Logistic LASSO in 22q deletion data

Model

Notation:

- n individuals
- p covariates (e.g. SNPs minor allele counts: $0, 1, 2$)
- Binary response $Y_{n\times 1}$ (e.g. case-control)
- Data matrix $X_{n\times p}$

The loglikelihood of the logistic regression is

$$l(\beta) = \sum_{i=1}^{n} y_i \log \pi_i(\beta) + (1 - y_i) \log(1 - \pi_i(\beta))$$

where $\beta = (\beta_1, ..., \beta_p)^T$ and

$$\pi_i(\beta) = \frac{\exp(x_i^T \beta)}{1 + \exp(x_i^T \beta)}$$

Since $n << p$, we need to add some restriction to the optimization of β. That is, we want to assume that most of the $\beta_i = 0$.

So, by LASSO technique, we want to maximize the following penalized likelihood:

$$\max_{\beta} l(\beta) - \lambda \sum_{j=1}^{p} |\beta_j|$$

Data

519 people (229 males, 290 females) with 259 cases (psychosis) and 260 controls with whole genome genotype data (chromosomes 1-23 with roughly 31 million SNPs). All these individuals have a deletion in chromosome 22 that is suspected to cause psychotic diseases, like schizophrenia. Scientists want to identify SNPs that are related to the presence of psychosis, to later use this information in the general population (the ones that do not have this deletion, but could still have psychosis). The detection of SNPs associated with psychosis might help to identify pathways that could clarify the progression to psychosis in the general population.

We will study first only the chromosome 22: 519×450438 matrix, thinking that the SNPs close to the deletion could have some effect on the disease.

After talking to Kevin Keys, he pointed at the phase transition here and the realization that we have few data compared to the number of covariates.

Chromosome 22

We can read the BED files into julia, and convert to a numerical matrix 519×450438 of minor allele counts: each entry is $0, 1, 2$ depending on the number of minor alleles for that individual in that SNP.

```
using SnpArrays
datafolder = "/Users/Clauberry/Documents/gwas/data/22q/22q_files_NEW/"
chr22 = SnpArray(string(datafolder,"bedfiles/22q-chr22"))
#chr22mat=convert(Matrix{Float64},chr22)
# convert to SparseMatrixCSC{Float32, UInt32} matrix
#@time snpbin29c_f32sp = convert(SparseMatrixCSC{Float32, UInt32}, snpbin29c)
maf, minor_allele, missings_by_snp, missings_by_person = summarize(chr22)
# - minor_allele: a BitVector indicating the minor allele for each SNP.
# minor_allele[j]==true means A1 is the minor allele for SNP j;
# minor_allele[j]==false means A2 is the minor allele for SNP j.
# - missings_by_snp: number of missing genotypes for each snp.
# - missings_by_person: number of missing genotypes for each person.
size(missings_by_snp) #(450438,)
size(missings_by_person) #(519,)
sum(missings_by_snp) / length(chr22) ## 0.004832068356057223
# filter out rare SNPs with MAF < 0.05
#chr22orig = chr22
chr22 = chr22[:, maf .  0.05]
```

Low missingness, so we can impute. We also get rid of rare SNPs to avoid constant columns that would affect the convergence of the optimization in `Lasso.jl` (see below).

Using a subset of 10,000 SNPs

To make it easier first, let's take a subset of 10000 SNPs. We do this before converting to matrix, because converting the whole matrix of 450438 uses ~8Gb memory.

```
r = 12345
srand(r)
using DataFrames
ind = sample(1:size(chr22,2),10000,replace=false)
writetable(string("subset",r,".txt"), DataFrame(x=ind))
chr22sub = chr22[:,ind]
```

Now, we convert to the types needed by the julia packages: `Lasso` and `SparseRegression`. We use `impute=true` because they cannot handle missing data.

```
@time X=convert(Matrix{Float64},chr22sub,impute=true)
```

```
0.724464 seconds (10.60 M allocations: 201.641 MiB, 64.02% gc time)
```

Next, we have to read the response vector y:

```
dat = readtable(string(datafolder,"bedfiles/22q-chr22.fam"), separator=' ', header=fal
y = convert(Vector,dat[:,6])
y = y-1 ## 0=controls, 1=cases
y = convert(Array{Float64,1},y)
```

Exploration

```
using Gadfly
m=mean(chr22mat,1)
x = collect(1:1:length(m))
plot(x=x[1:1000],y=m[1:1000])
```

Fit

Next, we can fit a logistic LASSO regression. First we are doing this in the `understanding-lasso.md` file.

Using Lasso.jl

We managed to make `Lasso` work. Documentation here The command is fitting a pure Lasso model (default is $\alpha = 1$), with 100 values of λ and naive coordinate descent. The convergence criterion is with the coefficients, by default.

For this example, we got rid at the beginning of rare SNPs, but this is how we would get rid of constant columns:

```
#ctecol = [std(X[:,i])==0 for i in 1:size(X,2)]
#@show sum(ctecol)
#X = X[:,.!ctecol];
```

```
using Lasso
#f = fit(LassoPath,X,y,Binomial(),LogitLink())
#ERROR: maximum number of coefficients 1038 exceeded at   = 0.06694246927158318 ( j=0
#f = @time fit(LassoPath,X,y,Binomial(),LogitLink(), maxncoef=10000)
#ERROR: coordinate descent failed to converge in 100000 iterations at   = 0.06694246
f2=@time fit(LassoPath,X,y,Bernoulli(),LogitLink())
```

```
7.073524 seconds (50.08 k allocations: 42.645 MiB, 0.07% gc time)
```

This works!! They use $\lambda \in (0.0669, 0.0007)$ with $(1, 427)$ covariates included in the model. In `f2.coefs` we have the beta coefficients in a `SparseMatrixCSC` of model coefficients. Columns correspond to fit models; rows correspond to predictors. Here `10000×100 SparseMatrixCSC{Float64,Int64} with 28640 stored entries`

```
@show f2.coefs[:,2]
```

```
f2.coefs[:, 2] =   [1243 ]  =  0.000450402
  [8611 ]  =  -0.0227341
```

```
@show f2.coefs[:,3]
```

```
f2.coefs[:, 3] =   [1243 ]  =  0.0235091
  [8611 ]  =  -0.0439576
```

```
## real index (if we got rid of constant columns):
#ind2 = find(x->x==false,ctecol)
@show ind[8550]
```

```
ind[8550] = 23476
```

```
@show ind[[4307,3098]]
```

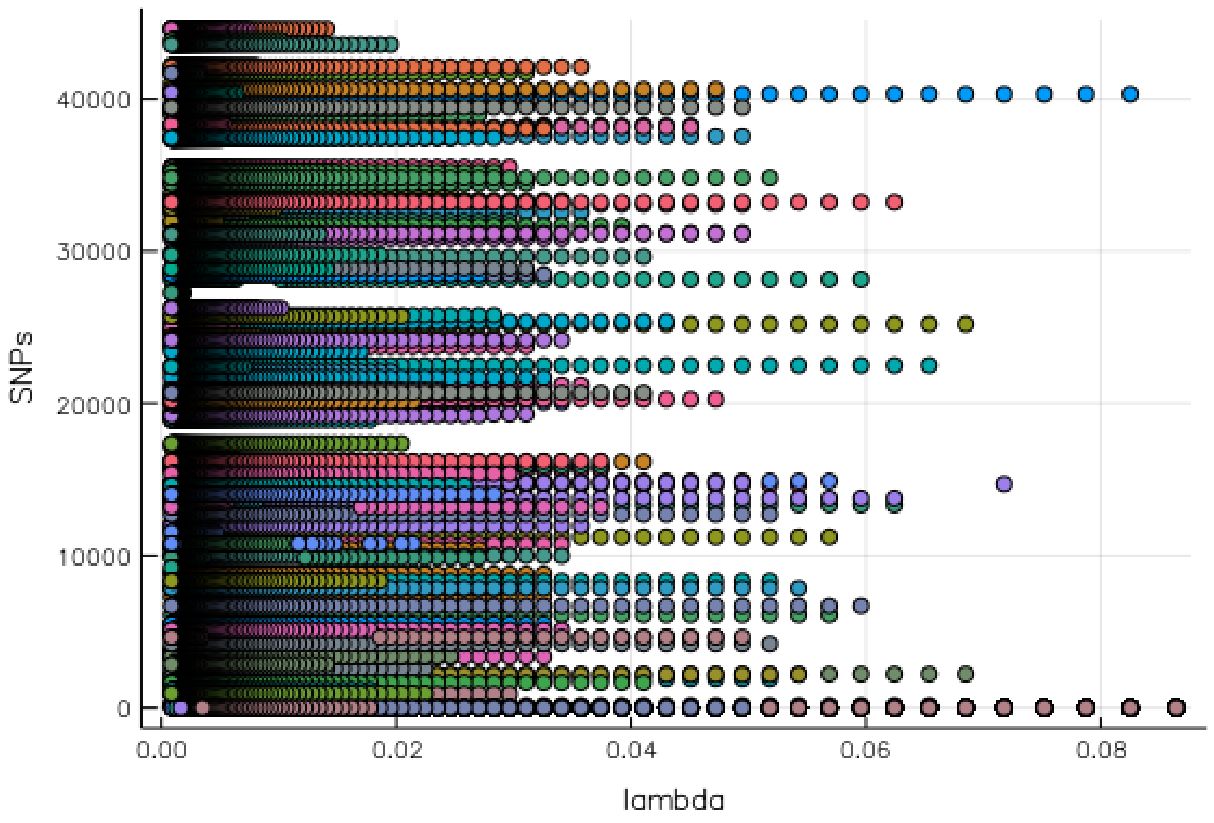
```
ind[[4307, 3098]] = [31991, 26369]
```

This means that the first SNP to be selected in the model (with the largest $\lambda = 0.0893361$) was SNP 22249, followed by SNPs 22272,40377.

Now we can plot the SNPs that appear in the last model (with smallest $\lambda = 0.0007013$). This is a bar plot similar to the ones in Topological Data Analsys (TDA), in which we show for which value of λ each coefficient appears in the model. The bigger the λ, the more stringent the penalty. So, SNPs whose line starts more towards the right were selected for the model sooner.

```
using Plots
ii = find(x->x!=0,f2.coefs[:,100])
l = length(ii)

scatter(f2. ,[(f2.coefs[ii[i],:] .!=0).*ind[ii[i]] for i in 1:Int(floor(l/2))],
xlabel="lambda", ylabel="SNPs",legend=false)
scatter(f2. ,[(f2.coefs[ii[i],:] .!=0).*ind[ii[i]] for i in (Int(floor(l/2))+1):l],
xlabel="lambda", ylabel="SNPs",legend=false)
```



The bottom line at zero should be ignored.

Using SparseRegression.jl

Now, we can fit the penalized logistic regression with LASSO. There are other options for penalties (see here)

It turns out that `SparseRegression` needs the response to be $-1, 1$, instead of $0, 1$. I had a problem before, see a github issue here.

```
using SparseRegression
y[y.==0] = -1
##obs = Obs(X, y)
##s = SModel(obs, LogisticRegression(), L1Penalty()) ## need to choose
## aqui voy: sparse regression cambio!!
s = SModel(X,y, LogitDistLoss(), L1Penalty()) ## need to choose
@time learn!(s, strategy(ProxGrad(s,0.001), MaxIter(10000), Converged(coef)))
```

```
0.083072 seconds (39 allocations: 239.750 KiB)
```

```
find(x -> x != 0,s. ) ##0 SNPs for  =0.1
find(x -> x < 0,s. )
```

```
0-element Array{Int64,1}
```

Now, we can try with the best `GLMNet` lambda:

```
##obs = Obs(X, y)
s = SModel(X,y, LogitDistLoss(), L1Penalty(),fill(0.062,size(X,2))) ## need to choose
@time learn!(s, strategy(ProxGrad(s,0.001), MaxIter(10000), Converged(coef)))
```

```
0.015625 seconds (33 allocations: 239.469 KiB)
```

```
find(x -> x != 0,s. ) ##0 SNPs for  =0.1
find(x -> x < 0,s. )
```

```
0-element Array{Int64,1}
```

Josh Day says that `SparseRegression` and `GLMNet` parametrize the betas differently. So, let's try with a smaller lambda:

```
##obs = Obs(X, y)
s = SModel(X,y, LogitDistLoss(), L1Penalty(),fill(0.00062,size(X,2))) ## need to choos
@time learn!(s, strategy(ProxGrad(s,0.001), MaxIter(10000), Converged(coef)))
```

```
28.713092 seconds (80.02 k allocations: 764.928 MiB, 0.59% gc time)
```

```
#find(x -> x != 0,s. ) ##5415 SNPs for  =0.00062
#find(x -> x < 0,s. ) ## 2657 SNPs
```

What happens when we use another penalty (we don't want to, because we can do post selection with the LASSO estimates):

```
s = SModel(X,y, LogitDistLoss(), ElasticNetPenalty()) ## need to choose
@time learn!(s, strategy(ProxGrad(s,0.001),MaxIter(10000), Converged(coef)))
```

```
7.405221 seconds (23.36 k allocations: 223.285 MiB, 0.60% gc time)
```

```
find(x -> x != 0,s. ) ##58 SNPs for  =0.1,  =0.5
find(x -> x < 0,s. ) ## 0 SNPs, weird
```

```
1-element Array{Int64,1}:
 8611
```

Using GLMNet

We will compare to `GLMNet.jl`, which we do not love because it is a wrapper to fortran code (not coded in julia).

We have the response y as one binary vector, but we need a matrix with 2 columns, one where 1 represents success, and the other were 1 represents failure.

```
using GLMNet, Distributions
y[y.==-1] = 0
y1 = Bool.(y)
y2 = .!y1
yy = [y1 y2]
path = @time glmnet(X, convert(Matrix{Float64}, yy), Binomial())
path.betas[:, 1]
cv = @time glmnetcv(X, yy, Binomial())
#11.709583 seconds (299.73 k allocations: 623.351 MiB, 1.03% gc time)
#Logistic GLMNet Cross Validation
#100 models for 10000 predictors in 10 folds
#Best   0.079 (mean loss 1.380, std 0.004)
```

With the best $\lambda = 0.062$, we have 19 covariates in the model. The `GLMNet` method again fits 100 models and choose the best lambda.

Now we want to count how many betas different than zero in each model.

```
[sum(path.betas[:,i].!=0)/size(path.betas,1) for i in 1:size(path.betas,2)]
[sum(path.betas[:,i].!=0) for i in 1:size(path.betas,2)]
```

Choice of λ

We will use a cross validation technique to choose the best lambda. We will only do this for the subset data, and use the same λ for all the later analysis. We get a best λ of 0.062 from `GLMNet` package (19 covariates). We will do our own pseudo cross validation as well, see `choose-lambda.jl` file: we get a best $\lambda = 0.000893361$ (427 covariates).

First, we notice that $\lambda = 1.0$ is too restrictive already (we get all zeros). Actually, with `GLMNet`, we get all $\beta = 0$ even with $\lambda = 0.089$ (probably chosen as the best λ because if all responses are predicted as 0, then you get the best prediction score (more responses must be 0, than 1: yes, 260 vs 259)).

We can compare `SparseRegression` with the best λ:

```
s = SModel(X,y, LogitDistLoss(), L1Penalty(),fill(0.000893361,size(X,2))) ## need to c
@time learn!(s, strategy(ProxGrad(s,0.001), MaxIter(10000), Converged(coef)))
#find(x -> x != 0,s. ) ##3016 SNPs for  =0.1
#find(x -> x < 0,s. ) ## 1401 SNPs
```

Finally, let's check if we can fit Lasso with only one λ:

```
#f3=@time fit(LassoPath,X,y,Bernoulli(),LogitLink(), =[0.000893361],n =1)
#ERROR: maximum number of coefficients 1038 exceeded at   = 0.000893361 ( j=0.0008933
```

There is a bug! So, we will have to run all models for future cases. I opened a github issue here

Confounding covariates

Besides the data matrix X with the genotype data, we might have other covariates that you want to include in the model to adjust for their effects: age, sex, medical or environmental information. The challenge in this case is that we do not want to penalize the coefficients associated with these confounding covariates.

One “dirty” trick is to fit a logistic regression only with the confounding covariates, and then fit a LASSO linear regression with the residuals as response and the genotype data as the data matrix. The downside is that there is some controversy to using the residuals of logistic regression (references?). At the moment, we will not include confounding covariates.

The good news is that the `SparseRegression` package allows a different λ per covariate, so we can choose a $\lambda = 0$ for the covariates that we do not want to penalize.

Data types

One question is whether we can use the `SnpArray` as input for the Lasso functions, instead of converting to matrix (which is heavier). Apparently not:

```
@show typeof(chr22sub)
#obs = Obs(chr22sub, y)
#ERROR: TypeError: Type: in T, expected T<:Number, got Type{Tuple{Bool,Bool}}
#f3=@time fit(LassoPath,chr22sub,y,Bernoulli(),LogitLink())
#ERROR: MethodError: no method matching fit(::Type{Lasso.LassoPath}, ::SnpArrays.SnpA
```

See github issue and file: `bugs/sparsereg-bug.md`.

Using IHT.jl

There is much debate around the use of the lasso penalty. People suggest other penalties because lasso is supposed to impose sparsity and shrinkage on β, and most of the times, we only want sparsity. The shrinkage imposed by lasso causes an increase on false positives (see Lee et al, 2016).

The julia package `IHT.jl` fit a penalized regression with “iterative hard thresholding” penalty, which involves restricting the number of betas to a certain value k: $||\beta||_0 \leq k$.

Kevin Keys (the author of `IHT.jl`) comments that “But I could not ensure stability in crossvalidation with logistic IHT despite my best efforts. It is still an open problem.” So, we will not attempt to use the cross validation step. They use as default $\lambda = \sqrt{\log(10000)/519}$.

Needed to run in this order to make it work:

```
#Pkg.clone("https://github.com/klkeys/PLINK.jl.git")
#Pkg.clone("https://github.com/klkeys/RegressionTools.jl.git")
#Pkg.clone("https://github.com/klkeys/IHT.jl.git")
```

```
using IHT
@show sqrt(log(10000)/519)
```

```
sqrt(log(10000) / 519) = 0.13321531654646374
```

```
output = @time L0_log(X, y, 1)
```

```
0.009871 seconds (395 allocations: 1.161 MiB)
```

```
@show output
```

```
output = IHT results:

Compute time (sec):   0.0098
Final loss:           0.0687335
Iterations:           4
IHT estimated 1 nonzero coefficients.
1×2 DataFrames.DataFrame
  Row   Predictor   Estimated_


  1     8611        -2.7334e14
```

Now, the SNP number 8611 in the subset is the first one selected. This does not match what we saw with `Lasso.jl`, and the β coefficient differs.

Sadly, we cannot run `L0_log` with any $k \geq 3$, because we get the error `ERROR: BoundsError: attempt to access 519×3 Array{Float64,2} at index [Base.OneTo(519), Base.OneTo(4)]`. I opened a github issue here. This bug was fixed already.

From Kevin Keys (authot of `IHT.jl`, see his email in `email-log.md`): "You will see that logistic IHT really suffers to recover the true model. A rule of thumb is that you should have ~20 samples per true nonzero predictor. This toy scenario deliberately pushes that boundary. If you try, say, n = 1000 and p = 2500 and k = 5, then IHT can recover indices of for "large" effect sizes but not "small" ones. I asked about the model generation because many genetic association tests are not truly sparse; they actually have many nonzero effects of very small size. With a continuous outcome IHT can usually approximate the underlying model pretty well despite the tiny effects. The jump from continuous variables to binary ones induces a dramatic information loss, which makes model estimation with small effects much harder."

Finally, we try to run cross validation with `IHT.jl`: ~~{.julia} #cvout = @time cv_log(X,y) #cvout = @time cv_log(X,y,tol=0.01, tolG=0~~

Post selection

We are using the R package from the Lee et al (2016) paper `selectiveInference`. See the package here.

```
using RCall
R"""
library(selectiveInference)
#logistic model
set.seed(43)
n = 50
```

```
p = 10
sigma = 1

x = matrix(rnorm(n*p),n,p)
x=scale(x,TRUE,TRUE)

beta = c(3,2,rep(0,p-2))
y = x%*%beta + sigma*rnorm(n)
y=1*(y>mean(y))
# first run glmnet
gfit = glmnet(x,y,standardize=FALSE,family="binomial")

# extract coef for a given lambda; note the 1/n factor!
# (and here  we DO  include the intercept term)
lambda = .8
beta_hat = coef(gfit, x=x, y=y, s=lambda/n, exact=TRUE)

# compute fixed lambda p-values and selection intervals
out = fixedLassoInf(x,y,beta_hat,lambda,family="binomial")
out
"""
```

Not sure if we want to use the R function, or if we want to create our own julia function. First let's see which type of beta argument is needed:

```
R"""
out2 = fixedLassoInf(x,y,beta_hat[1:11],lambda,family="binomial")
"""
```

So, apparently `x,y,beta` are just vectors. So, we can create our own julia function. See file `selective-inference.jl` ingithub.