L0Learn: A Scalable Package for Sparse Learning using L0 Regularization

Hussein Hazimeh
Google Research

Rahul Mazumder
Massachusetts Institute of Technology

Tim Nonet
Massachusetts Institute of Technology

Editor: Alexandre Gramfort

Abstract

We present L0Learn: an open-source package for sparse linear regression and classification using \( \ell_0 \) regularization. L0Learn implements scalable, approximate algorithms, based on coordinate descent and local combinatorial optimization. The package is built using C++ and has user-friendly R and Python interfaces. L0Learn can address problems with millions of features, achieving competitive run times and statistical performance with state-of-the-art sparse learning packages. L0Learn is available on both CRAN and GitHub.¹

Keywords: sparsity, sparse regression, sparse classification, \( \ell_0 \) regularization, coordinate descent, combinatorial optimization

1. Introduction

High-dimensional data is common in many important applications of machine learning, such as genomics and healthcare (Bycroft et al., 2018). For statistical and interpretability reasons, it is desirable to learn linear models with sparse coefficients (Hastie et al., 2015). Sparsity can be directly obtained using \( \ell_0 \) regularization, which controls the number of nonzero coefficients in the model. To illustrate this, let us consider the standard linear regression problem with a data matrix \( X \in \mathbb{R}^{n \times p} \), regression coefficients \( \beta \in \mathbb{R}^p \), and a response vector \( y \in \mathbb{R}^n \). Using \( \| \beta \|_0 \) to denote the number of nonzero entries in \( \beta \), the \( \ell_0 \)-regularized least-squares problem is given by:

\[
\min_{\beta} \frac{1}{2}\|y - X\beta\|_2^2 + \lambda \|\beta\|_0,
\]

where \( \lambda \geq 0 \) is a regularization parameter. Problem (1) is a fundamental problem in statistics and machine learning known as best subset selection, since it searches for a subset of the features that leads to the best fit (in terms of squared error) (Hocking and Leslie, 1967). Statistical optimality properties of estimator (1) have been discussed in Greenshtein (2006); Raskutti et al. (2011); Zhang et al. (2014). Problem (1) is NP-hard and poses computational challenges. Various approaches have been proposed to approximate solutions to (1).

¹Links: https://cran.r-project.org/package=L0Learn and https://github.com/hazimehh/L0Learn
These include continuous proxies to the $\ell_0$ norm such as the $\ell_1$ norm (Tibshirani, 1996) and nonconvex penalties such as SCAD and MCP (Fan and Li, 2001; Zhang, 2010). For these continuous penalties, specialized software packages have been developed, e.g., glmnet (Friedman et al., 2010), ncvreg (Breheny and Huang, 2011), sparsenet (Mazumder et al., 2011), and picasso (Ge et al., 2019). Approaches for cardinality-constrained problems include greedy heuristics (e.g., stepwise methods), iterative hard thresholding (IHT) (Blumensath and Davies, 2008), BeSS (Wen et al., 2020), abess (Zhu et al., 2020), among others.

Recently, there have been significant advances in computing globally optimal solutions to $\ell_0$-regularized problems, by using mixed integer programming (Wolsey and Nemhauser, 1999)—see for example Bertsimas et al. (2016); Bertsimas and Van Parys (2020); Hazimeh et al. (2022), and references therein. For example, $\ell_0$-regularized regression problems can be solved to optimality in minutes to hours for $p \sim 10^7$ when highly sparse solutions are desired (Hazimeh et al., 2022). In several applications however, high-quality approximate solutions for $\ell_0$-regularized problems may be more practical than obtaining global optimality certificates. Our recent works (Hazimeh and Mazumder, 2020; Dedieu et al., 2021) propose fast, approximate algorithms for computing high-quality solutions to $\ell_0$-regularized problems, with running times comparable to that of fast $\ell_1$ solvers. Our algorithms are based on a combination of coordinate descent (CD) and local combinatorial optimization, for which we establish convergence guarantees. Local search often results in improved solution quality over using CD alone. Indeed, the experiments in Hazimeh and Mazumder (2020); Dedieu et al. (2021) indicate that our algorithms can outperform state-of-the-art methods based on $\ell_1$, MCP, SCAD, IHT, and others in terms of different statistical metrics (prediction, estimation, variable selection) and across a wide range of statistical settings.

This paper introduces L0Learn: a package for $\ell_0$-regularized linear regression and classification. In addition to the $\ell_0$ penalty, L0Learn supports continuous penalties such as the $\ell_1$ or squared $\ell_2$ norm. L0Learn implements highly optimized CD and local combinatorial optimization algorithms (Hazimeh and Mazumder, 2020; Dedieu et al. 2021) in C++, along with user-friendly R and Python interfaces to support fitting and visualizing models.

2. Package Overview

**Problem formulation.** In L0Learn, we consider a supervised learning setting with samples $\{(x_i, y_i)\}_{i=1}^n$, where $x_i \in \mathbb{R}^p$ is the $i$-th feature vector and $y_i \in \mathbb{R}$ is the corresponding response. Let $L: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ be an associated loss function (such as squared error or logistic loss). L0Learn approximately minimizes the empirical risk, penalized with an $\ell_0\ell_1$ or $\ell_0\ell_2$ regularization. Specifically, for a fixed $q \in \{1, 2\}$, we compute approximate solutions to

$$\min_{\beta_0, \beta} \sum_{i=1}^n L(y_i, \beta_0 + x_i^T \beta) + \lambda \|\beta\|_0 + \gamma \|\beta\|_q^q,$$

where $\lambda$ and $\gamma$ are non-negative regularization parameters. While the $\ell_0$ penalty performs variable selection, the $\ell_q$ regularization induces shrinkage to help mitigate overfitting, especially in low-signal settings (Mazumder et al., 2023; Hazimeh and Mazumder, 2020). L0Learn supports regression using squared-error loss, classification using logistic and squared-hinge losses; and also accommodates box constraints on the $\beta_i$s.
# Assume the data matrix (x) and response (y) have been loaded
fit <- L0Learn::fit(x, y, penalty="L0") # Fit an L0 regularized regression model
plot(fit) # Plot the regularization path
cv_fit <- L0Learn::cvfit(x, y, penalty="L0", nFolds=5) # 5-fold cross validation
plot(cv_fit) # Plot the cross-validation error

---

Figure 1: Simple examples of using L0Learn in R.

**Overview of the algorithms.** L0Learn uses a combination of (i) cyclic CD and (ii) local combinatorial optimization. The choice of CD is inspired by its strong performance in sparse learning with continuous penalties (Friedman et al., 2010; Mazumder et al., 2011). Standard convergence results for cyclic CD, e.g., Tseng (2001), do not apply for the discontinuous objective in (2). Hence, in Hazimeh and Mazumder (2020), we show that a variant of cyclic CD converges to a local minimizer of Problem (2). Given a CD solution ($\hat{\beta}_0, \hat{\beta}$), the local combinatorial optimization algorithm searches a local neighborhood for solutions with a better objective.\(^2\) After finding a new improved solution using local search, we run CD with the new solution in an attempt to further improve the current solution. We keep iterating between local search and CD, until convergence—see Hazimeh and Mazumder (2020) for details.

**Efficient computation of the regularization path.** L0Learn solves (2) over a grid of regularization parameters. We use computational schemes such as warm starts, active sets, greedy cyclic order for CD, and correlation screening (Hazimeh and Mazumder, 2020), to speed up the algorithm. Due to the nature of the $\ell_0$ penalty, different values of $\lambda$ in (2) can lead to the same solution. Thus, to avoid duplicate solutions and unnecessary computations, we develop a new method that computes a data-dependent grid of $\lambda$s, which is guaranteed to avoid duplicate solutions—see Hazimeh and Mazumder (2020) for details.

**Implementation and development.** All the algorithms in the package are implemented in C++, along with high-level R and Python interfaces. For linear algebra operations, we rely on the Armadillo library (Sanderson and Curtin, 2016), which is accelerated by Basic Linear Algebra Subprograms (BLAS) (Lawson et al., 1979). The R interface is integrated with C++ using RcppArmadillo (Eddelbuettel and Sanderson, 2014). The Python interface is integrated with C++ using carma (Urlus, 2023) and pybind11 (Jakob et al., 2017). L0Learn supports both dense and sparse data matrices. Sparse matrices generally speed up computation and reduce memory requirements. In all functions that require the data matrix as an input argument, we use function templates that accept a generic matrix type. Thus, exactly the same code is used for both dense and sparse matrices, but each matrix type uses specialized linear algebra implementations.

For development, we use continuous integration based on Travis CI to build and test the package. Our unit and integration tests are primarily implemented in R based on testthat (Wickham, 2011) and have a coverage of 97% (as measured by covr). L0Learn achieved the highest rating (A) for code quality by the code analysis tool Codacy (www.codacy.com). We have additional CI checks that use Github Actions for our Python bindings to ensure the Python bindings interact with the C++ library identically to the R bindings.

---

\(^2\)We implement the case where the neighborhood consists of all solutions obtained by removing one variable from the support of $\beta$ and adding another previously zero variable to the support.
3. Usage and Documentation

L0Learn can be installed in R by executing `install.packages("L0Learn")` and can be installed in Python by executing `pip install l0learn`. It is supported on Linux, macOS, and Windows. In Figure 1, we provide simple examples of how to use L0Learn in R. More elaborate examples and a full API documentation can be found in L0Learn’s Vignette and Reference Manual, which are available on L0Learn’s main pages\(^1\). Similar examples and API documentation are available in Python as well.

4. Experiments

Our main goal in these experiments is to compare the running time of L0Learn with similar toolkits, designed for sparse learning problems. Specifically, we compare with `glmnet`, `ncvreg`, `picasso`, and `abess`. For space constraints, we focus on linear regression, and refer the reader to Dedieu et al. (2021) for sparse classification experiments. Our experiments also shed some light on the statistical performance of the different approaches: for in-depth studies of statistical properties, see Hazimeh and Mazumder (2020); Hastie et al. (2020); Mazumder et al. (2023). We note that there are modern approaches for solving \(\ell_0\)-regularized problems to global optimality e.g., Bertsimas and Van Parys (2020); Hazimeh et al. (2022, 2023). However, due to their focus on optimality certification, they are usually (much) slower than the competing toolkits we present here (Hazimeh et al., 2022).

**Setup.** Following Hazimeh and Mazumder (2020), we consider synthetic data as per a linear regression model under the fixed design setting (exponential correlation model with \(\rho = 0.3\)). We take \(n = 1000\), \(k = 50\), signal-to-noise ratio (SNR)\(^3\) to be 5 and vary \(p \in \{10^3, 10^4, 10^5\}\). In L0Learn, we used the default CD algorithm with the \(\ell_0\ell_2\) penalty. In `picasso`, we used \(\ell_1\) regularization and changed the convergence threshold (prec) to \(10^{-10}\) so that its solutions roughly match those of `glmnet`. In `ncvreg`, we used the (default) MCP penalty. All competing methods are tuned to minimize MSE on a validation set with the same size as the training set. In L0Learn, `ncvreg`, and `abess`, we tune over a two-dimensional grid consisting of 100 \(\lambda\) values (chosen automatically by the toolkits) and 100 \(\gamma\) values (in the range \([10^{-2}, 10^2]\) for L0Learn, \([1.5, 10^3]\) for `ncvreg`, \([1, 10^3]\) for `abess`\(^4\)). Experiments were performed on a Linux c5n.2xlarge EC2 instance running R 4.0.2.

**Metrics.** We report the running time in seconds for computing a path of 100 solutions. (L0Learn, `abess`, and `ncvreg` require additional time for tuning their second parameter \(\gamma\)). In addition, given an estimator \(\hat{\beta}\), we compute the following statistical metrics: (i) prediction error (PE) given by \(\|X\hat{\beta} - X\beta^*\|_2/\|X\beta^*\|_2\), (ii) the number of false positives (FP), i.e., the nonzero variables in \(\hat{\beta}\) that are not in \(\beta^*\), and (iii) the support size (SS).

**Results.** In Table 1, we report the metrics averaged over 10 repetitions. The results indicate that L0Learn achieves best-in-class run times and statistical performance. While we only show results on synthetic data sets due to space constraints, L0Learn is useful and can lead to good improvements on real data sets, as demonstrated in many studies,

\(^3\)Similar to Bertsimas et al. (2016), SNR := Var(\(X\beta^*\))/\(\sigma^2\).

\(^4\)We found out that `abess` requires a different range for \(\gamma\) (the parameter of the squared \(\ell_2\) regularizer) to perform well. This is expected since `abess` considers a different, cardinality-constrained formulation.
including Hazimeh and Mazumder (2020); OShea et al. (2021); Cao et al. (2021); Li et al. (2023).

5. Conclusion

We introduced L0Learn: a scalable package for \( \ell_0 \)-regularized regression and classification. The package is implemented in C++ along with R and Python interfaces. It offers two approximate algorithms: a fast coordinate descent-based method and a local combinatorial search algorithm that helps in improving solution quality. Our experiments indicate that L0Learn is highly scalable and can outperform popular sparse learning toolkits in important high-dimensional settings. The package has been successfully used in a variety of applications in healthcare and genetics; and has also proven to be effective in warm starting exact \( \ell_0 \) regularization solvers (Hazimeh et al., 2022). As a future direction, it would be interesting to study the performance of CD and local combinatorial search in (\( \ell_0 \)-based) pruning of neural networks (Benbaki et al., 2023).

Acknowledgements

We acknowledge research support from the Office of Naval Research and National Science Foundation.
References


