

 Open access • Posted Content • DOI:10.1101/2021.02.14.431030

Fast Numerical Optimization for Genome Sequencing Data in Population Biobanks

— [Source link](#) 

Ruilin Li, Christopher C. Chang, Yosuke Tanigawa, Balasubramanian Narasimhan ...+3 more authors

Institutions: Stanford University

Published on: 16 Feb 2021 - bioRxiv (Cold Spring Harbor Laboratory)

Topics: Sparse matrix, Lasso (statistics), Matrix (mathematics), Population and Proximal Gradient Methods

Related papers:

- [Fast Numerical Optimization for Genome Sequencing Data in Population Biobanks.](#)
- [GenoGAM 2.0: scalable and efficient implementation of genome-wide generalized additive models for gigabase-scale genomes](#)
- [Validating Paired-End Read Alignments in Sequence Graphs](#)
- [Using Genome Graph Topology to Guide Annotation Matrix Sparsification](#)
- [Communication-Efficient Jaccard Similarity for High-Performance Distributed Genome Comparisons](#)

Share this paper:    

View more about this paper here: <https://typeset.io/papers/fast-numerical-optimization-for-genome-sequencing-data-in-46rj8xsoen>

Fast Numerical Optimization for Genome Sequencing Data in Population Biobanks

Ruilin Li^{*1}, Christopher Chang², Yosuke Tanigawa³, Balasubramanian Narasimhan^{3,4}, Trevor Hastie^{3,4}, Robert Tibshirani^{3,4}, and Manuel A. Rivas^{†3}

¹Institute for Computational and Mathematical Engineering, Stanford University

²Grail, Inc.

³Department of Biomedical Data Science, Stanford University

⁴Department of Statistics, Stanford University

Abstract

We develop two efficient solvers for optimization problems arising from large-scale regularized regressions on millions of genetic variants sequenced from hundreds of thousands of individuals. These genetic variants are encoded by the values in the set $\{0, 1, 2, \text{NA}\}$. We take advantage of this fact and use two bits to represent each entry in a genetic matrix, which reduces memory requirement by a factor of 32 compared to a double precision floating point representation. Using this representation, we implemented an iteratively reweighted least square algorithm to solve Lasso regressions on genetic matrices, which we name `snpnet-2.0`. When the dataset contains many rare variants, the predictors can be encoded in a sparse matrix. We utilize the sparsity in the predictor matrix to further reduce memory requirement and computational speed. Our sparse genetic matrix implementation uses both the compact 2-bit representation and a simplified version of compressed sparse block format so that matrix-vector multiplications can be effectively parallelized on multiple CPU cores. To demonstrate the effectiveness of this representation, we implement an accelerated proximal gradient method to solve group Lasso on these sparse genetic matrices. This solver is named `sparse-snpnet`, and will also be included as part of `snpnet` R package. Our implementation is able to solve group Lasso problems on sparse genetic matrices with more than 1,000,000 columns and almost 100,000 rows within 10 minutes and using less than 32GB of memory.

1 Introduction

Constantly growing biobanks have provided scientists and researchers with unprecedented opportunities to understand the genetics of human phenotypes. One component is to predict phenotypes of an individual using genetic data. However, datasets of increasing size also pose computational challenges for this task. On the statistics side, genetic datasets are usually high-dimensional, meaning the number of genetic variants is larger than the number of sequenced individuals. High dimensional statistics have been studied for more than two decades with well understood solutions. One such

^{*}Corresponding author: ruilinli@stanford.edu

[†]Corresponding author: mrivas@stanford.edu

34 solution is to “bet on sparsity”: the assumption that only a small subset of variables are associated
35 with the response. The sparsity assumption is usually embodied through an objective function that
36 encourages sparsity in the solution. Well known examples include the Lasso and the group Lasso.
37 On the computation side, a statistical estimator that describes the relationship between the genetic
38 variants and the response of interest are often obtained by optimizing an objective function involving
39 the genetic matrix. While off-the-shelf solvers may exist for these optimization problems, they are
40 usually not optimal for genetics data. First, these general purpose solvers require loading a floating
41 point predictor matrix in memory before optimization can be done. This can demand a very large
42 amount of memory for biobank scale data. For example, loading a matrix with 200,000 rows and
43 1,000,000 columns as double precision floating point numbers takes 1.6 terabytes, much larger than
44 the RAM size of most machines. In particular, they do not exploit the fact that genetic variants can
45 take on only four possible values. Secondly, many of these solvers do not fully utilize modern hard-
46 ware features such as multi-core processors, which leaves lots of performance on the table. Thirdly, a
47 large number of variants in exome and whole genome sequencing data are rare variants. If a variant
48 is encoded as the number of copies of the minor allele, then the corresponding genetic matrix is
49 sparse. In the UK Biobank’s exome sequencing data (Szustakowski et al. 2020), more than 99% of
50 the variants in the targeted regions have minor allele frequency less than 1%. As a result, more than
51 98% of the entries of the corresponding genetic matrix are zero. The sparsity in the predictor matrix
52 can potentially be exploited to improve both memory requirements and computational speed.

53 The main result of this work is an extremely efficient regularized regression solver for problems
54 with sparse genetic predictors, named `sparse-snpnet`. The main features of this solver are the
55 following:

- 56 1. A compact, two bits representation of genetic variants based on PLINK2’s (Chang et al. 2015)
57 pgen files.
- 58 2. Good scalability to multi-core processors.
- 59 3. A simplified version of the compressed sparse block format so that arithmetic operations on
60 the genetic matrices are more amenable to parallelism.

61 In addition, we provide an extension to the popular R package `glmnet` (Friedman et al. 2010, Simon
62 et al. 2011) specifically for Lasso problems involving genetic matrices. This extension exploits the
63 compact representation and is multi-threaded, but does not assume sparsity of the input genetic
64 matrix. We incorporate this solver to the screening framework (Qian et al. 2020) in `snpnet` and
65 name it `snpnet-2.0`. Both solvers are implemented in C++ and wrapped as part of the R package
66 `snpnet`, which is available at <https://github.com/rivas-lab/snpnet/tree/compact>. We refer
67 the readers to section 5 for comparisons between these two methods.

68 2 Results

69 2.1 Optimization Algorithm

70 We focus on regularized regression problems whose objective functions are in the following form:

$$f(\beta) = h(X\beta) + \lambda R(\beta) \quad (1)$$

71 where $X \in \{0, 1, 2, \text{NA}\}^{n \times d}$ is a genetic matrix, $\beta \in \mathbb{R}^d$ is the parameter vector, $h : \mathbb{R}^n \mapsto \mathbb{R}$
72 is usually the negative log-likelihood function of a generalized linear model (Hastie & Tibshirani

1986), and is always assumed to be smooth and convex. We have omitted the dependence of h on the response vector to simplify the notation. $R: \mathbb{R}^d \mapsto \mathbb{R}_+$ is a regularization function, and $\lambda \in \mathbb{R}_+$ represents the strength of regularization. Here are some examples of h :

1. Linear regression: $h(X\beta) = \frac{1}{n} \|y - X\beta\|_2^2$ for a response vector $y \in \mathbb{R}^n$.
2. Logistic regression: write $\eta = X\beta$, $h(X\beta) = h(\eta) = \sum_{i=1}^n y_i \log(1 + e^{\eta_i}) + (1 - y_i) \log(1 + e^{-\eta_i})$ for a binary response $y \in \{0, 1\}^n$.
3. Cox regression (Cox 1972): Write $\eta = X\beta$, $h(X\beta) = h(\eta) = \sum_{i=1}^n O_i \left[-\eta_i + \log \left(\sum_{y_j \geq y_i} e^{\eta_j} \right) \right]$ for a survival time vector $y \in \mathbb{R}_+^n$ and an event indicator $O \in \{0, 1\}^n$.

The regularization function is usually a seminorm but not always. Some examples are:

1. Lasso (Tibshirani 1996): $R(\beta) = \|\beta\|_1 = \sum_{i=1}^n |\beta_i|$.
2. Elastic net (Zou & Hastie 2005): $R(\beta) = \|\beta\|_1 + \alpha \|\beta\|_2^2$ for some $\alpha > 0$.
3. Group Lasso (Yuan & Lin 2006): $R(\beta) = \sum_{g \in \mathcal{G}} \|\beta_g\|_2$, where $g \in \mathcal{G}$, $g \subseteq \{1, 2, \dots, d\}$ represents a subset of variables.

To minimize (1), we apply an accelerated proximal gradient descent algorithm (Nesterov 1983, Daubechies et al. 2004, Beck & Teboulle 2009) with backtracking line search to determine the step size. This algorithm has fast convergence rate, essentially no tuning parameter, and is particularly suitable for the simple regularization functions that we use. In short, this algorithm alternates between a gradient descent step that decreases the value of $h(X\beta)$, and a proximal step that ensures that the regularization term is not too large. Note that the gradient here refers to the gradient of $h(X\beta)$ with respect to β . The regularization function is usually not differentiable at 0. The proximal operator is defined as:

$$\text{prox}_{R,t}(\beta) := \arg \min_{z \in \mathbb{R}^d} \frac{1}{2t} \|z - \beta\|_2^2 + \lambda R(z). \quad (2)$$

When the regularization function is one of the examples above, the corresponding proximal operator have explicit expression. We summarize this process in the pseudo-code in algorithm 1.

We observe that in this algorithm, the only operations that involve the predictor matrix X are matrix-vector multiplications $X\beta$ and $X^T r$, where $r = \nabla h(X\beta) \in \mathbb{R}^d$. When X is dense, these two operations are also the most computationally intensive ones in this algorithm, having complexity $\mathcal{O}(nd)$, whereas all other operations are either $\mathcal{O}(n)$ or $\mathcal{O}(d)$. This, together with the need to reduce the amount of memory required to load X , motivate a more compact and efficient representation of the genetic predictor matrix X .

2.2 Sparse Genotype Matrix Representation

In this section we describe the format we use to represent sparse genetic matrices. First of all, we pack each entries in the matrix to two bits. 0, 1, 2 and NA are represented by 00, 01, 10 and 11, respectively. The compressed sparse column (CSC) format is a popular way to store a sparse matrix. The PLINK 2.0 library (Chang et al. 2015) provides functions that make loading a genetic matrix into this format straightforward. Under CSC, a matrix with n rows, d columns and nnz non-zero entries are represented by three arrays:

- 110 1. A column pointer array `col_ptr` of size $d + 1$.
- 111 2. A row index array `row_idx` of size nnz .
- 112 3. A value array `val` of size nnz .

113 For each column $j \in \{1, 2, \dots, d\}$, the non-zero entries in that column are stored from the `col_ptr[j]`th
114 (inclusive) entry to the `(col_ptr[j+1] - 1)`th entry of `row_idx` and `val`, where `row_idx` stores the
115 row index of the non-zero entry and `val` stores the non-zero value. Figure 1 provides an illustration
116 of a sparse genetic matrix under CSC format.

117 When a sparse matrix is stored in CSC format, accessing a particular column is simple. As a
118 result, one can trivially parallelize the computation of $X^T r$. For example, thread j can compute
119 the inner product of the j th column of X and r and write to the j th entry of the output without
120 interfering with other threads. However, same thing can't be said about $X\beta$. We can't directly
121 access a row of X stored in CSC format, so there is no easy way to make each thread compute the
122 inner product between β and a row of X . Another way is to, say, have thread j add β_j times the
123 j th column of X to the output, but doing this in parallel leads to data race. Alternatively, one can
124 store X in the compressed sparse row format, which makes parallelizing $X\beta$ easy but $X^T r$ difficult.

125 Our implementation uses a simplified version of the compressed sparse block (CSB) format
126 proposed in Buluç et al. (2009). In this format, the sparse matrix is partitioned into a grid of
127 smaller, rectangular sub-matrices with same dimensions, which are referred to as blocks. When
128 partitioned to B blocks, a matrix with n rows, d columns, and nnz non-zero entries are represented
129 by four arrays:

- 130 1. A block pointer array `blk_ptr` of size $B + 1$.
- 131 2. A row index array `row_idx` of size nnz .
- 132 3. A column index array `col_idx` of size nnz .
- 133 4. A value array `val` of size nnz .

134 In this representation, non-zero entries in a block (as oppose to those in a column in CSC format)
135 are stored contiguously. The row indices, column indices, and values of the non-zero values in a
136 block $b \in \{1, 2, \dots, B\}$ are stored in `row_idx`, `col_idx`, and `val`, starting at the index `blk_ptr[b]`
137 and ending at the index `blk_ptr[b+1]-1`. In the original CSB paper the non-zero elements in each
138 block has a Z-Morton ordering, while the blocks can have any order. In our simplified version we
139 store the blocks and the non-zero elements within a block in a column major fashion. Figure 2
140 provides an illustration.

141 Under this representation accessing a block in the sparse matrix is easy. As a result, parallelizing
142 both $X\beta$ and $X^T r$ are straightforward. For example if X is the matrix in figure 2, then to compute
143 $X\beta$ we can have thread 1 compute the inner product of the first three rows of X and β , thread 2
144 compute the inner product of row 4-6 and β , etc. Similarly, to compute $X^T r$ thread b will compute
145 the inner product between r and the columns $3b - 2, 3b - 1, 3b$ in X for $b \in \{1, 2, 3\}$.

146 Since our implementation uses 2 bits to store a matrix entry and 32 bits to store each index, the
147 genetic CSB format (compared to a dense representation) will only save memory when the matrix is
148 sufficiently sparse (approximately $< 3\%$ of entries are non-zero). While there are many techniques
149 to reduce the number of bits needed to represent the indices (such as storing the indices relative

Algorithm 1: Accelerated Proximal Gradient Method for (1)

Set line search parameter $\gamma > 1$;
Initialize the parameter vector $\beta^{(0)} = 0$;
Set iteration count $i = 0$; Set initial step-size $t = 1$; Set Nesterov weights $w_0, w_1 = 1$;
while β has not converged **do**
 Nesterov acceleration:
 $w_1 \leftarrow (1 + \sqrt{1 + 4w_0^2})/2$;
 $\beta \leftarrow \beta^{(i)} + (w_0 - 1)(\beta^{(i)} - \beta^{(i-1)})/w_1$; $w_0 \leftarrow w_1$;
 Compute the gradient $g = X^T \nabla h(X\beta)$;
 Start backtracking line search:
 repeat
 $\beta^{(i+0.5)} \leftarrow \beta - tX^T \nabla h(X\beta)$;
 Apply proximal step: $\beta^{(i+1)} \leftarrow \text{prox}_{R,t}(\beta^{(i+0.5)})$;
 if $h(X\beta^{i+1}) \leq h(X\beta) + (\beta^{i+1} - \beta)^T g + \|\beta^{i+1} - \beta\|_2^2/(2t)$ **then**
 | **break**;
 end
 Shrink step size $t \leftarrow t/\gamma$;
 until the break condition above is satisfied;
 Accept the iterate β^{i+1} ;
 $i \leftarrow i + 1$;
 Check convergence based on objective value change or parameter change.
end
return β^{i+1}

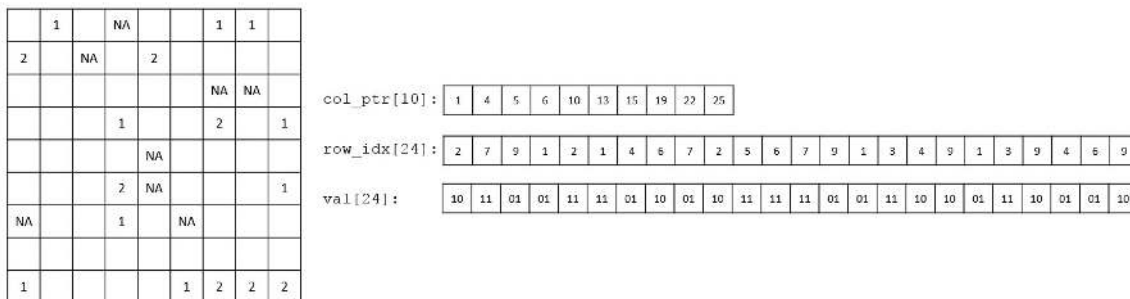


Figure 1: A sparse genetic matrix represented in the compressed sparse column format. The zero entries on the left are omitted.

150 to the start of the block, differential encoding, bit packing), the current version of our software
151 does not implement these techniques. For variants with high minor allele frequency, we store the
152 corresponding columns in dense format and keep track of their column indices. We also keep an
153 additional array of length d to store the mean imputation of the missing values in X .

154 3 Benchmarks

155 3.1 Performance on Dense Matrix-Vector Multiplications

156 In the first benchmark we evaluate the performance improvement when the predictor matrix uses
157 the 2-bit compact representation, but not the sparse format described in the last section. The
158 genetics data are dense and simulated through the `plink2 --dummy` command. The matrix have
159 $n = 200,000$ rows and $d = 30,000$ columns with approximately 5% entries NAs. In Figure 3 we
160 show the relative speedup of the compact matrix as a function of the number of threads used. The
161 baseline is R's builtin matrix-vector multiplication functions for double precision matrices (a basic,
162 single threaded BLAS implementation). The numbers reported are based on the median wall time
163 of 10 runs. Figure 3 demonstrates a more than 20-35 folds of speedup over the baseline, and good
164 performance scalability in the number of threads for up to almost 20 threads. Unless otherwise
165 specified, all computational experiments in this paper are done on an Intel Xeon Gold 6258R. For
166 most of our applications 16 out of the 28 CPU cores that comes with this CPU are used.

167 3.2 Performance on Solving Large-scale Lasso Problems

168 In the second benchmark we compare the performance of the 2-bit compact genetic matrix represen-
169 tation when it is incorporated in the software packages `glmnet` and `snpnet` to solve large-scale Lasso
170 problems where the predictors are mostly Single-nucleotide polymorphisms (SNPs) (with perhaps a
171 few real-valued covariates such as age). As mentioned in the introduction, we call this implemen-
172 tation `snpnet-2.0`. The main performance gain comes from these factors (the readers can refer to
173 Qian et al. (2020), Li et al. (2020) for the definitions of some terms below):

- 174 1. `glmnet` fitting is based on the iteratively reweighted least square (IRLS) algorithm, where the
175 main bottleneck is computing inner-products between columns of X and a real-valued vector.
176 This is done using the 2-bit compact representation and is multi-threaded in `snpnet-2.0`.
- 177 2. `snpnet` uses a screening procedure named the batch screening iterative Lasso (BASIL). At
178 each BASIL iteration a different set of predictors is used to fit a model. Using the compact
179 representation reduces the amount of memory traffic needed.
- 180 3. `snpnet-2.0` uses reduced precision floating point numbers (`float32` instead of `float64`) to
181 do KKT checking.
- 182 4. Warm start support, as well as more relaxed convergence criteria, for binomial model and Cox
183 model.

184 Since `snpnet-2.0` also uses more relaxed convergence criteria than the previous version, it is not
185 very fair to just compare the speed of these packages. As a result we provide both the time spent
186 as well as the test set prediction performance. The data used here are a combination of directly

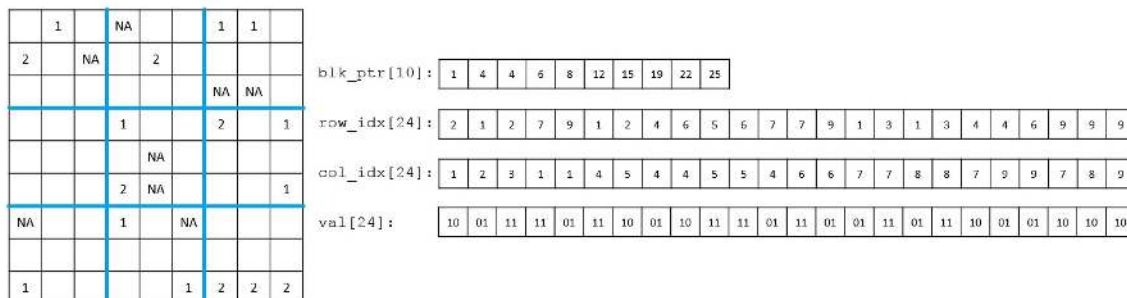


Figure 2: A sparse genetic matrix represented in the compressed sparse block format. The zero entries on the left are omitted. The boundaries of each block are highlighted in blue. The non-zero elements in each block are stored contiguously in an top-to-bottom, left-to-right order. The blocks are also in an top-to-bottom, left-to-right order

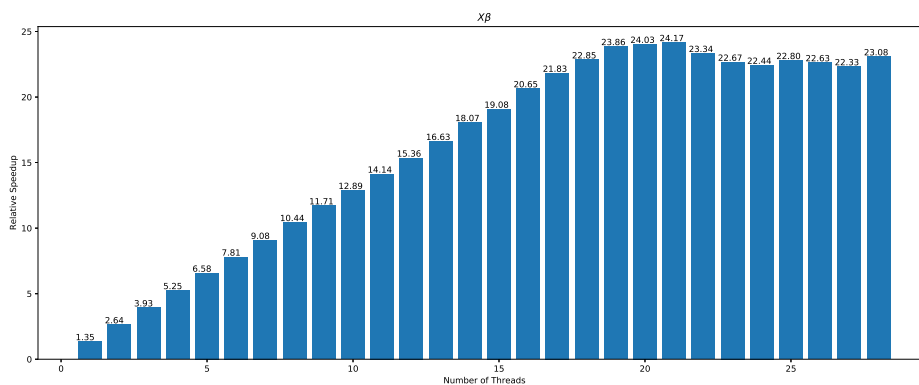


Figure 3: A bar plot demonstrating the relative speedup in computing $X\beta$ when the compact representation is used. The baseline is the R's builtin matrix-vector multiplication function for double precision matrices. The horizontal axis is the number of threads. The vertical axis is the ratio of between time spent in the baseline and the time spent with the compact representation. The baseline for $X\beta$ is 9.8 seconds.

187 genotyped variants (release version 2 of Sudlow et al. (2015)), the imputed allelotypes in human
188 leukocyte antigen allelotypes (Venkataraman et al. 2020), and copy number variations described in
189 Aguirre et al. (2019), resulting in a genotype matrix of 1,080,968 variants, as described in Sinnott-
190 Armstrong et al. (2021). The study population consists of 337,129 unrelated participants of white
191 British ancestry described in DeBoever et al. (2018). We randomly select 70% of the participants
192 as the training set, 10% as the validation set, and 20% as the test set. The results are summarized
193 in table 3. The table shows that `snpnet-2.0` achieves significant speedup over the old version while
194 having very similar test set prediction performance. We note that for standing height, more than
195 80,000 variants are selected to fit the model. Since the predictor matrix is duplicated in its fitting
196 process, the old version of `snpnet` requires more than 400GB to successfully finish. On the other
197 hand, 32GB of memory is sufficient for `snpnet-2.0`.

| | Time (new) | Time (old) | Test metric (new) | Test metric (old) |
|--------------------------|------------|------------|-------------------|-------------------|
| High cholesterol (B) | 21.9 | 109.8 | 0.72533 | 0.72531 |
| Asthma (B) | 21.7 | 130.0 | 0.61609 | 0.61608 |
| Standing Height (Q) | 99.9 | 405.8* | 0.71096 | 0.71100 |
| BMI (Q) | 51.5 | 208.3* | 0.11408 | 0.11412 |
| Other hypothyroidism (S) | 13.5 | 71.5 | 0.75194 | 0.75205 |
| Thyrotoxicosis (S) | 3.6 | 10.0 | 0.71020 | 0.71021 |

Table 1: Speed comparison between the old version of `snpnet` and `snpnet-2.0`. Time is measured in **minutes**. (B) indicates the response is binary, (Q) indicates the response is quantitative, and (S) indicates that the response is a survival time. For binary response, the test metric is the area under the ROC curve (AUC). For quantitative response, the metric is the R-squared. For survival response, the metric is the C-index. *The machine we used for most of the applications here has a dual-socket architecture, each having around 400 GB of local memory. The memory requirements by the old version of `snpnet` for both standing height and BMI exceeds the capacity of the local memory of a single socket in this machine. As a result, we ran these two experiments on an Intel Xeon Gold 6130 (also 16 cores) machine with more memory.

198 3.3 Performance of the Sparse Format

199 In the third benchmark we evaluate the performance improvement when the genetic predictors
200 make use of both the 2-bit compact representation, and the sparse representation described in the
201 last section. In this case we use real exome data from the UK Biobank. The raw data has 200,643
202 individuals and 17,777,950 variants. For this benchmark we only use variants with least 3 individuals
203 having the minor allele and with missing rate at most 10%. The result is a sparse genetic matrix
204 with 200,643 rows and 7,462,671 columns. For our application in the next section the number
205 of variants used to fit models will be smaller since the training set will be a subset of the entire
206 population in this data. On average each column of this matrix has 1399.5 non-zero entries, half of
207 the columns have less than 7 non-zero entries, and 90% of the columns have less than 91 non-zero
208 entries. In our sparse representation we divide this matrix into $16 \times 16 = 256$ blocks, each with
209 dimension 12,540 by 466,416 (the size of the blocks is a tuning parameter), except at the boundary
210 the block size could be larger. As we mentioned in the last section, storing dense blocks using our
211 version of the compressed sparse block format is not memory efficient, so if a column has a large
212 number of non-zero entries we store all entries of that column separately. For this particular matrix,
213 223,596 variants does not use the sparse representation. In table 2 we present the amount of time to
214 load the matrix and to compute $X\beta$, $X^T r$ using the sparse matrix representations. Again 16 cores
215 are used for the computation. Loading such matrix would take almost 12 terabytes of memory if

216 the entries are stored as double precision floating point numbers.

| Loading | $X\beta$ | $X^T r$ |
|---------|----------|---------|
| 56.5 | 1.86 | 1.80 |

Table 2: The loading and computation time in **seconds** when the genetic matrix is stored in sparse format. This matrix has 200,643 rows and 7,462,671 columns with more than 10 billion non-zero entries. The computation time are the median of 10 runs. Both dense and sparse format compute the matrix-vector multiplication using 16 cores.

217 4 Applications to UK Biobank Exome Sequencing Data

218 In this section we put our method into practice. Specifically, we use the exome data described in
 219 the last part of section 3 to fit group-sparse linear models on multiple phenotypes. The method
 220 described in this section is implemented in `sparse-snpnet`. In this case the regularization term will
 221 be the sum of the 2-norms of the predefined groups. For our applications the groups are defined by
 222 the gene symbol of the variants. For example, the objective function for a Gaussian model is:

$$\frac{1}{n} \|y - X\beta\|_2^2 + \lambda \sum_{g \in \mathcal{G}} \sqrt{|g|} \|\beta_g\|_2 \quad (3)$$

223 where $\mathcal{G} = \{g : g \subseteq \{1, 2, \dots, d\}\}$ is a collection of indices corresponding to variants with the same
 224 gene symbol. $|g|$, the number of element in the group, is part of the regularization term so that
 225 groups of same size are penalized by the same degree. $\beta_g \in \mathbb{R}^{|g|}$ is the sub-vector of β corresponding
 226 to the indices in g . We do not allow overlapping groups, so \mathcal{G} needs to be a partition of all variables.
 227 That is $\cup_{g \in \mathcal{G}} g = \{1, 2, \dots, d\}$, and $\sum_{g \in \mathcal{G}} |g| = d$. One can show that the proximal operator for this
 228 regularization function satisfies:

$$z' := \text{prox}_{R,t}(\beta) := \arg \min_{z \in \mathbb{R}^d} \frac{1}{2t} \|z - \beta\|_2^2 + \lambda \sum_{g \in \mathcal{G}} \sqrt{|g|} \|\beta_g\|_2. \quad (4)$$

229

$$z'_g = \begin{cases} 0 & \text{if } \|z\|_2 \leq t\lambda\sqrt{|g|} \\ \left(1 - \frac{t\lambda\sqrt{|g|}}{\|z\|_2}\right) z & \text{if } \|z\|_2 > t\lambda\sqrt{|g|} \end{cases} \text{ for all } g \in \mathcal{G}. \quad (5)$$

230 In practice we would like to adjust for covariates such as age, sex, and other demographic
 231 information when fitting a regression model. In our application we first fit a unregularized regression
 232 model of the response on these covariates and fit the regularized model of the residual on the
 233 genetic variants. The number of covariates are usually much smaller compared to the number of
 234 individuals, so the first fitting is not computationally or statistically challenging. To be more precise,
 235 let $X_{cov} \in \mathbb{R}^{n \times c}$ be the $c \geq 0$ covariates that we would like to adjust for. Using the same notation
 236 in (1). We fit a model in two steps:

237 1. First, we fit a unregularized model using the covariates:

$$\hat{\beta}_{cov} = \arg \min_{\beta_{cov} \in \mathbb{R}^c} h(X_{cov}\beta_{cov}). \quad (6)$$

238 2. Then, we fit the regularized model on the “residuals” using the variants:

$$\hat{\beta} = \arg \min_{\beta \in \mathbb{R}^d} h(X_{cov}\hat{\beta}_{cov} + X\beta) + \lambda R(\beta). \quad (7)$$

239 For example, if we write the predicted value of the covariates as $\gamma = X_{cov}\hat{\beta}_{cov} \in \mathbb{R}^n$, then for a
240 Gaussian model, the objective function of the second step above is:

$$f(\beta) = \frac{1}{n} \|y - \gamma - X\beta\|_2^2 + \lambda \sum_{g \in \mathcal{G}} \sqrt{|g|} \|\beta_g\|_2 \quad (8)$$

241 For logistic regression this becomes:

$$f(\beta) = \frac{1}{n} \sum_{i=1}^n y_i \log(1 + e^{\eta_i + \gamma_i}) + (1 - y_i) \log(1 + e^{-\eta_i - \gamma_i}) + \lambda \sum_{g \in \mathcal{G}} \sqrt{|g|} \|\beta_g\|_2, \quad \eta = X\beta. \quad (9)$$

242 For Cox model the objective function is

$$f(\beta) = \frac{1}{n} \sum_{i=1}^n O_i \left[-\eta_i - \gamma_i + \log \left(\sum_{y_j \geq y_i} e^{\eta_j + \gamma_j} \right) \right] + \lambda \sum_{g \in \mathcal{G}} \sqrt{|g|} \|\beta_g\|_2, \quad \eta = X\beta. \quad (10)$$

243 We optimize these objective functions for a decreasing sequence of λ s starting from one such that
244 the solution just becomes non-zero. The initial value for the proximal gradient method of the next
245 λ are initialized from the solution from the current λ (warm start). As alluded in the last section,
246 we randomly assign 70% of the white British individuals in this dataset to the training set, 10%
247 to the validation set, and 20% to the test set. We remove individuals whose phenotype value is
248 missing, keeping variants with at least 3 minor allele count, has an associated gene symbol, has less
249 than 10% of missing value, and the ratio of the missing value and minor allele is less than 10. We
250 further filter out the variants that are not protein truncating or protein altering. Depending on the
251 number of missing values in the phenotype, the number of individuals and variants used for fitting
252 could be different. In all of the examples here the training set has more than 90,000 individuals
253 and the number of genetic variants used are over 1,000,000. The covariates are the sex, age, and 10
254 principal components of the genetic data described in the second benchmark of section 3.2.

255 To evaluate the fitted model, we use the R-squared value for quantitative phenotype, the area
256 under the receiver operating characteristic (ROC) curve (AUC) for binary phenotype, and the con-
257 cordance index (C-index) for time-to-event phenotype. These metrics will be computed on the
258 validation set to determine the optimal regularization parameter λ and on the test set to evaluate
259 the model corresponding to the λ used. Once the validation metric starts to decrease, we stop the
260 fitting process and do not compute the solutions for smaller λ values. Figures 5, 6, 7 illustrate a few
261 Lasso path plots obtained from our implementation. It is worth noting that Alzheimer's disease has
262 a sharp increase in C-index for λ indices 7-10.

263 In terms of computation, unlike in `snpnet` (or the 2.0 version), the optimization in `sparse-snpnet`
264 are all done without variable screening, and the entire training data (in sparse format) is loaded in
265 memory before fitting starts. This eliminates all the I/O operations carried out in the KKT check-
266 ing step of `snpnet`. The applications in this section successfully finished when we allocate 32GB of
267 memory to these jobs. In addition, while the applications in this paper focus on Gaussian, logistic,
268 and Cox families and group Lasso regularization, our implementation uses several abstraction in
269 C++ so it's easy to extend to other generalized linear models and regularization functions.

270 5 Discussions

271 We present two fast and memory efficient solvers for generalized linear models with regularization
272 on large genetic data. Both methods utilize a 2-bit compact representation of genetic variants and

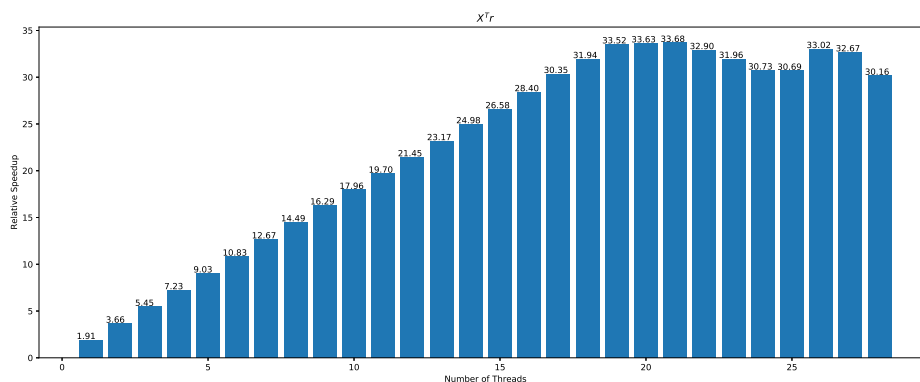


Figure 4: A bar plot demonstrating the relative speedup in computing $X^T r$ when the compact representation is used. The baseline is the R's builtin matrix-vector multiplication function for double precision matrices. The horizontal axis is the number of threads. The vertical axis is the ratio of between time spent in the baseline and the time spent with the compact representation. The baseline for $X^T r$ is 11.4 seconds.

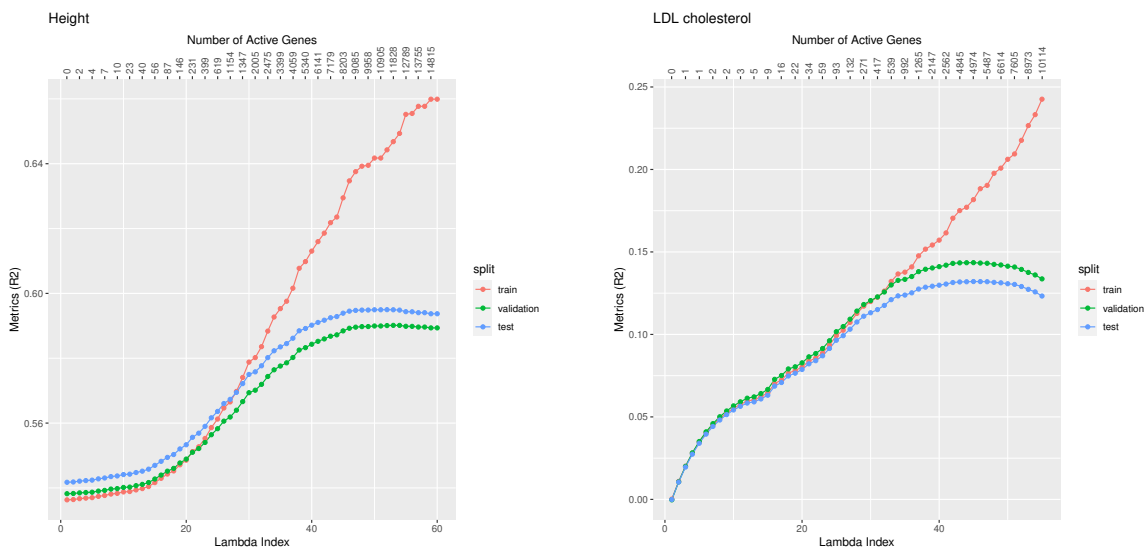


Figure 5: Lasso path plots for the quantitative phenotypes Height and LDL cholesterol. The horizontal axis is the index of the regularization parameter λ , the vertical axis are the R-squared of the solution corresponding to each λ index. The numbers on the top are the number of genes with non-zero coefficients at the corresponding λ indices. The color corresponds to the train, validation and test set. The duration of training these two models are 8.34 minutes and 8.35 minutes respectively.

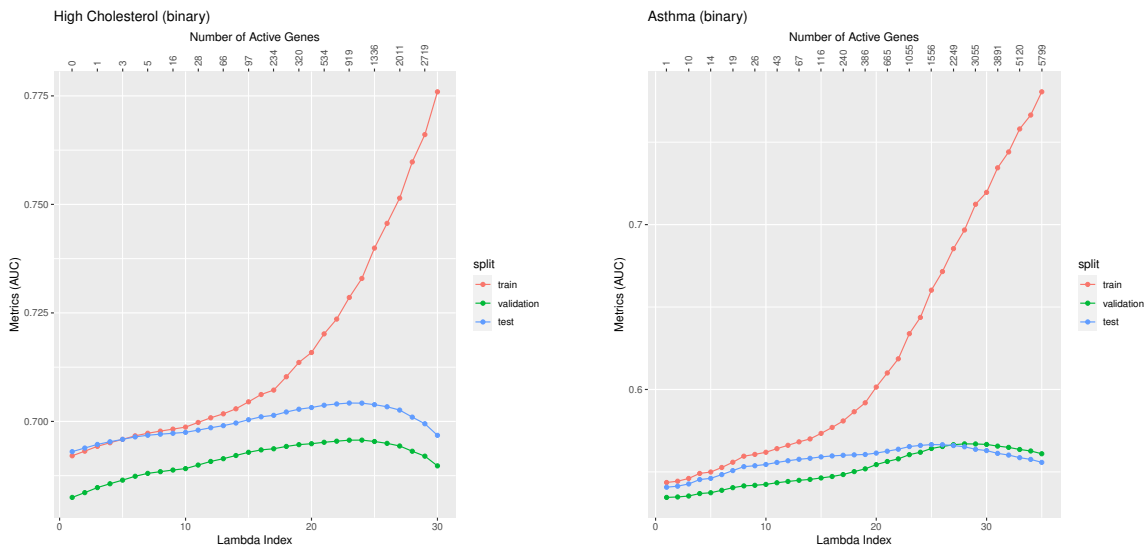


Figure 6: Lasso path plots for the binary phenotypes High Cholesterol and Asthma. The horizontal axis is the index of the regularization parameter λ , the vertical axis are the AUC values of the solution corresponding to each λ index. The numbers on the top are the number of genes with non-zero coefficients at the corresponding λ indices. The color corresponds to the train, validation and test set. The duration of training these two models are 6.88 minutes and 8.27 minutes respectively.

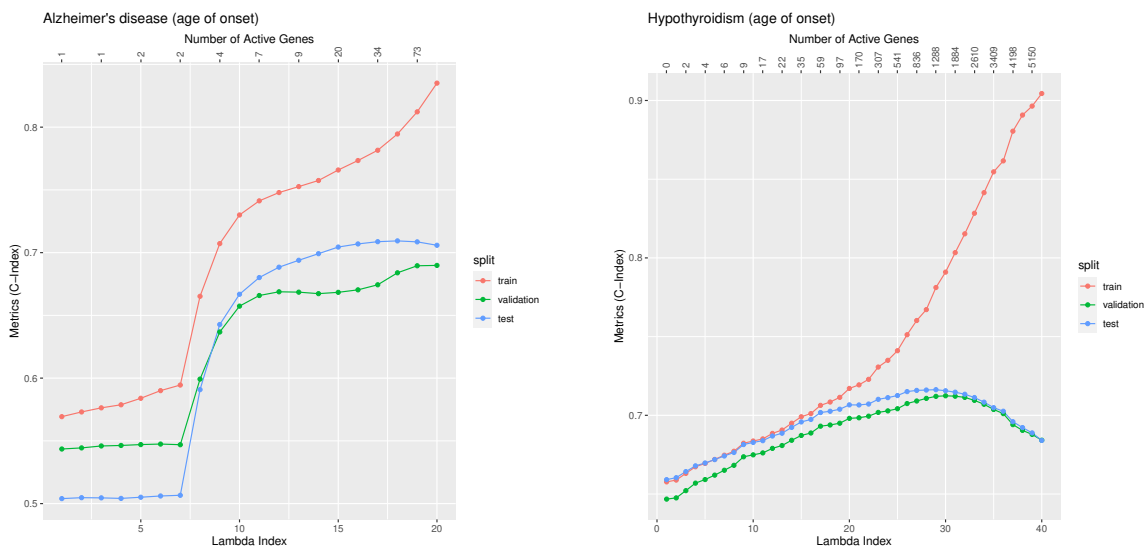


Figure 7: Lasso path plots for the time-to-event phenotypes Alzheimer's disease and Hypothyroidism. The horizontal axis is the index of the regularization parameter λ , the vertical axis are the C-index of the solution corresponding to each λ index. The numbers on the top are the number of genes with non-zero coefficients at the corresponding λ indices. The color corresponds to the train, validation and test set. The duration of training these two models are 6.12 minutes and 6.92 minutes respectively.

273 are accelerated through multi-threading on CPUs with multiple cores. The first solver implements
274 the iteratively-reweighted least square algorithm in `glmnet`, and its goal is to provide boosted com-
275 putational and memory performance to the large-scale Lasso solver described in (Qian et al. 2020,
276 Li et al. 2020). The second solver implements an accelerated proximal gradient method that’s able
277 to solve more general regularized regression problems. One important feature of this solver is that it
278 combines a version of compressed sparse block format for sparse matrices and the 2-bit encoding of
279 genetic variants. We summarize the characteristics of these two solvers in table 3. We demonstrate
280 the effectiveness of our methods through several benchmarks and UK Biobank exome data applica-
281 tions. We believe our method will be a useful tool as whole genome sequencing data becomes more
282 common.

| | <code>snpnet-2.0</code> | <code>sparse-snpnet</code> |
|---|-------------------------|----------------------------|
| Algorithm | IRLS | Proximal gradient |
| Use variable screening | Yes | No |
| Easy to extend to other GLMs | Yes | Yes |
| Easy to extend to other regularizations | No | Yes |
| Use 2-bit representation of variants | Yes | Yes |
| Use sparse matrix format | No | Yes |
| Multi-threaded | Yes | Yes |

Table 3: A comparison between the two solvers we present in this paper.

283 6 Acknowledgments

284 Y.T. is supported by a Funai Overseas Scholarship from the Funai Foundation for Information
285 Technology and the Stanford University School of Medicine.

286 M.A.R. is supported by Stanford University and a National Institute of Health center for Multi
287 and Trans-ethnic Mapping of Mendelian and Complex Diseases grant (5U01 HG009080). This work
288 was supported by National Human Genome Research Institute (NHGRI) of the National Institutes
289 of Health (NIH) under awards R01HG010140. The content is solely the responsibility of the authors
290 and does not necessarily represent the official views of the National Institutes of Health.

291 R.T was partially supported by NIH grant 5R01 EB001988-16 and NSF grant 19 DMS1208164.

292 T.H. was partially supported by grant DMS-1407548 from the National Science Foundation, and
293 grant 5R01 EB 001988-21 from the National Institutes of Health.

294 This research has been conducted using the UK Biobank Resource under application number
295 24983. We thank all the participants in the study. The primary and processed data used to generate
296 the analyses presented here are available in the UK Biobank access management system (<https://amsportal.ukbiobank.ac.uk/>) for application 24983, “Generating effective therapeutic hypotheses from genomic and hospital linkage data” (<http://www.ukbiobank.ac.uk/wp-content/uploads/2017/06/24983-Dr-Manuel-Rivas.pdf>).

300 All of the computing for this project was performed on the Nero and Sherlock clusters. We
301 would like to thank Stanford University and the Stanford Research Computing Center for providing
302 computational resources and support that contributed to these research results.

303 *Conflict of Interest:* None declared.

References

- 304
- 305 Aguirre, M., Rivas, M. A. & Priest, J. (2019), ‘Phenome-wide Burden of Copy-Number Variation
306 in the UK Biobank’, *Am J Hum Genet* **105**(2), 373–383.
- 307 Beck, A. & Teboulle, M. (2009), ‘A fast iterative shrinkage-thresholding algorithm for linear inverse
308 problems’, *SIAM J. Img. Sci.* **2**(1), 183–202.
309 **URL:** <https://doi.org/10.1137/080716542>
- 310 Buluç, A., Fineman, J. T., Frigo, M., Gilbert, J. R. & Leiserson, C. E. (2009), Parallel sparse matrix-
311 vector and matrix-transpose-vector multiplication using compressed sparse blocks, in ‘Proceedings
312 of the Twenty-First Annual Symposium on Parallelism in Algorithms and Architectures’, SPAA
313 ’09, Association for Computing Machinery, New York, NY, USA, p. 233–244.
314 **URL:** <https://doi.org/10.1145/1583991.1584053>
- 315 Chang, C., Chow, C., Tellier, L., Vattikuti, S., Purcell, S. & Lee, J. (2015), ‘Second-generation plink:
316 Rising to the challenge of larger and richer datasets’, *GigaScience* **4**.
- 317 Cox, D. R. (1972), ‘Regression models and life-tables’, *Journal of the Royal Statistical Society. Series*
318 *B (Methodological)* **34**(2), 187–220.
319 **URL:** <http://www.jstor.org/stable/2985181>
- 320 Daubechies, I., Defrise, M. & De Mol, C. (2004), ‘An iterative thresholding algorithm for linear
321 inverse problems with a sparsity constraint’, *Communications on Pure and Applied Mathematics*
322 **57**(11), 1413–1457.
323 **URL:** <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpa.20042>
- 324 DeBoever, C., Tanigawa, Y., Lindholm, M. E., McInnes, G., Lavertu, A., Ingelsson, E., Chang,
325 C., Ashley, E. A., Bustamante, C. D., Daly, M. J. et al. (2018), ‘Medical relevance of protein-
326 truncating variants across 337,205 individuals in the uk biobank study’, *Nature communications*
327 **9**(1), 1–10.
- 328 Friedman, J., Hastie, T. & Tibshirani, R. (2010), ‘Regularization paths for generalized linear models
329 via coordinate descent’, *Journal of Statistical Software* **33**(1), 1–22.
330 **URL:** <https://www.jstatsoft.org/v33/i01/>
- 331 Hastie, T. & Tibshirani, R. (1986), ‘Generalized additive models’, *Statist. Sci.* **1**(3), 297–310.
332 **URL:** <https://doi.org/10.1214/ss/1177013604>
- 333 Li, R., Chang, C., Justesen, J. M., Tanigawa, Y., Qian, J., Hastie, T., Rivas, M. A. & Tibshirani, R.
334 (2020), ‘Fast Lasso method for large-scale and ultrahigh-dimensional Cox model with applications
335 to UK Biobank’, *Biostatistics* . kxaa038.
336 **URL:** <https://doi.org/10.1093/biostatistics/kxaa038>
- 337 Nesterov, Y. (1983), ‘A method for solving the convex programming problem with convergence
338 rate $O(1/k^2)$ ’, *Proceedings of the USSR Academy of Sciences* **269**, 543–547.
- 339 Qian, J., Tanigawa, Y., Du, W., Aguirre, M., Chang, C., Tibshirani, R., Rivas, M. A. & Hastie, T.
340 (2020), ‘A fast and scalable framework for large-scale and ultrahigh-dimensional sparse regression
341 with application to the uk biobank’, *PLOS Genetics* **16**(10), 1–30.
342 **URL:** <https://doi.org/10.1371/journal.pgen.1009141>
- 343 Simon, N., Friedman, J., Hastie, T. & Tibshirani, R. (2011), ‘Regularization paths for cox’s propor-
344 tional hazards model via coordinate descent’, *Journal of Statistical Software* **39**(5), 1–13.
345 **URL:** <https://www.jstatsoft.org/v39/i05/>

- 346 Sinnott-Armstrong, N., Tanigawa, Y., Amar, D., Mars, N. J., Aguirre, M., Venkataraman, G. R.,
347 Wainberg, M., Ollila, H. M., Pirruccello, J. P., Qian, J., Shcherbina, A., FinnGen, Rodriguez, F.,
348 Assimes, T. L., Agarwala, V., Tibshirani, R., Hastie, T., Ripatti, S., Pritchard, J. K., Daly, M. J.
349 & Rivas, M. A. (2021), ‘Genetics of 38 blood and urine biomarkers in the uk biobank’, *Nature*
350 *Genetics (in press)* .
351 **URL:** <https://www.biorxiv.org/content/early/2019/06/05/660506>
- 352 Sudlow, C., Gallacher, J., Allen, N., Beral, V., Burton, P., Danesh, J., Downey, P., Elliott, P.,
353 Green, J., Landray, M., Liu, B., Matthews, P., Ong, G., Pell, J., Silman, A., Young, A., Sprosen,
354 T., Peakman, T. & Collins, R. (2015), ‘Uk biobank: An open access resource for identifying the
355 causes of a wide range of complex diseases of middle and old age’, *PLOS Medicine* **12**(3), 1–10.
356 **URL:** <https://doi.org/10.1371/journal.pmed.1001779>
- 357 Szustakowski, J. D., Balasubramanian, S., Sasson, A., Khalid, S., Bronson, P. G., Kvikstad, E.,
358 Wong, E., Liu, D., Davis, J. W., Haefliger, C., Loomis, A. K., Mikkilineni, R., Noh, H. J.,
359 Wadhawan, S., Bai, X., Hawes, A., Krasheninina, O., Ulloa, R., Lopez, A., Smith, E. N., Waring,
360 J., Whelan, C. D., Tsai, E. A., Overton, J., Salerno, W., Jacob, H., Szalma, S., Runz, H.,
361 Hinkle, G., Nioi, P., Petrovski, S., Miller, M. R., Baras, A., Mitnau, L. & Reid, J. G. a. (2020),
362 ‘Advancing human genetics research and drug discovery through exome sequencing of the uk
363 biobank’, *medRxiv* .
364 **URL:** <https://www.medrxiv.org/content/early/2020/11/04/2020.11.02.20222232>
- 365 Tibshirani, R. (1996), ‘Regression shrinkage and selection via the lasso’, *Journal of the Royal Sta-*
366 *tistical Society. Series B (Methodological)* **58**(1), 267–288.
367 **URL:** <http://www.jstor.org/stable/2346178>
- 368 Venkataraman, G. R., Olivieri, J. E., DeBoever, C., Tanigawa, Y., Justesen, J. M., Dilthey, A. &
369 Rivas, M. A. (2020), ‘Pervasive additive and non-additive effects within the hla region contribute
370 to disease risk in the uk biobank’, *bioRxiv* .
371 **URL:** <https://www.biorxiv.org/content/early/2020/06/12/2020.05.28.119669>
- 372 Yuan, M. & Lin, Y. (2006), ‘Model selection and estimation in regression with grouped variables’,
373 *Journal of the Royal Statistical Society Series B* **68**, 49–67.
- 374 Zou, H. & Hastie, T. (2005), ‘Regularization and variable selection via the elastic net’, *Journal of*
375 *the Royal Statistical Society. Series B (Statistical Methodology)* **67**(2), 301–320.
376 **URL:** <http://www.jstor.org/stable/3647580>