Snpnet Vignette

Junyang Qian and Trevor Hastie

2020-09-24

Introduction

Snpnet is a package that is used to fit the lasso on big genomics data. We assume the data are stored in .pgen/.pvar/.psam format by the PLINK library. The potential training/validation split can be specified with a separate column in the phenotype file.

The most essential parameters in the core function `snpnet` include:

- `genotype.pfile`: the PLINK 2.0 pgen file that contains genotype. We assume the existence of genotype.pfile.{pgen,pvar.zst,psam}.
- `phenotype.file`: the path of the file that contains the phenotype values and can be read as a table.
- `phenotype`: the name of the phenotype. Must be the same as the corresponding column name in the phenotype file.
- `covariates`: a character vector containing the names of the covariates included in the lasso fitting, whose coefficients will not be penalized. The names must exist in the column names of the phenotype file.
- `family`: the type of the phenotype: "gaussian", "binomial" or "cox". If not provided or NULL, it will be detected based on the number of levels in the response.
- `alpha`: the elastic-net mixing parameter, where the penalty is defined as $\alpha \cdot \|\beta\|_1 + (1-\alpha) \cdot \|\beta\|_2^2/2$. `alpha = 1` corresponds to the lasso penalty, while `alpha = 0` corresponds to the ridge penalty.
- `split.col`: the column name in the phenotype file that specifies the membership of individuals to the training or the validation set. The individuals marked as `"train"` and `"val"` will be treated as the training and validation set, respectively. When specified, the model performance is evaluated on both the training and the validation sets.
- `status.col`: the column name for the status column for Cox proportional hazards model. When running the Cox model, the specified column must exist in the phenotype file.
- `mem`: Memory (MB) available for the program. It tells PLINK 2.0 the amount of memory it can harness for the computation. IMPORTANT if using a job scheduler.

Some additional important parameters for model building include:

- `nlambda`: the number of lambda values on the solution path.
- `lambda.min.ratio`: the ratio of the minimum lambda considered versus the maximum lambda that makes all penalized coefficients zero.
- `p.factor`: a named vector of separate penalty factors applied to each coefficient. This is a number that multiplies lambda to allow different shrinkage. If not provided, default is 1 for all variables. Otherwise should be complete and positive for all variables.

The other parameters can be specified in a config list object, such as `missing.rate`, `MAF.thresh`, `nCores`, `num.snps.batch` (batch size M of the BASIL algorithm), `save` (whether to save intermediate results), `results.dir`, `prevIter` (when starting from the middle), `use.glmnetPlus` and `glmnet.thresh` (convergence threshold). More details can be seen in the function documentation. In particular, If we want to recover results and continue the procedure from a previous job, we should have `save = TRUE` and specify `prevIter` with the index of the last successful (and saved) iteration.

Snpnet depends on two other programs **plink2** and **zstdcat**. If they are not already on the system serach path, it is important to specify their locations in the `configs` object and pass it to `snpnet`.

```
configs <- list(
  # results.dir = "PATH/TO/SAVE/DIR",  # needed when saving intermediate results
  # save = TRUE,  # save intermediate results per iteration (default FALSE)
  # nCores = 16,  # number of cores available (default 1)
  # niter = 100,  # max number of iterations (default 50)
  # prevIter = 15,  # if we want to start from some iteration saved in results.dir
  # use.glmnetPlus = TRUE,  # recommended for faster computation
  # early.stopping = FALSE,  # whether to stop based on validation performance (default TRUE)
  plink2.path = "plink2",   # path to plink2 program
  zstdcat.path = "zstdcat"  # path to zstdcat program
)
# check if the provided paths are valid
for (name in names(configs)) {
  tryCatch(system(paste(configs[[name]], "-h"), ignore.stdout = T),
    condition = function(e) cat("Please add", configs[[name]], "to PATH, or modify the path in the confi
  )
}
```

A Simple Example

We demonstrate a simple lasso example first.

```
library(snpnet)
```

```
genotype.pfile <- file.path(system.file("extdata", package = "snpnet"), "sample")
phenotype.file <- system.file("extdata", "sample.phe", package = "snpnet")
phenotype <- "QPHE"
covariates <- c("age", "sex", paste0("PC", 1:10))

fit_snpnet <- snpnet(
  genotype.pfile = genotype.pfile,
  phenotype.file = phenotype.file,
  phenotype = phenotype,
  covariates = covariates,
  # split.col = "split",  # split column name in phenotype.file with train/val/test labels
  # mem = 128000,  # amount of memory available (MB), recommended
  configs = configs
)  # we hide the intermediate messages
```

The intercept and coefficients can be extracted by `fit_snpnet$a0` and `fit_snpnet$beta`. It also saves the evaluation metric, which by default is R^2 for the Gaussian family.

```
fit_snpnet$metric.train
#>   [1] 0.2089871 0.2104402 0.2129640 0.2163671 0.2203983 0.2245927 0.2297627
#>   [8] 0.2358391 0.2423690 0.2491165 0.2566939 0.2652310 0.2730096 0.2813346
#>  [15] 0.2910509 0.3017561 0.3136223 0.3265565 0.3395117 0.3528559 0.3672129
#>  [22] 0.3826513 0.3984702 0.4149482 0.4320622 0.4499550 0.4681395 0.4868573
#>  [29] 0.5063165 0.5255936 0.5445475 0.5634130 0.5826291 0.6016155 0.6202583
#>  [36] 0.6387452 0.6569201 0.6746515 0.6916680 0.7083095 0.7246041 0.7404483
#>  [43] 0.7556963 0.7702659 0.7841292 0.7973795 0.8099013 0.8218621 0.8332552
#>  [50] 0.8441034 0.8545183 0.8643099 0.8735285 0.8822317 0.8904601 0.8981543
#>  [57] 0.9054622 0.9123111 0.9187217 0.9247878 0.9303812 0.9355430 0.9402540
```

```
#>  [64] 0.9446735 0.9488678 0.9527262 0.9562351 0.9595308 0.9626171 0.9655038
#>  [71] 0.9681871 0.9706793 0.9729908 0.9751322 0.9771209 0.9789609 0.9806068
#>  [78] 0.9821547 0.9835932 0.9849168 0.9861369 0.9872630 0.9883033 0.9892658
#>  [85] 0.9901059 0.9909487 0.9916699 0.9923515 0.9929845 0.9935694 0.9940606
#>  [92] 0.9945264 0.9949655 0.9953755 0.9957552 0.9961060 0.9964293 0.9967272
#>  [99] 0.9970012 0.9972531
```

We can make prediction with the fitted object `fit_snpnet`. For example,

```
pred_snpnet <- predict_snpnet(
  fit = fit_snpnet,
  new_genotype_file = genotype.pfile,
  new_phenotype_file = phenotype.file,
  phenotype = phenotype,
  covariate_names = covariates,
  split_col = "split",
  split_name = c("train", "val"),  # can also include "test" if such samples are available in the pheno
  configs = configs)
```

We can find out both the predicted values from the `prediction` field and the evaluation metrics from the `metric` field.

```
str(pred_snpnet$prediction)
#> List of 2
#>  $ train: num [1:1600, 1:100] -0.0942 0.1855 -0.3101 -0.0924 0.1544 ...
#>   ..- attr(*, "dimnames")=List of 2
#>   .. ..$ : chr [1:1600] "per0_per0" "per1_per1" "per2_per2" "per3_per3" ...
#>   .. ..$ : chr [1:100] "s0" "s1" "s2" "s3" ...
#>  $ val  : num [1:200, 1:100] -0.855 -0.0804 -0.838 -0.3573 0.7198 ...
#>   ..- attr(*, "dimnames")=List of 2
#>   .. ..$ : chr [1:200] "per4_per4" "per15_per15" "per20_per20" "per27_per27" ...
#>   .. ..$ : chr [1:100] "s0" "s1" "s2" "s3" ...
str(pred_snpnet$metric)
#> List of 2
#>  $ train: Named num [1:100] 0.211 0.212 0.215 0.218 0.222 ...
#>   ..- attr(*, "names")= chr [1:100] "s0" "s1" "s2" "s3" ...
#>  $ val  : Named num [1:100] 0.186 0.187 0.19 0.193 0.197 ...
#>   ..- attr(*, "names")= chr [1:100] "s0" "s1" "s2" "s3" ...
```

Lasso with Refitting

Refitting is often recommended for the lasso/elastic-net to make the most of the validation set (more than serving for tuning parameter selection). One may take the following steps:

- fit models on the training set under different parameters;
- choose the optimal parameter based on the metric (R^2/AUC) on the validation set;
- refit the model with the chosen parameter on a combined training and validation set.

We show a code example of refitting below. To do that, we will need to use the `split` column in the phenotype file.

```
fit_snpnet_train <- snpnet(
  genotype.pfile = genotype.pfile,
  phenotype.file = phenotype.file,
  phenotype = phenotype,
  covariates = covariates,
```

```
  split.col = "split",
  # mem = 128000,  # amount of memory available (MB), recommended
  configs = configs
)  # we hide the intermediate messages
```

Due to the default early stopping criterion, snpnet doesn't fit all the way to the end.

```
max_idx <- sum(!is.na(fit_snpnet_train$metric.val))
fit_snpnet_train$metric.val[1:max_idx]
#>  [1] 0.1871064 0.1876379 0.1881991 0.1900906 0.1936576 0.1977614 0.2020771
#>  [8] 0.2064773 0.2125167 0.2203595 0.2272916 0.2333822 0.2393646 0.2451911
#> [15] 0.2500949 0.2544346 0.2579107 0.2610755 0.2623503 0.2639159 0.2654058
#> [22] 0.2660344 0.2666799 0.2670676 0.2676056 0.2680150 0.2691833 0.2705949
#> [29] 0.2706501 0.2706059 0.2702432 0.2685013 0.2660019 0.2637800 0.2609096
```

To do the refitting, we have created a separate column `split_refit` in the phenotype file that merges the original `train` and `val` labels, and replaces the test labels with `val` so that snpnet will conveniently evaluate the test performance of the refit models.

```
library(data.table)
phe_tbl <- fread(phenotype.file)
table(phe_tbl$split, phe_tbl$split_refit)
#>
#>         train  val
#>   test      0  200
#>   train  1600    0
#>   val     200    0
```

We extract the exact same lambda sequence from the fit above and refit using the `split_refit` column. Note that we want to turn off `early.stopping` in this case.

```
configs[["early.stopping"]] <- FALSE
fit_snpnet_refit <- snpnet(
  genotype.pfile = genotype.pfile,
  phenotype.file = phenotype.file,
  phenotype = phenotype,
  covariates = covariates,
  split.col = "split_refit",
  lambda = fit_snpnet_train$full.lams[1:max_idx],
  configs = configs
)  # we hide the intermediate messages
```

We may take a look at the refit training and test performance.

```
fit_snpnet_refit$metric.train
#>  [1] 0.2092566 0.2092566 0.2102606 0.2125659 0.2160379 0.2199773 0.2260090
#>  [8] 0.2327893 0.2396929 0.2475833 0.2564766 0.2650072 0.2739793 0.2845658
#> [15] 0.2957047 0.3072280 0.3195407 0.3322154 0.3455897 0.3594264 0.3749953
#> [22] 0.3917710 0.4097319 0.4287484 0.4482760 0.4681127 0.4883932 0.5089945
#> [29] 0.5291372 0.5492970 0.5689396 0.5883670 0.6076702 0.6269533 0.6460628
fit_snpnet_refit$metric.val  # this is in fact the performance evaluated on the test individuals
#>  [1] 0.1956831 0.1956831 0.1966735 0.1994116 0.2044847 0.2097991 0.2164871
#>  [8] 0.2241968 0.2314194 0.2390105 0.2473809 0.2551320 0.2625772 0.2700116
#> [15] 0.2763305 0.2816834 0.2878536 0.2943366 0.3000984 0.3056831 0.3118075
#> [22] 0.3177306 0.3232243 0.3278932 0.3323033 0.3346956 0.3367966 0.3385153
#> [29] 0.3395552 0.3396960 0.3384233 0.3365555 0.3340420 0.3314546 0.3284774
```

In the end, we should extract the test performance at the λ value that achieves the best validation performance earlier. We may also output the corresponding model size.

```
opt_idx <- which.max(fit_snpnet_train$metric.val)
metric_optimal_test <- fit_snpnet_refit$metric.val[opt_idx]
size_optimal <- sum(fit_snpnet_refit$beta[[opt_idx]] != 0)
list(metric = metric_optimal_test, size = size_optimal)
#> $metric
#> [1] 0.3395552
#>
#> $size
#> [1] 314
```

Numerical Comparison with Glmnet

To compare with **glmnet**, we need to convert the genotype data into a normal R object.

```
ids <- readIDsFromPsam(paste0(genotype.pfile, '.psam'))
phe <- readPheMaster(phenotype.file, ids, "gaussian", covariates, phenotype, NULL, NULL, configs)
vars <- readRDS(system.file("extdata", "vars.rds", package = "snpnet"))
pvar <- pgenlibr::NewPvar(paste0(genotype.pfile, '.pvar.zst'))
pgen <- pgenlibr::NewPgen(paste0(genotype.pfile, '.pgen'), pvar = pvar, sample_subset = NULL)
data.X <- pgenlibr::ReadList(pgen, seq_along(vars), meanimpute=F)
colnames(data.X) <- vars
p <- ncol(data.X)
pnas <- numeric(p)
for (j in 1:p) {
  pnas[j] <- mean(is.na(data.X[, j]))
  data.X[is.na(data.X[, j]), j] <- mean(data.X[, j], na.rm = T)  # mean imputation
}
```

```
data.X <- as.matrix(cbind(age = phe$age, sex = phe$sex, phe[, paste("PC", 1:10, sep = "")], data.X))
data.y <- phe$QPHE
pfactor <- rep(1, p + 12)
pfactor[1:12] <- 0  # we don't penalize the covariates

fit_glmnet <- glmnet::glmnet(data.X, data.y, penalty.factor = pfactor, standardize = F)
```

```
# check difference of coefficients matched by the names
checkDiff <- function(x, y) {
  unames <- union(names(x), names(y))
  xf <- yf <- rep(0, length(unames))
  names(xf) <- names(yf) <- unames
  xf[match(names(x), unames)] <- x
  yf[match(names(y), unames)] <- y
  list(max = max(abs(xf-yf)), mean = mean(abs(xf-yf)))
}
```

We show the difference of the computed λ sequence and the estimated coefficients. There is small discrepancy between the two solutions within the range of convergence threshold. The gap will shrink and eventually goes to 0 if we keep on tightening the threshold.

```
max(abs(fit_snpnet$full.lams - fit_glmnet$lambda*length(pfactor)/sum(pfactor)))  # adjustment due to so
#> [1] 4.750225e-11
```

```
checkDiff(fit_snpnet$beta[[6]], fit_glmnet$beta[, 6])
#> $max
#> [1] 4.280143e-07
#>
#> $mean
#> [1] 1.75303e-10
```

More Examples

We also show two more sophisticated usage of the `snpnet` function.

```
configs[["nCores"]] <- 2
configs[["num.snps.batch"]] <- 500
fit_snpnet_ent <- snpnet(
  genotype.pfile = genotype.pfile,
  phenotype.file = phenotype.file,
  phenotype = phenotype,
  covariates = covariates,
  alpha = 0.5,  # elastic-net
  split.col = "split",  # the sample phenotype file contains a column specifying the training/validatio
  configs = configs
)
```

When `split.col` is specified, validation metric will automatically be computed. Moreover, by default, early stopping is adopted to stop the process based on the validation metric (the extent controlled by `stopping.lag` in `configs` object).

```
fit_snpnet_ent$metric.train
#>   [1] 0.2115936 0.2126226 0.2136493 0.2156094 0.2196077 0.2245383 0.2297587
#>   [8] 0.2351583 0.2419546 0.2504558 0.2604812 0.2710511 0.2817251 0.2925574
#>  [15] 0.3037298 0.3157702 0.3286286 0.3426865 0.3589287 0.3764867 0.3944012
#>  [22] 0.4131001 0.4327160 0.4532889 0.4743553 0.4952979 0.5161484 0.5370624
#>  [29] 0.5580632 0.5787104        NA        NA        NA        NA        NA
#>  [36]        NA        NA        NA        NA        NA        NA        NA
#>  [43]        NA        NA        NA        NA        NA        NA        NA
#>  [50]        NA        NA        NA        NA        NA        NA        NA
#>  [57]        NA        NA        NA        NA        NA        NA        NA
#>  [64]        NA        NA        NA        NA        NA        NA        NA
#>  [71]        NA        NA        NA        NA        NA        NA        NA
#>  [78]        NA        NA        NA        NA        NA        NA        NA
#>  [85]        NA        NA        NA        NA        NA        NA        NA
#>  [92]        NA        NA        NA        NA        NA        NA        NA
#>  [99]        NA        NA
fit_snpnet_ent$metric.val
#>   [1] 0.1871064 0.1875658 0.1880553 0.1896925 0.1928345 0.1965032 0.2004100
#>   [8] 0.2044619 0.2100379 0.2173241 0.2238545 0.2296996 0.2355830 0.2413149
#>  [15] 0.2461747 0.2506889 0.2544121 0.2577139 0.2592561 0.2610149 0.2629657
#>  [22] 0.2639808 0.2645202 0.2654656 0.2666435 0.2674386 0.2693372 0.2707239
#>  [29] 0.2705973 0.2705524        NA        NA        NA        NA        NA
#>  [36]        NA        NA        NA        NA        NA        NA        NA
#>  [43]        NA        NA        NA        NA        NA        NA        NA
#>  [50]        NA        NA        NA        NA        NA        NA        NA
#>  [57]        NA        NA        NA        NA        NA        NA        NA
#>  [64]        NA        NA        NA        NA        NA        NA        NA
```

```
#>  [71]        NA        NA        NA        NA        NA        NA        NA
#>  [78]        NA        NA        NA        NA        NA        NA        NA
#>  [85]        NA        NA        NA        NA        NA        NA        NA
#>  [92]        NA        NA        NA        NA        NA        NA        NA
#>  [99]        NA        NA
```

```
fit_snpnet_bin <- snpnet(
  genotype.pfile = genotype.pfile,
  phenotype.file = phenotype.file,
  phenotype = "BPHE",  # binary phenotype with logistic regression
  covariates = covariates,
  alpha = 0.5,
  family = "binomial",
  split.col = "split",
  configs = configs
)
```

For binary phenotypes, instead of R^2, AUC is computed.

```
fit_snpnet_bin$metric.train
#>   [1] 0.6790872 0.6802187 0.6812283 0.6822316 0.6830131 0.6840945 0.6847806
#>   [8] 0.6856089 0.6865779 0.6883658 0.6917791 0.6959206 0.7002607 0.7068997
#>  [15] 0.7145858 0.7240395 0.7354499 0.7491092 0.7632139 0.7789503        NA
#>  [22]        NA        NA        NA        NA        NA        NA        NA
#>  [29]        NA        NA        NA        NA        NA        NA        NA
#>  [36]        NA        NA        NA        NA        NA        NA        NA
#>  [43]        NA        NA        NA        NA        NA        NA        NA
#>  [50]        NA        NA        NA        NA        NA        NA        NA
#>  [57]        NA        NA        NA        NA        NA        NA        NA
#>  [64]        NA        NA        NA        NA        NA        NA        NA
#>  [71]        NA        NA        NA        NA        NA        NA        NA
#>  [78]        NA        NA        NA        NA        NA        NA        NA
#>  [85]        NA        NA        NA        NA        NA        NA        NA
#>  [92]        NA        NA        NA        NA        NA        NA        NA
#>  [99]        NA        NA
fit_snpnet_bin$metric.val
#>   [1] 0.6584274 0.6590377 0.6596481 0.6606652 0.6605635 0.6611738 0.6614790
#>   [8] 0.6623945 0.6626996 0.6624962 0.6626996 0.6633099 0.6648357 0.6662598
#>  [15] 0.6674804 0.6668701 0.6658529 0.6662598 0.6652426 0.6623945        NA
#>  [22]        NA        NA        NA        NA        NA        NA        NA
#>  [29]        NA        NA        NA        NA        NA        NA        NA
#>  [36]        NA        NA        NA        NA        NA        NA        NA
#>  [43]        NA        NA        NA        NA        NA        NA        NA
#>  [50]        NA        NA        NA        NA        NA        NA        NA
#>  [57]        NA        NA        NA        NA        NA        NA        NA
#>  [64]        NA        NA        NA        NA        NA        NA        NA
#>  [71]        NA        NA        NA        NA        NA        NA        NA
#>  [78]        NA        NA        NA        NA        NA        NA        NA
#>  [85]        NA        NA        NA        NA        NA        NA        NA
#>  [92]        NA        NA        NA        NA        NA        NA        NA
#>  [99]        NA        NA
```