## lme1en: A fast algorithm to infer Elastic Net with random batch effect when ‘*n<<p*’

*lme1en = linear mixed effect model (1=random intercept only) with elastic net penalties*

*‘n<<p’ = the number of covariates is* ***much*** *larger than the number of observations*

### The data:

batches (studies)

For each individual within batch we have

Response: , for instance age

Covariate: , for instance methylation signal

Total number of observations:

### The (batch effect) model and its likelihood function:

For batch and observation within batch: :

where are coefficients-weights (vectorized).

Vectorized version of model (for batch ):

Where

Reparameterization: and such that

The model parameters are denoted as

The log-likelihood function at batch becomes:

And over all batches we have the total log-likelihood as

### Cholesky factorization of covariance matrix

We Cholesky factorize the correlation structure such that

And we also invert the Cholesky matrix: such that the likelihood becomes

### Introducing shrinkage using the elastic net model

We introduce the Elastic net penalty to shrink the coefficients in the model:

Here is the tradeoff between LASSO () and RIDGE ( and is a *hyperparameter* (to be tuned). Notice that in the *glmnet* implementation, the penalty is slightly modified:

In lme1en the user can choose which of the penalties or to use.

### The objective function

In this application we want to estimate the coefficientand treat () as known. From this the objective function (to minimize) is given as

We only need the proportional expression of the objective function which is given as

We define the residual sum square as

and rescaling the objective function with further gives

where . We continue to use for . is now similar to the objective function defined in the *glmnet* implementation. This is useful because we can utilize this implementation to provide warm-ups for .

### Optimizing the objective function using the coordinate descent algorithm

EXPLAIN COORDINATE DESCENT ALGORITHM

In each step of the algorithm we are interested in optimizing the ‘one-variable’-problem

### Theorem

There is an exact solution to this problem without the need of numerical optimization.

where

and the following linear transformations are carried out (decorrelating wrt covariance matrix

### Proof

We start by substituting the variables: and andexpand such that

Hence

To optimize wrt we derivate:

Setting the derivative gives the equation

with the form .

The equation c has solution and this is used to get the expression for .

### Avoid scaling with n

From the equation above we see that we can simply replace with to avoid the scaling each of the sum squares with . However, for very large this may be less robust.

### The *glmnet* penalty

To use the *glmnet* penalty instead of we simply replace with in the term.

### Efficient variable storage in the algorithm

Storing variables before running the iterations is essential for fast computation:

|  |  |  |
| --- | --- | --- |
| (I) |  |  |

|  |  |  |
| --- | --- | --- |
| (II) |  |  |

(III)

The following expression is dynamically changing with the iteration number :

Notice that the long vector

is a matrix multiplication which can be calculated only ones for the batches . This is very efficient for the situations.is a variable which is updated dynamically to efficient calculate for each iteration .

### The algorithm

*Initiation:* We first initiate using the glmnet implementation (=0 and penalty used with and same as input). Before iterating the for-loop we calculate for each batch and we compute at the same time when traversing through each batch. Notice that is a long dot product, hence only O( in total computation.

***For each***  we do as follow:

Step 1) Calculate using formula (this is new

Step 2) If (end of for-loop), check if |-:

If TRUE: Solution has converged

If FALSE: Proceed in a new loop (not converged), starting with (complete Step 3 first)

Step 3) **Post-steps** (preparing for solution):

Updating to and at the same time calculate :

* Calculate for each batch
* Calculate when traversing the batches (in same loop).

The operations costs O(n) in computation.

End-of-loop notice: , since starts at 1 if .

### Sufficient variables

Data input: 3 -long vectors + long list with matrices = in total.

* initiations for
* for
* for
* matrices for batches

### Extension to more (nested) layers of batch effects

For batch (level 1), and batch (level 2 within level 1) and observation within batch (level 1): :

where are coefficients-weights (vectorized) as before. The parameter could belong to a specific batch without loss of generalization (however introducing more parameters).

As before we vectorize the observation for batch (which also contains info about batch ):

where

and the size of the block structures is decided by the number of observations in batch .

Reparameterization: and and such that

which is computed in the pre-step and the rest of the algorithm remains the same. Parameters are . Extending the model with even more nested layers of batches is possible by splitting the blocks further (as carried out here).