Heritability, Genetic variation, and the number of risk SNPs effect on deep learning and polygenic risk scores AUC

Release 1.0

Muhammad Muneeb and Samuel F. Feng

Feb 28, 2022

CONTENTS

Sample files and code used for this study are available on Google Drive.

If we generate 400 different datasets for 5000 people, then saving them would take 30*400 GB = 1.2 TB space, and it is nearly impossible to keep all datasets on a high-performance computing cluster. We mitigated this issue by converting the code to a producer and consumer scenario. The producer generates the data, the consumer performs the analysis after the analysis results are saved, old data is deleted, and the producer deletes new data. The space consumption is almost constant throughout the analysis.

CHAPTER

ONE

CONTENT

1.1 Step 0 - Generate Data

The generated dataset is available on Google Drive.

1.1.1 Dataset

If we generate 400 different datasets for 5000 people, it will take 30*400 GB = 1.2 TB space, and it is nearly impossible to keep all datasets on a high-performance computing cluster. We mitigated this issue by converting the code to a producer and consumer scenario. The producer generates the data, the consumer performs the analysis after the analysis results are saved, old data is deleted, and the producer deletes new data. The space consumption is almost constant throughout the analysis.

1.1.2 Dataset generation process

```
Following files and directories are required to generate the dataset.
1. Merge (Directory which contains the merged files)
2.1 generate.py  (Python script to generate data from Hapgen2)
2.2 generate.sh  (Bash script to deploy generate.py on HPC)
3. gtool/plink  (Tools for processing)
4. merge.py  (Code for merging in python)
5. merge.sh  (Bash script to deploy merge.py 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 (Directories that contain PhenotypeSimulator data)
11 1, 2, 3, and 4 (Directories that contain test, train, and result for 1 2 3 4␣
↪iterations, respectively)
12 generatingdata.py (This file takes parameters and run the process for specific␣
↪parameters)
13 generatingdata.sh (Bash script to deploy generatingdata.py on HPC)
14 parameters.csv (This file contains the parameters ('Number of risk SNPs',
↪'Heritability', 'Genetic Variation'))
```

1.1.3 Dataset generation - Generate data using Hapgen2 (generate.py)

Execute this code one time to generate data through Hapgen2.

```
# Execute this code only one time.

import os
import sys
import time

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"

# Generated data is saved in the main directory.
dest ="/l/proj/kuin0025/MuhammadMuneeb/mlvsprs/finalized/rawdata"

# Input to Hapgen is CEU.leg, CEU.haps, and CEU.map
# 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 50 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")
```

1.1.4 Dataset generation - Generate data using Hapgen2 (generate.sh)

Execute this code one time to generate data through Hapgen2.

```
#!/bin/sh
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --job-name=PSBenchmark
#SBATCH --time=1:00:00
#SBATCH --mem=3000MB
#SBATCH --partition=prod
#SBATCH --account=kuin0009
#SBATCH --output=Hapgen.%A_%a.out
#SBATCH --error=Hapgen.%A_%a.err
#SBATCH --array=1-100


module purge
module load miniconda/3
```

(continues on next page)

(continued from previous page)

```
radius=$(head -n $SLURM_ARRAY_TASK_ID people.txt | tail -n 1)
python generate.py  $radius
```

1.1.5 Dataset generation - Generate data using Hapgen2 (people.txt)

```
1
2
3
4
.
.
.
99
100
```

After the steps mentioned above, the genotype dataset will be finalized/rawdata directory.

1.1.6 Dataset generation - parameters.sh

```
Iteration,cNrSNP,Heritability,genVar,totalSNPeffect
2,5,0.05263158,0.9,0.047368422
3,5,0.10526316,0.9,0.094736844
4,5,0.15789474,0.9,0.142105266
5,5,0.21052632,0.9,0.189473688
6,5,0.26315789,0.9,0.236842101
7,5,0.31578947,0.9,0.284210523
8,5,0.36842105,0.9,0.331578945
9,5,0.42105263,0.9,0.378947367
10,5,0.47368421,0.9,0.426315789

# Note totalSNPeffect= Heritability*genVar
# cNrSNP = Number of risk SNPs
# genVar = Genetic Variation
```

1.1.7 Dataset generation - generatingdata.sh

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --job-name=Merge-5000
#SBATCH --time=48:00:00
#SBATCH --mem=1000MB
#SBATCH --partition=prod
#SBATCH --account=kuin0025
#SBATCH --output=head.%A_%a.out
#SBATCH --error=head.%A_%a.err
#SBATCH --array=1-1
```

(continues on next page)

(continued from previous page)

```


module purge
module load miniconda/3

python generatingdata.py
```

1.1.8 Dataset generation - generatingdata.py

```
# This is the starting point of the documentation.
# This file contains the code which executes other tasks like data generation,␣
↪calculating polygenic risk, and saving results.


import os
import sys
import pandas as pd
import sys
import time

os.system("rm -rf Results.txt")

# This file stores the parameters which should be passed to sample.R file to generate␣
↪the genotype data with specific parameters.

parameters = pd.read_csv("parameterstest.csv")
print(parameters)

# Parse the rows and call the sample.sh file.
for row in parameters.iterrows():
   text =os.popen('sbatch sample.sh '+str(row[1][1])+" "+str(row[1][3])+"
↪"+str(row[1][4])).read()
   aucs = [str(row[1][1]),str(row[1][3]),str(row[1][4])]
   textfile = open("Results.txt", "a+")

   # Write the parameters to specific files.
   for element in aucs:
      textfile.write(str(element))
   textfile.write('\n')
   textfile.close()
   time.sleep(5)
   print(text.split('\n')[0].split(" ")[-1])
   time.sleep(10)
   print(os.popen("squeue | grep "+text.split('\n')[0].split(" ")[-1]).read())

   # Wait until the above job is not finished.

   while len(os.popen("squeue | grep "+text.split('\n')[0].split(" ")[-1]).read())>0:
      pass

```

(continues on next page)

(continued from previous page)

```
    time.sleep(10)

    # Remove the log files.
    os.system('rm -rf PSBenchmark.*')

    # Merge all the genotype files
    text =os.popen('sbatch merge.sh').read()
    print(text.split('\n')[0].split(" ")[-1])
    time.sleep(10)

    # Wait for the merging.
    while len(os.popen("squeue | grep "+text.split('\n')[0].split(" ")[-1]).read())>0:
        pass


    time.sleep(10)
    os.system('rm -rf CEU_*')
    os.system('rm -rf TEMP.*')

    # Call division.sh file, and it will calculate PRS.

    text =os.popen('sbatch division.sh '+str(row[1][1])+" "+str(row[1][3])+"
↪"+str(row[1][4])).read()
    print(text.split('\n')[0].split(" ")[-1])
    time.sleep(10)
    while len(os.popen("squeue | grep "+text.split('\n')[0].split(" ")[-1]).read())>0:
        pass
    time.sleep(10)

    # Remove log files and other files
    os.system('rm -rf Merge')
    os.system('rm -rf prs.*')
    os.system('rm -rf {1..5}')
```

1.1.9 Dataset generation - sample.sh

```
#!/bin/sh
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --job-name=PSBenchmark
#SBATCH --time=1:00:00
#SBATCH --mem=3000MB
#SBATCH --partition=prod
#SBATCH --account=kuin0025
#SBATCH --output=PSBenchmark.%A_%a.out
#SBATCH --error=PSBenchmark.%A_%a.err
#SBATCH --array=1-100
```

(continues on next page)

(continued from previous page)

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

#PASS data to PhenotypeSimulator
#$1 = cNrSNP
#$2 = Genetic Variation
#$3 = Heritability

Rscript sample.R $radius $1 $2 $3
```

1.1.10 Dataset generation - sample.R

```
require("PhenotypeSimulator")



args = commandArgs(trailingOnly=TRUE)

start_time <- Sys.time()

simulating <- function(risksnp,genovar, number) {
numberofPeople = 50
print(args[1])
filem <-paste("CEU",strtoi(args[1]),sep="_")
filem <-paste(filem,"gen",sep=".")
filem <-paste("/l/proj/kuin0025/MuhammadMuneeb/mlvsprs/finalized/rawdata",filem,sep="/")
genotypefile <- filem
genotypefile <- gsub("\\.gen","", filem)


genVar <- genovar
noiseVar <- 1 - genVar
totalSNPeffect <- number
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 = risksnp, 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),
```

(continues on next page)

(continued from previous page)

```
                                  distConfounders = c("bin", "cat_norm", "cat_unif", "norm"),
                                  probConfounders = 0.2, catConfounders = c(3, 4), pcorr = 0.8,
                                  verbose = TRUE, seed = 3000)




result <-paste("CEU",strtoi(args[1]),sep="_")
out <- savePheno(phenotype, directory=getwd(),
                   outstring=result,
                   format=c("csv", "snptest"), verbose=FALSE)


}
print(as.double(args[2]))
print(as.double(args[3]))
print(as.double(args[4]))

simulating(as.double(args[2]),as.double(args[3]),as.double(args[4]))
```

1.1.11 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")
```

1.1.12 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
python generate.py $radius

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

1.1.13 Dataset generation - merge.py

```
import os
import pandas as pd

files = os.listdir("./")
if not os.path.isdir("Merge"):
os.mkdir("Merge")
print(files)
data = pd.DataFrame()
gendata = pd.DataFrame()
count=0
for direc in files:
   if "Merge" not in direc and "lib" 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)
```

(continues on next page)

(continued from previous page)

```
      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
gendata = pd.concat([frontgen,gendata],axis=1)
gendata.to_csv("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("Merge"+os.sep+"Ysim_snptest.sample",index=False, sep=" ")
```

1.2 Step 1 - Analysis

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.

1.2.1 Code execution

python dividedata.sh (Bash file to perform analysis)

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --job-name=ped
#SBATCH --time=48:00:00
#SBATCH --mem=1000MB
#SBATCH --partition=prod
#SBATCH --account=kuin0025
#SBATCH --output=prs.%A_%a.out
#SBATCH --error=prs.%A_%a.err
#SBATCH --array=1-5

module purge
module load miniconda/3

#This command divides the dataset into cases and controls
python dividedata.py $SLURM_ARRAY_TASK_ID

#This is for quality control steps and Polygenic Risk Score calculation.
python CalculatePRS.py $SLURM_ARRAY_TASK_ID

# Perform p-value thresholding on each Iteration.
python pvalue.py $SLURM_ARRAY_TASK_ID 1
python pvalue.py $SLURM_ARRAY_TASK_ID 0.1
python pvalue.py $SLURM_ARRAY_TASK_ID 0.01
python pvalue.py $SLURM_ARRAY_TASK_ID 0.001
```

(continues on next page)

(continued from previous page)

```


# Machine learning for each Iteration.
python pmodel.py $SLURM_ARRAY_TASK_ID

# Get results for each Iteration.
python getresults.py $SLURM_ARRAY_TASK_ID
```

1.2.2 Code in dividedata.py

See this link dividedata.

1.2.3 Actual Code in CalculatePRS.py

See this link CalculatePRS.

1.2.4 Actual Code in QCTarget.R

See this link QCTarget.

1.2.5 Actual Code in pvalue.py

See this link Pvaluethreshold.

1.2.6 Actual Code in pmodel.py

See this link MachineLearning.

1.3 Step 2 - GetResults

1.3.1 Code execution

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

1.3.2 Actual Code in GetResults

```
import pandas as pd
import sys
import os
import glob
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
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))

   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, roc_auc_score(np.array(best["PRS"].values), np.array(pheno['phenotype'].
↪values))


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:
```

(continues on next page)

(continued from previous page)

```
      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,maxauc

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

   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, AUC


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)
```

(continues on next page)

(continued from previous page)

```
        #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, maxauc

x,y1 = calculateAUCPRScice(direc)
n_samples = len(x)
rng = np.random.RandomState(0)
x = sns.distplot(x,hist=False,color='y')


x,y2 = calculateAUCPlink(direc)
x = sns.distplot(x,hist=True,color = 'r')

x,y3 = calculateAUClasso(direc)
x = sns.distplot(x, hist=True,color='g')


x,y4 = calculateAUCMachineLearning(direc)
x = sns.distplot(x, hist=True,color='b')

aucs = [y1,y2,y3,y4]
textfile = open("Results.txt", "a+")

# Write accuracy for each tool in a file.
for element in aucs:
   textfile.write(str(element))
textfile.write('\n')
textfile.close()
```

1.3.3 Helper Functions plotting

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from decimal import *

# Read the result file
data = pd.read_csv("Resultsparameter.txt",header=None)

# Read the parameter file
chart = pd.read_csv("parameters2.csv")

from statistics import mean

# Temporary variable to store the results separately
tempprsice = []
tempplink = []
templasso = []
tempml = []
prsice = []
plink = []
lasso = []
ml = []


for ind in data.index:
   if len(data[0][ind])>20:
      # Remove lines if length is less than 20 because it contains the parameter values.
      print(data[0][ind].split(".")[1:])
      tempprsice.append(float(data[0][ind].split(".")[1:][0][:2])/100)
      tempplink.append(float(data[0][ind].split(".")[1:][1][:2])/100)
      templasso.append(float(data[0][ind].split(".")[1:][2][:2])/100)
      tempml.append(float(data[0][ind].split(".")[1:][3][:2])/100)
      print(tempml)
   else:
      snp = data[0][ind].split(".")[0]
      if len(tempml)>0:
         prsice.append(mean(tempprsice))
         plink.append(mean(tempplink))
         lasso.append(mean(templasso))
         ml.append(mean(tempml))
         tempprsice = []
         tempplink = []
         templasso = []
         tempml = []
prsice.append(mean(tempprsice))
plink.append(mean(tempplink))
lasso.append(mean(templasso))
ml.append(mean(tempml))
tempprsice = []
tempplink = []
templasso = []
```

(continues on next page)

(continued from previous page)

```
tempml = []

prsice = np.array(prsice)
plink = np.array(plink)
lasso = np.array(lasso)
ml = np.array(ml)

del chart["Iteration"]

chart["Machine learning AUC"] = ml
chart["PRSice AUC"] = prsice
chart["Plink AUC"] = plink
chart["Lasso AUC"] = lasso
chart.rename({'cNrSNP': 'Number of risk SNPs', 'Heritibility': 'Heritability','genVar':
↪'Genetic Variation'}, axis=1, inplace=True)

# This file contains the parameter values and tool's AUC at each low. AUC for each tool␣
↪is an average of 5 iterations.
chart.to_csv("finalparameter2.csv",index=False)
```

1.3.4 Helper Functions plotting 2

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from decimal import *
font = {'family' : 'normal',
      'size'   : 15}
matplotlib.rc('font', **font)

matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)

from scipy.ndimage.filters import gaussian_filter1d
chart1 = pd.read_csv("finalparameter1.csv")
chart2 = pd.read_csv("finalparameter2.csv")
chart3 = pd.read_csv("finalparameter3.csv")
chart4 = pd.read_csv("finalparameter4.csv")

smooth= 2
plt.plot(range(1,len(chart1["Machine learning AUC"].values)+1),gaussian_filter1d(chart1[
↪"Machine learning AUC"].values,sigma=smooth),"g--",color="blue",label='ML, G = 0.9')
plt.plot(range(1,len(chart1["Machine learning AUC"].values)+1),gaussian_filter1d(chart1[
↪"PRSice AUC"].values,sigma=smooth),"g-.",color="blue", label='PRSice, G = 0.9')
plt.plot(range(1,len(chart1["Machine learning AUC"].values)+1),gaussian_filter1d(chart1[
↪"Plink AUC"].values,sigma=smooth),"g:",color="blue",label='Plink, G = 0.9')
plt.plot(range(1,len(chart1["Machine learning AUC"].values)+1),gaussian_filter1d(chart1[
↪"Lasso AUC"].values,sigma=smooth),color="blue", label='Lasso, G = 0.9')
```

(continues on next page)

(continued from previous page)

```
plt.plot(range(1,len(chart2["Machine learning AUC"].values)+1),gaussian_filter1d(chart2[
↪"Machine learning AUC"].values,sigma=smooth),"g--",color="red",label='ML, G = 0.8')
plt.plot(range(1,len(chart2["Machine learning AUC"].values)+1),gaussian_filter1d(chart2[
↪"PRSice AUC"].values,sigma=smooth),"g-.",color="red", label='PRSice, G = 0.8')
plt.plot(range(1,len(chart2["Machine learning AUC"].values)+1),gaussian_filter1d(chart2[
↪"Plink AUC"].values,sigma=smooth),"g:",color="red",label='Plink, G = 0.8')
plt.plot(range(1,len(chart2["Machine learning AUC"].values)+1),gaussian_filter1d(chart2[
↪"Lasso AUC"].values,sigma=smooth),color="red", label='Lasso, G = 0.8')

plt.plot(range(1,len(chart3["Machine learning AUC"].values)+1),gaussian_filter1d(chart3[
↪"Machine learning AUC"].values,sigma=smooth),"g--",color="black",label='ML, G = 0.7')
plt.plot(range(1,len(chart3["Machine learning AUC"].values)+1),gaussian_filter1d(chart3[
↪"PRSice AUC"].values,sigma=smooth),"g-.",color="black", label='PRSice, G = 0.7')
plt.plot(range(1,len(chart3["Machine learning AUC"].values)+1),gaussian_filter1d(chart3[
↪"Plink AUC"].values,sigma=smooth),"g:",color="black",label='Plink, G = 0.7')
plt.plot(range(1,len(chart3["Machine learning AUC"].values)+1),gaussian_filter1d(chart3[
↪"Lasso AUC"].values,sigma=smooth),color="black", label='Lasso, G = 0.7')

plt.plot(range(1,len(chart4["Machine learning AUC"].values)+1),gaussian_filter1d(chart4[
↪"Machine learning AUC"].values,sigma=smooth),"g--",color="g",label='ML, G = 0.6')
plt.plot(range(1,len(chart4["Machine learning AUC"].values)+1),gaussian_filter1d(chart4[
↪"PRSice AUC"].values,sigma=smooth),"g-.",color="g", label='PRSice, G = 0.6')
plt.plot(range(1,len(chart4["Machine learning AUC"].values)+1),gaussian_filter1d(chart4[
↪"Plink AUC"].values,sigma=smooth),"g:",color="g",label='Plink, G = 0.6')
plt.plot(range(1,len(chart4["Machine learning AUC"].values)+1),gaussian_filter1d(chart4[
↪"Lasso AUC"].values,sigma=smooth),color="g", label='Lasso, G = 0.6')

plt.axvline(19, 0, 1,linestyle='--')
plt.axvline(19*2, 0, 1,linestyle='--')
plt.axvline(19*3, 0, 1, linestyle='--')
plt.axvline(19*4, 0, 1, linestyle='--')
plt.legend(fontsize=100) # using a size in points
plt.legend(loc='best')
plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1.2),
         ncol=4, fancybox=True, shadow=True)

plt.show()
```