Implementation

SparSNP implements a fast and memory-efficient out-of-core version of the cyclical coordinate descent method (Friedman et al., 2007, 2010) for minimising $\ell_1$-penalised loss functions.

$\ell_1$-penalised loss functions

The problem of fitting linear models can be cast as minimising a convex loss function $L$. Two common loss functions are the squared loss for linear regression and the logistic loss for logistic regression. The squared loss function over $N$ samples in $p$ variables is

$$L(\beta_0, \beta) = \frac{1}{2} \sum_{i=1}^{N} (y_i - \beta_0 - x_i^T \beta)^2,$$

(1)

where $x_i \in \mathbb{R}^p$ are the inputs for the $i$th sample, $y \in \mathbb{R}^N$ is the $N$-vector of outputs, $\beta_0 \in \mathbb{R}$ is the intercept, and $\beta \in \mathbb{R}^p$ is a $p$-vector of model weights. Similarly, logistic loss for binary outcomes $y_i \in \{0, 1\}$ is

$$L(\beta_0, \beta) = \sum_{i=1}^{N} \log(1 + \exp(\beta_0 + x_i^T \beta)) - y_i(\beta_0 + x_i^T \beta).$$

(2)

Another loss function useful in classification is squared-hinge loss, which is equivalent to a least-squares support vector machine with a linear kernel (Chang et al., 2008),

$$L(\beta_0, \beta) = \frac{1}{2} \sum_{i=1}^{N} \max\{0, 1 - y_i(\beta_0 + x_i^T \beta)\}^2,$$

(3)

where $y_i \in \{-1, +1\}$. SparSNP uses the squared hinge loss as the classification model. We use the squared-hinge loss rather than the standard hinge loss since the former is twice-differentiable whereas the latter is not. Twice-differentiability means that the Newton step-size used in coordinate descent is the second derivative of the loss wrt $\beta_j$, whereas to use coordinate descent with the hinge loss requires some pre-chosen step size, which must be found through manual tuning or a line search procedure. The disadvantage of the squared-hinge loss is that it is potentially more sensitive to outliers in the data, with the loss increasing quadratically for mis-classified samples, whereas the hinge loss increases the loss only linearly.
Any of these loss functions can be penalised with $\ell_1$ (lasso) penalty and minimised to find the solutions $(\beta_0^*, \beta^*)$

$$(\beta_0^*, \beta^*) = \arg \min_{\beta_0 \in \mathbb{R}, \beta \in \mathbb{R}^p} L(\beta_0, \beta) + \lambda \sum_{j=1}^p |\beta_j|.$$  

(4)

The penalty $\lambda \geq 0$ is user-specified and controls the degree of penalisation. The $\ell_1$ penalty encourages sparse solutions (many $\hat{\beta}_j$ exactly zero for high-enough $\lambda$), whereas the $\ell_2$ penalty (sum of squared weights), used in ridge regression, induces proportional shrinkage of the estimates, but generally does not induce sparse solutions.

**Out-of-core coordinate descent**

$\ell_1$ regression is a convex optimisation problem. However, in general, it has no analytical solutions, and must be solved numerically. We use a variant of coordinate descent to numerically minimise the loss function.

In coordinate descent (Van der Kooij, 2007; Friedman et al., 2007, 2010), each variable is optimised with respect to the loss function using a univariable Newton step, while holding the other variables fixed. Since the updates are univariable, computation of the first and second derivatives is fast and simple (we assume that all our loss functions are twice-differentiable). The $\ell_1$ penalisation is achieved using soft thresholding (Friedman et al., 2007) of each estimated weight $\hat{\beta}_j$

$$\hat{\beta}_j \leftarrow S\left(\hat{\beta}_j - s_j, \lambda\right),$$  

(5)

where $s_j = \frac{\partial L}{\partial \hat{\beta}_j} / \frac{\partial^2 L}{\partial \hat{\beta}_2}$ is the Newton step with respect to $\beta_j$ and $S(\cdot, \cdot)$ is the soft thresholding operator

$$S(\alpha, \gamma) = \text{sign}(\alpha) \max\{0, |\alpha| - \gamma\}, \quad \gamma \geq 0.$$  

For the linear loss and the squared hinge loss, the Newton step yields the exact solution with respect to each $\beta_j$ (though it is not necessarily the same as the global solution, hence the need for iteration over all variables until convergence, see below).

Since SparSNP repeatedly iterates over $p$ variables in $N$ samples, it has a time complexity of $O(Np)$ per iteration, but in practice not all variables are visited in each iteration, due to the active set convergence method (see below). Running time also depends on how many SNPs are allowed in the model, how many of them can be cached in RAM, and the number of penalties on the grid.

There are two key aspects of this approach that allow efficient computation without keeping all data in working memory. First, since we are performing univariable minimisation, both the first and second derivatives are scalars and are computed in a single pass over the samples. Second, the partial derivative wrt $\hat{\beta}_j$ is computed efficiently since it is based on the linear predictor $l$, which is the sum of the contributions of all variables to the model $l_i = \hat{\beta}_0 + \sum_{j=1}^p x_{ij} \hat{\beta}_j$, $i = 1, \ldots, N$. The linear predictor changes only for one variable at a time, and only if that variable changes its value. Once the estimate for the $j$th variable has been updated, the linear predictor is updated by subtracting the old contribution $x_{ij} \hat{\beta}_j$ and adding the new contribution $x_{ij} \hat{\beta}'_j$. The linear predictor form can be used whenever a linear or log-linear statistical model is used, since then the predictor is additive in each variable’s contribution. By only storing the linear predictor, one vector of $N$ samples for a given SNP,
and the $p$-vector of model weights in memory at any given time, the memory requirements of SparSNP grow only linearly with the samples and the variables $O(N + p)$, allowing us to fit models to datasets far larger than available RAM. For the squared hinge loss, several auxiliary $N$-vectors are maintained in memory as well in order to avoid recomputing them when they do not change between iterations.

The coordinate descent algorithm is identical across all loss functions, the only difference is the computation of the Newton step and the update to the linear predictor. For the linear loss, the first derivative with respect to the weight $\beta_j$ is

$$\frac{\partial L}{\partial \beta_j} = \sum_{i=1}^{N} x_{ij}(l_i - y_i),$$

and the second derivative is

$$\frac{\partial^2 L}{\partial \beta_j^2} = \sum_{i=1}^{N} x_{ij}^2.$$  

For the squared hinge loss, the first derivative is

$$\frac{\partial L}{\partial \beta_j} = \sum_{i=1}^{N} y_i x_{ij}(y_i l_i - 1)I(1 - y_i l_i > 0),$$

and the second derivative is

$$\frac{\partial^2 L}{\partial \beta_j^2} = \sum_{i=1}^{N} x_{ij}^2 I(1 - y_i l_i > 0),$$

where $I(\cdot)$ is the indicator function which evaluates to 1 if its argument is true and to zero otherwise. Monomorphic SNPs are assigned zero weight since their first and second derivatives are both zero, making the Newton step undefined. When the input data are standardised such that each SNP has zero-norm and unit-variance, the second derivative is $\frac{\partial^2 L}{\partial \beta_j^2} \leq N - 1$ for linear and squared hinge loss (due to the $I(\cdot)$ term), with strict equality for linear loss. Therefore the Newton step can be computed as $s_j = (\frac{\partial L}{\partial \beta_j})/(N - 1)$, without explicitly computing the second derivative.

Tseng (2001) showed that coordinate descent converges to the global minimum when two conditions are met: (i) the loss function being minimised is convex, as is the case with most common loss functions such as those used here, and (ii) the penalty is separable, that is, the penalty is a sum of functions where each function is a function of one weight $\beta_j$. Since the $\ell_1$ penalty is a separable penalty, coordinate descent is guaranteed to find the global minimum, when used in conjunction with a convex loss function.

**Computational enhancements**

SparSNP employs several enhancements to the basic coordinate descent method, that greatly improve performance without affecting model fit.
Active-set convergence

The active set method (Friedman et al., 2007) is designed to take advantage of the sparsity of the weight vector $\beta$, as is commonly the case in analyses of SNP data, where only a small fraction of the SNPs are expected to have non-zero weights. The method has two main stages. First, we iterate over all variables, one at a time. If any variable $j$ becomes zero (inactive) due to the soft-thresholding, it is excluded from the next iteration. We then iterate over the remaining active variables. When the loss converges, we check whether the active set has changed. If the active set does not change in two such consecutive iterations then the algorithm terminates. Otherwise, all variables are added to the active set and iterated over as before.

Warm restarts

We use a warm-restart strategy (Friedman et al., 2010) whereby we run coordinate-descent over a grid of $\lambda$ penalties $\lambda^{\text{max}}, \ldots, \lambda^{\text{min}}$. We define the maximal $\lambda$ as the smallest $\lambda$ that makes all $\hat{\beta}_j = 0$; maximal $\lambda$ is computed by first computing the unpenalised intercept, and then evaluating the Newton step (Eq. 5) for each variable $j$. Each weight $\hat{\beta}_j$ is initialised to zero. Due to soft thresholding (Eq. 5), each $\hat{\beta}_j$ will remain zero if its step $|s_j| \leq \lambda$. Therefore, the maximal $\lambda$ is taken to be a small fraction of the maximal $\lambda$, usually $10^{-2}\lambda^{\text{max}}$.

The process proceeds to the next fit, with the results from the previous fit (including the vector of solutions $\hat{\beta}$, the linear predictor $l$, and the active set) used to initialise the algorithm for the next fit. This strategy reduces computation time considerably, since the $(k+1)$th fit can typically start from a small active set found by the $k$th model.

Caching

The active set is typically much smaller than the total number of SNPs, and tends to be accessed more often than other variables. Therefore, it is useful to keep the active set in memory rather than repeatedly read it from disk, which is orders of magnitude slower, as in addition to disk random-access being slower than random access of RAM, the data are byte-packed and must be unpacked before use. However, all SNPs need to be accessed at some stage during the active set convergence process. Therefore, we employ a simple priority cache of predetermined size. If there is room in the cache, more SNPs are read in until it is full. Once full, we read SNPs into the cache, replacing previous SNPs only if the new SNP has been accessed more often in previous iterations (we keep a counter for each SNP). This way, the active set SNPs (and other often-accessed SNPs, if there is room) tend to stay in the cache whereas the other SNPs do not, accelerating the active set method.

Since caching more SNPs reduces disk accesses, the performance of SparSNP critically depends on the amount of RAM allocated to the cache. Performance will increase with increasing cache size, until the point where the entire dataset is in RAM. Additionally, performance may degrade if SparSNP is run on networked drives, compared with local physical drives.
**Pre-scaling**

Inputs to $\ell_1$-penalised models are typically standardised such that each genotype has zero-mean and unit variance, since each input variable may be on a different scale, in which case using the same penalty for all variables may not make sense. In the context of SNP data, scaling the SNPs corresponds to giving more weight to rarer variants. Standardisation is also crucial for consistent convergence of the coordinate descent method.

Repeatedly scaling the discrete inputs is wasteful since the coordinate descent method repeatedly iterates over the variables. Therefore, we scale the data as a pre-processing step, and store the means and standard deviations in a file. These parameters are later loaded during the actual fitting stage, and the precomputed values for each SNP are fetched from a lookup table. When the fitting is complete, we transform the model weights back to their original scale.

**Byte packing the data**

We represent the genotypes in the minor allele dosage form $\{0, 1, 2, \text{NA}\}$, where “NA” denotes missing genotypes. We use the same byte packing scheme as PLINK, encoding 4 genotypes in one byte, greatly reducing space requirements compared to 8 bytes per genotype that would be required for double precision floating point representation (the default for most numerical software). Besides space savings, byte packing leads to faster I/O, which is the main bottleneck of our method. Note that for the fitting stage, data are used in their scaled form, which is in double precision floating point format, and must be unpacked prior to use. Fast unpacking is achieved using a pre-computed lookup table that maps bytes (interpreted as short unsigned integers in the range $[0, 255]$) to groups of four genotypes at a time.

**Update-on-change**

As mentioned earlier, the linear predictor $l$ must be updated every time a variable $j$ changes its value. This involves iterating over one $N$-vector (for linear loss) or several $N$-vectors (precomputed products of the $l$ vector with $y$ vector, for squared hinge loss). Such updates only occur when the $j$th variable is active (non-zero) and the estimate $\hat{\beta}_j$ has changed from the last iteration, saving unnecessary updates.
Supplementary Results

Figure 1: (A) Absolute value of the Pearson correlation of the model weights of all SNPs, across the five methods. (B) The proportion of SNPs with non-zero weights that were in common between each pair of methods, out of 128 SNPs. For both plots, we used lasso models trained on the entire Finnish celiac disease dataset inducing 128 SNPs. For the non-sparse LIBLINEAR-CDBLOCK (LL-CD-L2), all SNPs were in the model and we selected the top 128 SNPs by absolute value of the weight. We selected the hyperparameters for each method that lead to 128 SNPs with non-zero weights, as at this number of SNPs all methods showed good predictive performance. For the non-sparse LIBLINEAR-CDBLOCK method, we used $C = 1$ and selected the top 128 SNPs using the absolute value of the model weights. We then trained each method on the entire Finnish celiac disease dataset using these hyperparameters.
References


