For example: python GetResults.py 1
```

2.7.2 Actual Code in GetResults

```
import pandas as pd
import sys
import os
import glob
import shutil
import sys
import numpy as np
import scipy as sp
from scipy import stats
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
import math
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stats
import math
import matplotlib.pyplot as plt
from scipy.stats import norm
import statistics
import seaborn as sns

direc = sys.argv[1]

def NormalizeData(data):
    return (data - np.min(data)) / (np.max(data) - np.min(data))
def calculateAUCPRScice(direc):
    best = pd.read_csv(direc+os.sep+"result"+os.sep+"PRScice_PRS.best", sep="\s+")
    pheno = pd.read_csv(direc+os.sep+"/test/YRI.pheno", sep="\s+")

    temp = NormalizeData(best["PRS"].values)
    best["PRS"] = NormalizeData(np.array(best["PRS"].values))
```

(continues on next page)

(continued from previous page)

```

best["PRS"].values[best["PRS"] >=0.5] = 1
best["PRS"].values[best["PRS"] < 0.5] = 0

print("AUC",roc_auc_score(np.array(best["PRS"].values), np.array(pheno['phenotype'].
↪values)))
a= best["PRS"].values
b = pheno['phenotype'].values
accuracy = len([a[i] for i in range(0, len(a)) if a[i] == b[i]]) / len(a)
print("Accuracy",accuracy)
return temp

def calculateAUCplink(direc):
    pheno = pd.read_csv(direc+os.sep+"/test/YRI.pheno",sep="\s+")
    files = os.listdir(direc+os.sep+"files")
    maxxacc=0
    maxauc = 0
    profile = ""
    temp = []
    for loop in files:

        if ".profile" in loop:
            # Find AUC for all profiles.
            best = pd.read_csv(direc+os.sep+"files"+os.sep+loop,sep="\s+")
            best["SCORE"] = NormalizeData(best["SCORE"].values)
            best["SCORE"].values[best["SCORE"] >=0.5] = 1
            best["SCORE"].values[best["SCORE"] < 0.5] = 0

            if maxauc<roc_auc_score(best["SCORE"].values, pheno['phenotype'].values):
                maxauc = roc_auc_score(best["SCORE"].values, pheno['phenotype'].values)
                temp = NormalizeData(pd.read_csv(direc+os.sep+"files"+os.sep+loop,sep="\s+")[
↪"SCORE"].values)
                profile = loop
                a= best["SCORE"].values
                b = pheno['phenotype'].values
                accuracy = len([a[i] for i in range(0, len(a)) if a[i] == b[i]]) / len(a)
                maxxacc = accuracy
            print("AUC",maxauc)
            print("Accuracy",maxxacc)
            print("Profile", profile)
            return temp

def calculateAUClasso(direc):
    best = pd.read_csv(direc+os.sep+"result"+os.sep+"test.txt",sep="\s+",header=None)
    pheno = pd.read_csv(direc+os.sep+"/test/YRI.pheno",sep="\s+")

    best[0] = NormalizeData(best[0].values)
    temp = NormalizeData(best[0].values)
    best[0].values[best[0] >=0.5] = 1
    best[0].values[best[0] < 0.5] = 0

```

(continues on next page)

(continued from previous page)

```

print("AUC",roc_auc_score(best[0].values, pheno['phenotype'].values))
a= best[0].values
b = pheno['phenotype'].values
accuracy = len([a[i] for i in range(0, len(a)) if a[i] == b[i]]) / len(a)
print("Accuracy",accuracy)
return temp

def calculateAUCMachineLearning(direc):
    pheno = pd.read_csv(direc+os.sep+"/test/YRI.pheno",sep="\s+")

    files = os.listdir(direc)
    maxxacc=0
    maxauc = 0
    profile = ""
    temp = []
    for loop in files:
        # Find AUC for all p-values and machine learning.
        if "pv_" in loop:
            best = pd.read_csv(direc+os.sep+loop+os.sep+"ML_probability",sep="\s+",
↪header=None)
            #plt.hist(best["PRS"].values)
            #plt.show()

            #best["PRS"].values[best["PRS"] >= sum(best["PRS"].values)/len(best["PRS"].
↪values)] = 1
            #best["PRS"].values[best["PRS"] < sum(best["PRS"].values)/len(best["PRS"].
↪values)] = 0
            best[0] = NormalizeData(best[0].values)

            best[0].values[best[0] >=0.5] = 1
            best[0].values[best[0] < 0.5] = 0

            if maxauc<roc_auc_score(best[0].values, pheno['phenotype'].values):
                maxauc = roc_auc_score(best[0].values, pheno['phenotype'].values)
                temp = NormalizeData(pd.read_csv(direc+os.sep+loop+os.sep+"ML_probability",
↪sep="\s+",header=None)[0].values)
                profile = loop
                a= best[0].values
                b = pheno['phenotype'].values
                accuracy = len([a[i] for i in range(0, len(a)) if a[i] == b[i]]) / len(a)
                maxxacc = accuracy
            print("AUC",maxauc)
            print("Accuracy",maxxacc)
            print("Profile", profile)
            return temp

x = calculateAUCPRScice(direc)
n_samples = len(x)
rng = np.random.RandomState(0)
x = sns.distplot(x,hist=False,color='y')

```

(continues on next page)

(continued from previous page)

```
x = calculateAUCPlink(direc)
x = sns.distplot(x,hist=True,color = 'r')

x = calculateAUClasso(direc)
x = sns.distplot(x, hist=True,color='g')

x = calculateAUCMachineLearning(direc)
x = sns.distplot(x, hist=True,color='b')

x.figure.legend(labels=['PRScice', 'Plink', 'Lassosum', 'Machine Learning'])
x.figure.savefig(direc+os.sep+"result"+os.sep+"output.png")
```