MyESL: Sparse learning in molecular evolution and phylogenetic analysis

Maxwell Sanderford [1], Sudip Sharma [1,2], Glen Stecher [1], Jun Liu [3], Jieping Ye [4], and Sudhir Kumar [1,2]*

[1]Institute for Genomics and Evolutionary Medicine, Temple University, Philadelphia, PA-19140, USA,
[2]Department of Biology, Temple University, Philadelphia, PA-19140, USA,
[3]Infinia ML Inc., Durham, North Carolina, USA, and
[4]Zhejiang Lab, Hangzhou, P.R. China.

*To whom correspondence should be addressed.

Abstract

Evolutionary sparse learning (ESL) uses supervised sparse learning to construct evolutionary models with genomic loci as parameters. It utilizes the Least Absolute Shrinkage and Selection Operator (LASSO) to build models connecting a specified phylogenetic hypothesis with the variation across sequences in alignments. The MyESL software fills the need for open-source software for conducting ESL analyses with facilities to pre-process the input alignments, make LASSO flexible and computationally efficient, and post-process the output model to produce evolutionary metrics. The computational core of MyESL is written in C++, which offers model building using logistic regressions with bi-level sparsity for positions and groups of positions, such as genes. A set of Python utilities written for MyESL takes collections of FastA files and transforms them into numerical representations that are phylogenetically compressed and class-balanced for efficient ESL analyses. The Python library also writes results in user-friendly output in text and graphical files. In addition to the Python environment, MyESL is available as a standalone application and can be plugged into third-party software and pipelines.
Availability: Download source code, executable, and documentation from `https://github.com/kumarlabgit/MyESL`
Corresponding author: Sudhir Kumar (s.kumar@temple.edu).

1 Introduction

Evolutionary sparse learning (ESL) uses supervised machine learning with a sparsity constraint for comparative sequence analysis in a phylogenetic framework (Kumar and Sharma, 2021). ESL is applied directly to multiple sequence alignments and builds a model for a given phylogenetic hypothesis, such as the grouping of organisms in a clade or the presence or absence of a trait of interest across organisms in a phylogeny. Organisms can be species, individuals, strains, or cells, among other possibilities. ESL model parameters are genomic loci, which can be genes, proteins, exons, introns, intergenic regions, and individual genomic positions (Fig. 1a).

ESL uses the Least Absolute Shrinkage and Selection Operator, LASSO (Tibshirani, 1996) and automatically compares alternative models involving different combinations of genomic loci and positions using sparse group lasso with logistic loss (Simon *et al.*, 2013; Qiao *et al.*, 2017). The selected model reveals key genes and positions containing the most informative shared-derived evolutionary substitutions, along with a measure of the importance of each gene and position referred to as sparsity scores that are larger for more important loci as well as position in those loci (Kumar and Sharma, 2021). The classification model based on these genes could clearly distinguish between members and non-members of a clade in a phylogeny (Kumar and Sharma, 2021). Recently, ESL has been used to detect disruptive sequences and unstable clades in species phylogenies inferred using phylogenomic alignments (Sharma and Kumar, 2024).

The ESL approach was originally implemented using Sparse Learning with Efficient Projections (SLEP) software (Liu *et al.*, 2011). It implements LASSO regression in MATLAB but does not have built-in functionality to process input sequence alignments and ESL model outputs. MATLAB is neither universally accessible nor free of cost, limiting the use of ESL. For proprietary reasons, the MATLAB version of ESL cannot be distributed with free, user-friendly, and freely available software like MEGA (Tamura *et al.*, 2021). In addition, packaging updated MATLAB codes into a standalone software distribution by third parties will require that they have a paid MATLAB compiler.

The above considerations prompted us to develop an open-source software implementing SLEP's sparse group lasso functionalities in C++ for computational efficiency and facilitating easier updates. We also programmed a new library of key input/output functionalities to handle phylogenomic datasets. This library is written in Python and selected for its common use in genomics research. These programming language choices for the new programs, collectively called MyESL, enabled its compilation into platform-specific executables (e.g., MS Windows) that can be distributed and used without setting up a Python environment. This executable is linked to the MEGA software in the upcoming version 12, making its applications accessible via a widely used Graphical User Interface.

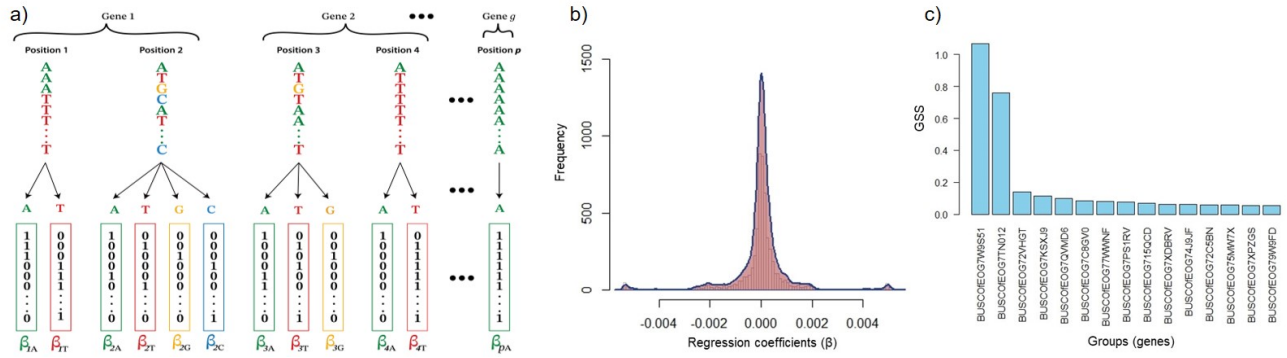

Fig. 1. One-hot encoding and Evolutionary Sparse Learning. (a) The sequence alignment input to ESL consists of p positions (columns) belonging to g groups (e.g., genes). The one-hot representation of the alignment is shown below the sequence, where each allele present at a position gets a bit column containing a 1 when the given allele is present in the position and a 0 otherwise. Every bit column is a feature in the ESL model, which produces weights (β) for each bit column. β captures the correlation between the binary pattern in the bit column and the hypothesis specified by labels ($+1$ or -1) assigned to rows in the alignment. (b) The distribution of non-zero regression coefficients that were estimated in MyESL. (c) Top genes ordered by gene sparsity score (GSS) for features selected in the ESL model.

2 Results

The analysis options are provided to the MyESL software on the command line, including a path for the input data file and a text file containing the evolutionary hypothesis.

2.1 Reading and processing the evolutionary hypothesis:

MyESL begins by reading the text file containing the evolutionary hypothesis specified in two ways: `--classes <classfile.txt>` or `--tree <treefile.nwk>` directive on the command line. The `classfile.txt` is a tab-separated two-column text file containing the organism name in the first column and a class designation ($+1$ or -1) in the second column for use as a response in the regression analysis. The +1 class is considered the focal class, which can be a clade in a phylogeny or a trait of interest.

Alternatively, MyESL can automatically generate classes by processing an input phylogeny provided in a tree file containing a rooted tree in the Newick format. In this tree, the internal node (clade) with a label will be used as the focal clade, i.e., all the names found in the subtree defined by that internal node will be assigned to the +1 class. The remaining taxa in the tree will be assigned to the -1 class. If multiple nodes in the input phylogeny have labels, the `--clade_list <name>` directive specifies the focal clade. MyESL can conduct multiple ESL analyses by automatically generating labels for all the internal nodes in the phylogeny using the `--gen_clade_list` directive, where users are required to define the minimum and maximum count of clade members.

2.2 Class balancing:

MyESL provides many ways to achieve class balancing, which refers to having an equal number of taxa in the two classes (+1 and -1) using the `--class_bal` command. Two classes can be balanced by up-sampling of the minority class (`<up>` option) or down-sampling of the majority class (`<down>` option). In these two scenarios, the taxa included are selected randomly. The `<weighted>` option will assign weights inversely proportional to the class size to balance the contribution of the two classes.

MyESL provides a novel option (`<phylo>`) when the evolutionary hypothesis is specified through a Newick tree that contains branch lengths. In this phylogeny-aware class balancing, MyESL first assigns a +1 to all the taxa in the focal clade. Then, it scans the sister of the focal clade and assigns -1 to each taxa therein (i.e., first cousins). If the number of taxa in the -1 class is smaller than the focal clade, then the search for additional taxa continues by moving up to the ancestor of the focal clade and assigning taxa in the sister clade at the next level to be -1 (i.e., second cousins). This process is repeated until no more taxa remain or the number of taxa with a -1 label exceeds those in the focal clade. At this stage, if the number of taxa in two classes is unequal, MyESL prunes the taxon with the shortest terminal branch length in the class with the larger number of taxa, a process repeated until the number of taxa becomes equal between the two classes. In this way, the most diverse taxa are selected for ESL analysis.

2.3 One-hot encoding of sequence alignment:

Sequence alignments are read from the FastA files. MyESL assumes that each FastA file represents a distinct group of alignment positions, such as a gene or a collection intended to be treated as a group (Fig. 1a). All sequences are then one-hot encoded, a functionality implemented in C++ for maximizing speed. In one-hot encoding, every position in the sequence alignment is converted into as many bit columns as the number of unique characters present at that position. No bit-columns are created for the alignment gap character () and the missing data character (?).

A `--data_type <nucleotide>` directive informs MyESL to treat A, T, C, G, and U as valid characters without case sensitivity. All other characters will be treated as missing data. Similarly, the `<protein>` option treats all unambiguous IUPAC amino acid letters (case insensitive) as valid characters. The `<molecular>` option provides a way to use both nucleotide and acid letters as valid characters, allowing for the mixing of two data types. By default, however, MyESL will treat all letters (case-sensitive) and digits as distinct characters, which makes MyESL useful for analyzing other types of molecular data. For example, one may include information about the methylation status of positions, the presence/absence of genes or genomic segments, and other molecular characters in the input data files. Even non-molecular characteristics and features of taxa could be specified using letters and digits and then used as input during ESL analysis. When using this flexibility, we suggest users to be careful when preparing their input and interpreting the results.

MyESL compresses the resulting data matrix by discarding all monomorphic bit-columns across organisms selected for ESL analysis. That is, all the sequences at that position have the same character across taxa, except for the missing data and alignment gaps. Singleton bit-columns are also removed because only one alignment row contains an

unambiguous character state different from others. Such features will never be informative in the lasso analysis. To further reduce the number of features input to lasso, one may drop all bit-columns in which bit 1 appears fewer than a certain number of times (`--bit_ct <count>`), making MyESL computationally efficient.

2.4 Model building using LASSO regression

MyESL estimates the coefficients of the sparse group LASSO regression model by minimizing the logistic loss ((Liu *et al.*, 2011)), which is defined as:

$$L'(\beta) = l(\beta) + \lambda_1|\beta|_1 + \lambda_2 \sum_{g=1}^{G} w_g \|\beta_g\|_2. \quad (1)$$

Here, the first term is the logistic loss function, and the second is the penalty for including individual bitcolumns in the regression model. λ_1, the regularization parameter, penalizes the inclusion of bit-columns in the ESL model. β is the column vector of regression coefficients, with its norm $|\beta| = \sum_{i=1} |\beta_i|$ where i ranges from 1 to the number of bit-columns in the whole dataset. The third term penalizes the inclusion of groups into the regression model. Here, λ_2 is the group regularization parameter, and $\|\beta_g\| = \sum_{i=1} |\beta_{gi}|$, where i ranges from 1 to the number of bitcolumns in group g, and β_{gi} is the regression coefficient for the i^{th} feature in group g. The product of $\|\beta_g\|$ and group weight (w_g) is summed over all G groups. The group weight is usually the square root of the number of bit columns in group g, which MyESL assumes. Alternatively, the user can provide group weights via a text file (`--group_wt <filename.txt>`) containing two tab-separated columns: the first containing group names and the second containing the corresponding group weights.

In MyESL, λ_1 and λ_2 are the bit-column and group penalty parameters, respectively. These parameters are user-specified by setting `--lambda1 <float>` and `--lambda2 <float>` and vary between 0 and 1. MyESL also allows performing LASSO regression without group sparsity using the flag `--no_group_penalty` or providing only a single FastA file as data input.

2.5 Sparse group lasso implementation

The LASSO regression analysis engine of MyESL is programmed in C++ for computational efficiency and portability across platforms. The C++ source code is a direct port of the MATLAB code for the sgLogisticR and other functions of SLEP, which were used by Kumar and Sharma (2021). The `sgLogisticR`, and thus MyESL, optimizes the regression weights by employing Moreau-Yosida regularization (Liu and Ye, 2010; Liu *et al.*, 2011) and minimizes the logistic loss for sparse group logistic lasso regression. MyESL uses the armadillo library in C++ for linear algebra and scientific computing (Sanderson and Curtin, 2016), which can employ multiple computing cores for matrix operations.

2.6 ESL Model output, cross-validation, and predictions

MyESL uses supervised machine learning to produce an ESL model for the given hypothesis. In this process, ESL estimates regression coefficients $\beta's$ for each bit-position in a one-hot encoded matrix, where most of these coefficients will be zero ($\beta = 0$) due to sparsity constraint. MyESL has a Python function (`MyESL_model_apply.py`) to utilize a pre-trained ESL model, specifically, the feature (bit columns) weights or regression coefficients produced. This function can classify a new set of organisms whose sequences are aligned with the data used to build the model. Each test organism is assigned a prediction score and a probability (0 to 1) output in a tab-separated text. A prediction score greater than zero or a probability exceeding 0.5 indicates classification into the class labeled as $+1$.

Users can also build a pre-trained ESL model for classification by performing cross-validation. MyESL optimizes regression weights for multiple training sets and validates the classification accuracy using the held-out sets. The cross-validation in MyESL is performed using the directive `--kfold <int>`. For example, 80% of the taxa are used in model training, while 20% of taxa are withheld for validation if k-fold is set at 5. MyESL produces feature weights from the training samples and classification accuracy for each holdout sample, and users can choose the model with the highest accuracy or use other criteria (e.g., root mean square error). Cross-validation in MyESL can select the best pair of sparsity parameters (Chetverikov *et al.*, 2021) or assess model accuracy without using test data (Xu and Goodacre, 2018). One should avoid cross-validation if the focal clade has only a few members.

2.7 Building multiple ESL models

Building multiple models with the same feature/response data is a common practice in machine learning to select an optimal pair of sparsity parameters or achieve model averaging. MyESL allows building multiple ESL models by performing a grid search over the regularization parameter space. The grid search option lets users specify the bit (`--lamba1_grid <float, float, float>`) and group (`--lamba1_grid <float, float, float>`) sparsity parameters by defining the minimum [0-1], maximum [0-1], and step size [0-1] of the parameter space. MyESL can prematurely terminate the grid search process to avoid building overly sparse models that may result in high sparsity parameter values. The `--min_group_ct <int>` option sets the minimum number of groups threshold to set the limit of grid search.

2.8 ESL output of evolutionary parameters

MyESL processes the regression coefficient and produces a series of result files containing different sparsity scores. These are tab-separated text files generated using the following directive `--stats_out <PGHS>`. Different letters in the input string for the directive will produce the corresponding results as follows: P: Position Sparsity Scores; G: Group Sparsity Scores; H: Hypothesis Sparsity Scores; S: A file containing both Species Prediction Score (SPS) and Species Prediction Probability (SPP) (see details in Kumar and Sharma (2021)).

2.9 An Application of MyESL

We analyzed a fungi dataset containing amino acid sequence alignments of $1,233$ genes from 86 yeast species (Shen *et al.*, 2016, 2017). We built an ESL model for a clade (44 species), where all species received a $+1$ label, while the remaining 42 species were assigned -1. The combined sequence alignments from all genes contained $609,013$ sites, and the total number of bit columns was $4,105,444$, distributed among $1,233$ groups. We used weighted class balance and set the position and group sparsity parameter values at 0.1 and 0.2, respectively. MyESL took 5.25 minutes to read and pre-process input datasets and less than 1 minute to build the ESL model and process result files. The peak memory usage for this analysis was 1.3 GB.

The resulting ESL model contained 33 genes ($< 3\%$), 3,745 positions ($< 1\%$), and $5,608$ bit-columns (1.3%). Regression coefficients were normally distributed (Fig. 1b). Two genes were much more important than others (Fig. 1c). Interestingly, one species (*A. rubescence*) in the focal clade received a low classification probability (0.05). Based on such an observation, Sharma and Kumar (2024) introduced a novel approach (DrPhylo) to detect fragile clades and causal gene-species combinations, implemented using MyESL. DrPhylo analysis can be conducted in MyESL using the `--DrPhylo` directive. MyESL produces

a two-dimensional visualization (species x genes) for this analysis that reveals causal gene-species combinations (Sharma and Kumar, 2024).

2.10 Distributions

The source codes (Python and C++) for all custom functions used in MyESL are freely available and distributed using a GitHub repository `https://github.com/kumarlabgit/MyESL`. The repository contains all instructions for installing MyESL for a Python environment in Linux and performs MyESL analysis for a clade in an example phylogeny using empirical sequence alignments. We have also packaged all of these utilities of MyESL in a standalone Windows executable (.exe) file, MyESL.exe, which is also distributed through the same GitHub repository `https://github.com/kumarlabgit/MyESL/tree/win10`. Using this executable, we are linking MyESL with DrPhylo mode to MEGA via its AppLinker interface, which will make MyESL capabilities directly accessible to users with one click when they are viewing the inferred phylogeny in MEGA's Tree Explorer.

3 Conclusion

MyESL is open-source, extensible, portable, efficient, and lightweight software that provides all the necessary utilities for researchers interested in using the ESL approach in molecular evolutionary and functional genomics. While a few generic packages are available for conducting sparse group lasso (Yang and Zou, 2015; Zeng and Breheny, 2017; Simon *et al.*, 2013; Klosa *et al.*, 2020; Civieta *et al.*, 2021), they are neither optimal nor efficient for handling large phylogenomic datasets, building phylogenetic hypotheses, achieving phylogeny-aware class balancing, and domain-specific post-processing of model outputs. Some of these packages cannot be compiled into standalone applications or proprietary, making their integration into GUI applications infeasible. These limitations have been overcome by developing MyESL.

In the above, we focused on ESL's use to build models for organismal phylogeny for brevity, but classes in ESL can contain sequences of duplicated genes or those carrying different traits. For instance, we have used MyESL to implement ESL methods to identify proteins associated with the evolution of convergent traits (e.g., echolocation) in mammals (Allard *et al.*, 2024) and identify fragile clades and associated sequences in phylogenomics (Sharma and Kumar, 2024). The use of MyESL produced highly influential positions and groups and predictive models for downstream analyses in these applications. In the future, we aim to equip MyESL with additional sparse learning methods, such as overlapping groups and tree-structured lasso, making it even more useful for data-driven molecular evolution and functional genomics discoveries.

Conflict of interest

None decleared.

Acknowledgments

We thank Alessandra Lamarca, John Allard, Hardik Sharma, Brandon Son, and Sikha Singh for their comments and for testing different versions of MyESL.

Funding

This work was supported by a research grant from the National Institutes of Health to SK (R35GM139540-04).

4 Data availability

An example Fungi dataset, including amino acid sequence alignments and a phylogenetic hypothesis in a text file, is available in GitHub, which can be accessed from `https://github.com/kumarlabgit/MyESL`. This dataset was originally published by Wickett *et al.* (2014) and analyzed by Shen *et al.* (2017) and Sharma and Kumar (2024).

References

Allard, J. B. *et al.* (2024). Evolutionary sparse learning reveals the shared genetic basis of convergent traits. *Nat. Comm. (revision invited)*.

Chetverikov, D. *et al.* (2021). On cross-validated lasso in high dimensions. *Ann. Stat.*, **49**(3), 1300–1317.

Civieta, Á. M. *et al.* (2021). asgl: A python package for penalized linear and quantile regression. *arXiv:2111.00472*.

Klosa, J. *et al.* (2020). Seagull: lasso, group lasso and sparse-group lasso regularization for linear regression models via proximal gradient descent. *BMC bioinformatics*, **21**(1), 407.

Kumar, S. and Sharma, S. (2021). Evolutionary sparse learning for phylogenomics. *Mol. Biol. Evol.*, **38**(11), 4674–4682.

Liu, J. and Ye, J. (2010). Moreau-Yosida regularization for grouped tree structure learning. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 2*, pages 1459–1467. Curran Associates Inc., NY.

Liu, J. *et al.* (2011). SLEP: Sparse learning with efficient projections. *Note[Online]*. 6:491. `http://yelabs.net/software/SLEP/manual.pdf`.

Qiao, L. *et al.* (2017). A systematic review of structured sparse learning. *Front. Inform. Technol. Electron. Eng*, **18**(4), 445–463.

Sanderson, C. and Curtin, R. (2016). Armadillo: a template-based c++ library for linear algebra. *J. Open Source Softw.*, **1**(2), 26.

Sharma, S. and Kumar, S. (2024). Discovering fragile clades and causal sequences in phylogenomics by evolutionary sparse learning. *Mol. Biol. Evol. (revision in review)*. `https://doi.org/10.1101/2024.04.26.591378`.

Shen, X. X. *et al.* (2016). Reconstructing the backbone of the saccharomycotina yeast phylogeny using genome-scale data. *G3: Genes, Genomes, Genetics*, **6**(12), 3927–3939.

Shen, X.-X. *et al.* (2017). Contentious relationships in phylogenomic studies can be driven by a handful of genes. *Nat. Eco. Evol.*, **1**(5), 126.

Simon, N. *et al.* (2013). A sparse-group lasso. *J. Comput. Graph. Stat.*, **22**(2), 231–245.

Tamura, K. *et al.* (2021). Mega11: molecular evolutionary genetics analysis version 11. *Mol. Biol. Evol.*, **38**(7), 3022–3027.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *J. Stat. Soc. Series B Stat. Methodol.*, **58**(1), 267–288.

Wickett, N. J. *et al.* (2014). Phylotranscriptomic analysis of the origin and early diversification of land plants. *Proceedings of the National Academy of Sciences*, **111**(45), 4859–4868.

Xu, Y. and Goodacre, R. (2018). On splitting training and validation set: A comparative study of Cross-Validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning. *J Anal Test*, **2**(3), 249–262.

Yang, Y. and Zou, H. (2015). A fast unified algorithm for solving group-lasso penalize learning problems. *Stat. Comput.*, **25**(6), 1129–1141.

Zeng, Y. and Breheny, P. (2017). The biglasso package: A memory-and computation-efficient solver for lasso model fitting with big data in r. *arXiv:1701.05936*.