Fast Sparse Classification for Generalized Linear and Additive Models

Jiachang Liu[1] **Chudi Zhong**[1] **Margo Seltzer**[2] **Cynthia Rudin**[1]
[1]Duke University [2] University of British Columbia
{jiachang.liu, chudi.zhong}@duke.edu, mseltzer@cs.ubc.ca, cynthia@cs.duke.edu

Abstract

We present fast classification techniques for sparse generalized linear and additive models. These techniques can handle thousands of features and thousands of observations in minutes, even in the presence of many highly correlated features. For fast sparse logistic regression, our computational speed-up over other best-subset search techniques owes to linear and quadratic surrogate cuts for the logistic loss that allow us to efficiently screen features for elimination, as well as use of a priority queue that favors a more uniform exploration of features. As an alternative to the logistic loss, we propose the exponential loss, which permits an analytical solution to the line search at each iteration. Our algorithms are generally 2 to 5 times faster than previous approaches. They produce interpretable models that have accuracy comparable to black box models on challenging datasets.

1 INTRODUCTION

Our goal is to produce sparse generalized linear models or sparse generalized additive models from large datasets in under a minute, even in the presence of highly-correlated features. Specifically, our interest is in the following problem:

$$\min_{\boldsymbol{w}} \sum_{i=1}^{n} \ell(\boldsymbol{w}, \boldsymbol{x}_i, y_i) + \lambda_0 \|\boldsymbol{w}\|_0 \tag{1}$$

with the logistic loss

$$\ell(\boldsymbol{w}, \boldsymbol{x}_i, y_i) = \log\left(1 + e^{-y_i(\boldsymbol{w}^T \boldsymbol{x}_i)}\right)$$

or the exponential loss

$$\ell(\boldsymbol{w}, \boldsymbol{x}_i, y_i) = e^{-y_i(\boldsymbol{w}^T \boldsymbol{x}_i)}$$

where $\boldsymbol{x}_i \in \mathbb{R}^p$ is the i-th observation, and $y_i \in \{-1, 1\}$ is the label of the i-th data sample. The logistic loss tends to yield nicely calibrated probability estimates, which explains its broad appeal. The exponential loss, used in boosting, has been overlooked as an approach to sparse additive modeling, but like logistic regression, it also yields direct probability estimates. It has the advantage of analytical solutions for line search, dramatically improving convergence rates.

A small ℓ_2 regularization is used with the logistic loss to speed up convergence, as discussed later:

$$\min_{\boldsymbol{w}} \sum_{i=1}^{n} \ell(\boldsymbol{w}, \boldsymbol{x}_i, y_i) + \lambda_0 \|\boldsymbol{w}\|_0 + \lambda_2 \|\boldsymbol{w}\|_2^2. \tag{2}$$

We do not include ℓ_1: since we are looking for very sparse and accurate models, ℓ_1 regularization would degrade the quality of the solution compared to true sparsity regularization with ℓ_0. The ℓ_0 penalty term makes Problems (1) or (2) NP-hard.

Problems (1) or (2) can produce generalized additive models (Lou et al., 2016; Hastie and Tibshirani, 2017; Nori et al., 2019; Rudin et al., 2022) through a transformation of the input variables, replacing each continuous feature $x_{\cdot,j}$ with a set of dummy variables $\tilde{x}_{\cdot,j,\theta} = \mathbf{1}_{[x_{\cdot,j} \geq \theta]}$, for θ set to be each realized value of feature j in the dataset. Then, solving (1) or (2) yields a generalized additive model where component function j is a sum of the weighted dummy variables for feature j. This transformation yields a large feature set with many correlated features, but the approaches provided here can handle such sizes.

There are at least two general approaches for tackling these problem (besides relaxing the ℓ_0 term to ℓ_1 and suffering the associated bias). The first uses callbacks to a mathematical programming solver, such as a mixed-integer programming (MIP) solver (Sato et al.,

Proceedings of the 25th International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151. Copyright 2022 by the author(s).

2016; Ustun and Rudin, 2017; Sato et al., 2017; Bertsimas and King, 2017; Bertsimas et al., 2021; Ustun and Rudin, 2019). This approach can solve the problem exactly. However, it cannot handle large feature spaces or highly-correlated features. A solver might take several days or run out of memory on even a modestly-sized problem. The second approach to Problems (1) or (2) is to use coordinate descent with local swap operations for best subset search, similar to simulated annealing, Metropolis-Hastings, or other MCMC methods (Metropolis et al., 1953; Kirkpatrick et al., 1983; Del Moral et al., 2006). Our approach is of this second type, though it is important to note that a solution from our method could be used as a warm-start for one of the MIP solvers; a better warm-start is the key to finding optimal solutions faster with MIP.

There are two main steps per iteration in these types of algorithms: (i) coordinate descent steps involving a line search along the objective function, often using a local surrogate function, and (ii) local swaps, where the support set (the set of features permitted to have nonzero coefficients) changes over iterations. Our work advances both of these steps over previous work. For (i), we show that a natural surrogate for the logistic loss used in previous work leads to inefficiency, in that its step sizes are provably too conservative. We propose a more aggressive step. This opens up the possibility of using *cutting planes* or *quadratic cuts*. Cuts often help us rapidly prune the search space: by comparing the lower bound from the cuts with the current best loss, we are often able to prove that there is no possible step size we could take that would reduce our objective, in which case we will try a more promising direction in the search space. The ℓ_2 penalty term permits us to use quadratic cuts. When we do not want the ℓ_2 term (i.e., $\lambda_2 = 0$), we can use cutting planes. For (ii), we find that the order in which we evaluate features plays an important role, which has been previously overlooked. We use a priority queue to dynamically manage the order of evaluating features. The priority queue discourages us from checking features that are unlikely to change the model's support set, making the process of finding high-quality solutions more efficient.

In addition, for (i), improving the speed of the coordinate descent steps, we propose to use the exponential loss, which has a major advantage over the logistic loss in that the line search taken at each coordinate descent iteration has an analytical solution. Another appealing property of the exponential loss is that its probabilistic interpretation is extremely similar to that of logistic regression. Also, minimizing the exponential loss is known to provably maximize a proxy for the Area Under the ROC Curve (Ertekin and Rudin, 2011), making it an ideal choice for this problem.

Our contributions are:

1. We prove that previous work on surrogate loss optimization yields step sizes that are too conservative (Theorem 4.1).
2. When $\lambda_2 = 0$, we propose a linear cutting plane algorithm that prunes the search space by efficiently determining whether adding a feature could potentially reduce the objective.
3. With a small amount of ℓ_2 regularization, we propose a quadratic cut algorithm giving a tighter lower bound than the linear cutting plane algorithm.
4. We propose a method using the exponential loss, which is cleaner and simpler.
5. For more efficient best subset search, we use a priority queue to dynamically manage the order of checking features.

Our algorithms provide a dramatic improvement over previous approaches, often achieving the same results in less than half the time, and are able to produce models for thousands of features and observations in seconds. For instance, on the challenging FICO dataset from the 2018 Explainable Machine Learning Challenge, which, after the transformation to dummy variables, has 1,917 dummy features and 10K observations, we produce a generalized additive model of 19 total dummy variables, with performance comparable to black-box performance, in under 5 seconds.

2 BACKGROUND

Coordinate descent is popular in machine learning. Other techniques that use variations of it include AdaBoost (Freund and Schapire, 1997) and Sequential Minimal Optimization used for support vector machines (Platt, 1998). Surrogate functions are also common, e.g., they are used by Expectation Maximization (Dempster et al., 1977). We begin with background, following Patrascu and Necoara (2015) and Dedieu et al. (2021).

The loss function in Problem (2) can be rewritten as:

$$\mathcal{L}(\boldsymbol{w}) = G(\boldsymbol{w}) + \lambda_0 \|\boldsymbol{w}\|_0,$$

with $G(\boldsymbol{w}) = \sum_{i=1}^{n} \log(1+\exp(-y_i(\boldsymbol{x}_i^T \boldsymbol{w}))) + \lambda_2 \|\boldsymbol{w}\|_2^2$.

Let us optimize $\mathcal{L}(\boldsymbol{w})$ along coordinate j starting at point $\boldsymbol{w}^t$ at iteration t. Let $\nabla_j G(\boldsymbol{w}^t)$ denote the j-th component of the gradient of $G(\boldsymbol{w}^t)$, and let L_j be the Lipschitz constant for $\nabla_j G(\boldsymbol{w}^t)$. For any $d \in \mathbb{R}$:

$$|\nabla_j G(\boldsymbol{w}^t + \boldsymbol{e}_j d) - \nabla_j G(\boldsymbol{w}^t)| \leq L_j |d|$$

where $\boldsymbol{e}_j$ is a vector with all components equal to 0 except for the j-th component, which is equal to 1. A surrogate upper bound on $G(\boldsymbol{w}^t + \boldsymbol{e}_j d)$ is thus:

$$G(\boldsymbol{w}^t + \boldsymbol{e}_j d) \leq G(\boldsymbol{w}^t) + d\nabla_j G(\boldsymbol{w}^t) + \frac{1}{2}L_j d^2. \quad (3)$$

Instead of minimizing the original loss function with respect to coordinate j (as would be typical in coordinate descent), we can minimize this quadratic upper bound with the new coefficient $w_j^{t+1} = w_j^t + d$:

$$\hat{w}_j^{t+1} \in \arg\min_u \; G(\boldsymbol{w}^t) + (u - w_j^t)\nabla_j G(\boldsymbol{w}^t) + \frac{1}{2}L_j(u - w_j^t)^2 + \lambda_0 \mathbb{1}_{u \neq 0}.$$

Following previous work (Dedieu et al., 2021), we have an analytical solution for the above problem:

$$\hat{w}_j^{t+1} = T(j, \boldsymbol{w}) = \begin{cases} c, & \text{if } |c| \geq \sqrt{\frac{2\lambda_0}{L_j}} \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

where $c = w_j^t - \nabla_j G(\boldsymbol{w}^t)/L_j$.

If a solution cannot be improved by coordinate descent using this surrogate and thresholding function, we say this solution is *surrogate 1-OPT*, meaning that no single coordinate can be changed to improve the objective when using this surrogate for the line search.

As discussed earlier, local swap, add, and remove operations are useful for best subset search and other local search problems. These govern the support of the coefficient vector, determining which coefficients are permitted to be nonzero. We use S to denote the support of the feature vector; that is, the set of features that are permitted to have nonzero coefficients. We can swap some features in the current support, denoted by $S_1 \subseteq S$, for other features not in the support, denoted by $S_2 \subseteq S^c$. After each swap, we optimize the coefficients that are permitted to be nonzero.

To reduce computational cost, while evaluating a possible swap, we use an approximate evaluation procedure where we update only the coefficients of the swapped features and keep coefficients of other unswapped features fixed. If such a swap leads to a better loss, we add S_2 to the support, remove S_1 from the support, and update all coefficients for the features in the new support. We will focus on single feature swaps (i.e. $|S_1| = |S_2| = 1$) in this work. If no allowed swap appears to improve the loss, then we call the solution a *swap 1-OPT* solution.

3 OVERVIEW OF FAST SPARSE LOGISTIC REGRESSION

Let us focus on the logistic loss. Given an initial solution, we optimize one feature's coefficient at a time, and swap features within the support set to improve the solution. Our technique evaluates whether it could be worthwhile to swap two features. It is based on a theorem showing that thresholding from (4) yields step sizes that are too conservative. Using this information, we develop an algorithm that uses *quadratic cuts*. Typically, cutting planes (Kelley, 1960) are used in mathematical programming solvers, whereas here, we use cuts as part of efficient feature elimination within coordinate descent. Our second technique uses a priority queue to manage the search order for pairs of features to swap. At each outer iteration, we drop a feature j in the support and at each inner iteration, we evaluate adding a feature j'. The full pseudocode is in Appendix B. The main steps are:

1. **Remove and find alternatives.** According to the priority queue, try removing feature j from the current support. Find J' features outside the support as alternatives for feature j. These alternative features are picked according to orthogonal matching pursuit (Lozano et al., 2011). For each feature $j' \in J'$, we evaluate whether it is worthwhile to include it in our support as a replacement of feature j. This is done using the following procedures.

2. **Aggressive step.** Given a new feature j' that we may want to include in our support, we wish to find two values on opposite sides of the optimal coefficient $w_{j'}^*$. However, at current value $w_{j'}$, the thresholding results stay on a single side of the optimal value (as we will prove in Theorem 4.1). Thus, we take an aggressive step by going double the distance suggested by thresholding, or triple the distance if necessary. If this triple-sized step does not get to the opposite side of $w_{j'}^*$, we iteratively apply thresholding (4) to get a near-optimal coefficient and move to Step 6.

3. **Binary search.** Suppose we have found two values a and b on opposite sides of $w_{j'}^*$. We then perform one binary search step to get a point closer to $w_{j'}^*$ by setting c to be the midpoint, $c = \frac{1}{2}(a + b)$. If c is on the same side of a, we replace a with c; if not, we replace b with c. We use quadratic cuts (via the *Quadratic Cut Bound*, Theorem 4.3) at points a and b to obtain a lower bound on the objective for the optimal coefficient of the feature. In the case of no ℓ_2 regularization, we use cutting planes instead. More detail on this is in the next section.

4. **Eliminate.** If the lower bound is larger than the current best loss we have encountered so far, the new

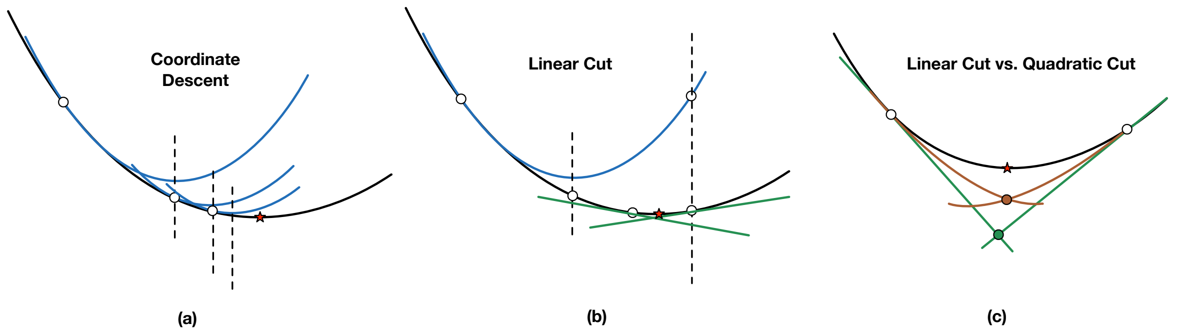

Figure 1: (a) We repeatedly apply coordinate descent until convergence to get the optimal coefficient (shown by the red star) and then calculate the loss. (b) We calculate a lower bound of the optimal loss by constructing two cutting planes. We can rule out the new feature if the lower bound of the loss from the cutting planes is larger than the best current loss. (c) Quadratic cuts (in red) form the lower bound instead and are tighter.

feature can be eliminated from consideration; we do not add this feature into our support. We move onto the next possible feature and start again from Step 2.

5. **Line search.** If the lower bound is smaller than the current best loss we have encountered, then feature j' could lead to a better solution. Therefore, we iteratively use thresholding (4) to obtain a near-optimal coefficient for the line search. (Alternatively, we could continue binary search for the minimum.)

6. **Complete the step.** We then calculate the loss with respect to this near-optimal coefficient for the line search. If the loss is higher than the current best loss, we eliminate this feature and move to the next best alternative feature; if the loss is lower, we add this new feature j' into the support to make up for the removed feature j and optimize all of the coefficients, completing a successful swap step.

7. **Update priority queue.** If no alternative feature can replace feature j, we add feature j back into the support and rate feature j less promising in our priority queue. This allows us to explore features that have a better chance of being swapped with an alternative feature next time.

4 SURROGATE QUADRATIC CUTS

Let us provide the theorem motivating our coordinate descent method for the logistic loss, which shows that the step sizes from thresholding in (4) are too conservative. Recall that thresholding is derived by minimizing a quadratic upper bound of the loss function. The coefficient of the quadratic function is the Lipschitz constant, which defines the maximum curvature the loss function can achieve. These connections imply:

Theorem 4.1. *(Thresholding is too conservative.) Let $\boldsymbol{w}^t$ be the current solution at iteration t, w_j^t be the coefficient for the j-th feature, and let w_j^* be the optimal value on the j-th coefficient while keeping all other coefficients fixed to their values at time t. Furthermore, let $\boldsymbol{w}^{t+1} = \boldsymbol{w}^t + \boldsymbol{e}_j(T(j, \boldsymbol{w}^t) - w_j^t)$, where $\boldsymbol{e}_j$ is a vector with 1 on the j-th component and 0 otherwise and $T(j, \boldsymbol{w}^t)$ is the thresholding operation with the support set fixed (i.e., $\lambda_0 = 0$). Then we have the following inequalities:*

$$\nabla_j G(\boldsymbol{w}^t)\nabla_j G(\boldsymbol{w}^{t+1}) \geq 0, \tag{5}$$

$$(w_j^t - w_j^*)(w_j^{t+1} - w_j^*) \geq 0, \tag{6}$$

$$\textit{and } G(\boldsymbol{w}^t) \geq G(\boldsymbol{w}^{t+1}). \tag{7}$$

This result shows that the thresholding operation will move the coefficient of the j-th feature closer to the optimal value w_j^* with a smaller loss value, as shown by (7). However, the coefficients before and after the thresholding operation will *always remain on the same side of* w_j^*, as shown by either (6) or (5). To see this, consider (6). We have two scalars of the same sign: $w_j^t - w_j^*$ and $w_j^{t+1} - w_j^*$. If w_j^{t+1} were on the opposite side of w^* than w_j^t, the product of these two scalars would instead be negative. Alternatively, by (5), if the slope of G at $\boldsymbol{w}^t$ is negative, the slope at $\boldsymbol{w}^{t+1}$ is also negative, indicating that we have not yet passed the minimum (of our convex logistic loss). Thus, this theorem indicates that the step size provided by the surrogate is too conservative; the distance is always too small to reach w_j^*. Figure 1 (left) illustrates this issue. The algorithm may make several steps before becoming sufficiently close to w_j^*.

Our technique chooses an aggressive step size that takes us beyond w_j^*, in order to use cuts to produce a lower bound on the loss at w^*. If the lower bound is too high, we can exclude the feature all together.

The first type of cut we introduce is classical *cutting planes*, which provide a linear lower bound on the loss. This can be used even if we have only ℓ_0 regularization on the logistic loss (i.e., if λ_2 in (2) is 0). With an additional ℓ_2 penalty term, we can obtain a strictly tighter lower bound on the loss, yielding quadratic cuts. We introduce both types of cuts next, starting with cutting planes.

Theorem 4.2. *(Classical cutting planes, not novel to this paper) Suppose $f(x)$ is convex and differentiable on domain $\mathbb{R}$. Let α_1 and α_2 be slopes of tangent lines of $f(x)$ at locations x_1 and x_2. If $\alpha_1\alpha_2 \leq 0$, there is a lower bound on the optimal value $f(x^*)$:*

$$f(x^*) \geq \frac{\alpha_1 f(x_2) - \alpha_2 f(x_1) + \alpha_1\alpha_2(x_1 - x_2)}{\alpha_1 - \alpha_2}. \quad (8)$$

This method originates from a first-order approximation of function $f(x)$. Figure 1(b) shows linear cuts.

With an additional ℓ_2 penalty term, we can obtain a strictly tighter lower bound on the loss via *quadratic cuts.* The ℓ_2 term makes $G(\boldsymbol{w})$ strongly convex, which means for any two points $\boldsymbol{w}$ and $\boldsymbol{w}'$ in the domain:

$$G(\boldsymbol{w}') \geq G(\boldsymbol{w}) + \nabla G(\boldsymbol{w})^T(\boldsymbol{w}' - \boldsymbol{w}) + \lambda_2\|\boldsymbol{w}' - \boldsymbol{w}\|_2^2.$$

Using this strongly convex property, we can tighten the lower bound given in Theorem 4.2 as follows:

Theorem 4.3. *(Quadratic Cut Bound) Suppose $f(x)$ is strongly convex and differentiable over $\mathbb{R}$ with λ_2 for the coefficient of the quadratic term. Let α_1 be the slope of the tangent line to $f(x)$ at location x_1. Then, there is a lower bound on the optimal value $f(x^*)$:*

$$f(x^*) \geq \mathcal{L}_{low} := f(x_1) - \frac{\alpha_1^2}{4\lambda_2}. \quad (9)$$

Let α_2 be the slope of the tangent line to $f(x)$ at another location x_2. If $\alpha_1\alpha_2 \leq 0$, a lower bound on the optimal value $f(x^*)$ is as follows:

$$f(x^*) \geq \mathcal{L}_{low} := f(\hat{x}) + \alpha_1(\hat{x} - x_1) + \lambda_2(\hat{x} - x_1)^2, \quad (10)$$
$$\hat{x} = \frac{-f(x_1) + f(x_2) + \alpha_1 x_1 - \alpha_2 x_2 - \lambda_2(x_1^2 - x_2^2)}{\alpha_1 - \alpha_2 - 2\lambda_2(x_1 - x_2)}.$$

Since this method originates from a second-order approximation of the function $f(x)$, we name this bound the Quadratic Cut Bound. Either this bound or cutting planes helps us decide when not to include a potential feature in our support, even without knowing its optimal coefficient from the line search.

5 FAST SPARSE CLASSIFICATION WITH EXPONENTIAL LOSS

Let us now switch from logistic loss to the exponential loss, optimizing:

$$\min_{\boldsymbol{w}} \left[\sum_{i=1}^{n} \exp(-y_i \boldsymbol{w}^T \boldsymbol{x}_i) + \lambda_0\|\boldsymbol{w}\|_0\right].$$

Though exponential loss typically is not used for sparse classification, it has no clear disadvantages over the logistic loss and even has several advantages. First we point out that exponential loss and logistic loss have remarkably *similar probabilistic interpretations* under the assumption that we have captured the correct set of features. While logistic regression estimates conditional probabilities as $\hat{P}_{\text{logistic}}(y = 1|\boldsymbol{x}) = \frac{e^{f(\boldsymbol{x})}}{1+e^{f(\boldsymbol{x})}}$ where $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}$, the exponential loss has almost the same probabilistic model: $\hat{P}_{\text{exp loss}}(y = 1|\boldsymbol{x}) = \frac{e^{2f(\boldsymbol{x})}}{1+e^{2f(\boldsymbol{x})}}$. Thus, both loss functions are equally relevant for modeling conditional probabilities.

The main benefit of exponential loss is that it has an *analytical solution for the line search* at each iteration when features are binary (-1 and 1). This avoids the necessity for cutting planes, quadratic cuts, or even surrogate upper bounds. Following the derivation of AdaBoost as a coordinate descent method (Schapire and Freund, 2013), its line search solution follows the formula $\frac{1}{2}\ln\left(\frac{1-d_-}{d_-}\right)$, where d_- indicates the weighted misclassification error of the feature chosen at iteration t (here we are interpreting each weak classifier as an individual feature, and the weak learning algorithm picks one of these features per iteration). AdaBoost's weight update step avoids calculation of the exponential loss at each iteration, and the full procedure is extremely efficient. (The main difference between our method and this reduced version of AdaBoost is that AdaBoost is not designed to yield sparse models.) In the following theorem, we provide a condition under which our method would decline to add a new feature at iteration t, because it does not provide an overall benefit to our objective. We use $\boldsymbol{z}_i \in \mathbb{R}^p$ with $\boldsymbol{z}_i = y_i\boldsymbol{x}_i$ to succinctly represent the product between y_i and $\boldsymbol{x}_i$. The objective can be then rewritten as:

$$\min_{\boldsymbol{w}}[H(\boldsymbol{w}) + \lambda_0\|\boldsymbol{w}\|_0]$$

where $H(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n}\exp(-\boldsymbol{w}^T\boldsymbol{z}_i)$.

Theorem 5.1. *Let $\boldsymbol{w}^t$ be the coefficient vector at iteration t, $H^t := H(\boldsymbol{w}^t)$ and λ_0 be the regularization constant for the ℓ_0 penalty. For the j-th coordinate, we update the coefficient according to:*

(1) Suppose $w_j^t = 0$. Let $d_- = \sum_{i:z_{ij}=-1}c_i / \sum_{i=1}^{n}c_i$, with $c_i = \exp(-(\boldsymbol{w}^t)^T\boldsymbol{z}_i)$. If d_- is within the interval:

$$\left[\frac{1}{2} - \frac{1}{2H^t}\sqrt{\lambda_0(2H^t - \lambda_0)}, \frac{1}{2} + \frac{1}{2H^t}\sqrt{\lambda_0(2H^t - \lambda_0)}\right],$$

then set w_j^{t+1} to 0. Otherwise set $w_j^{t+1} = \frac{1}{2}\ln\frac{1-d_-}{d_-}$.

(2) Suppose $w_j^t \neq 0$. Let $D_- = \sum_{i:z_{ij}=-1}c_i / \sum_{i=1}^{n}c_i$, with $c_i = \exp(-(\boldsymbol{w}^t - w_j^t\boldsymbol{e}_j)^T z_i)$. Let $H_{\neg j}^t = H(\boldsymbol{w}^t -$

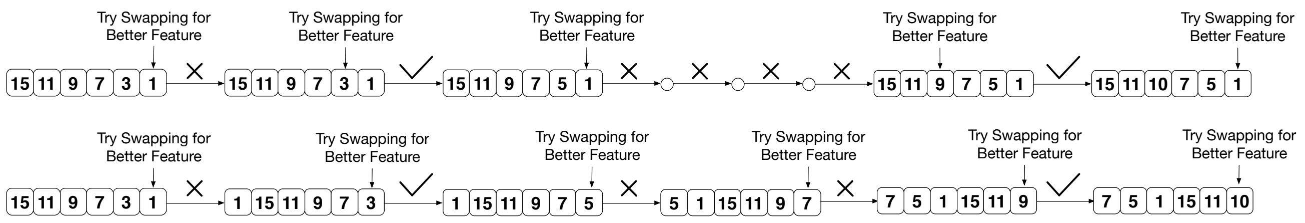

Figure 2: Sequential Ordering vs. Dynamic Ordering. Upper: We check each feature sequentially. Whenever we find a better feature, we always start from the beginning to find the next possible swap. Lower: We order the list, checking the feature that has failed the least amount of times first. We hold off checking less promising features until the end, saving substantial computational time.

$w_j^t \boldsymbol{e}_j$). *If D_- is within the interval:*

$$\left[\frac{1}{2}-\frac{1}{2H_{\neg j}^t}\sqrt{\lambda_0(2H_{\neg j}^t-\lambda_0)}, \frac{1}{2}+\frac{1}{2H_{\neg j}^t}\sqrt{\lambda_0(2H_{\neg j}^t-\lambda_0)}\right],$$

then set w_j^{t+1} to 0. Otherwise, set $w_j^{t+1} = \frac{1}{2}\ln\frac{1-D_-}{D_-}$.

Another potential benefit of the exponential loss is that it is a surrogate for the AUC, i.e., Area Under the ROC Curve (Ertekin and Rudin, 2011). Thus, we have reason to expect good AUC performance when optimizing the exponential loss.

6 DYNAMIC FEATURE ORDERING

Now that we can optimize along the coordinates using either logistic loss (Sections 3 and 4) or exponential loss (Section 5), we discuss the important swap steps that help the algorithm drop features that have promising swap candidates. As stated in Section 3, after coordinate descent is run until a local minimum is reached, we alternate between coordinate descent steps and swap steps. The technique proposed here is broadly applicable and can improve the speed not only for the logistic loss and the exponential loss but also for the squared loss in linear regression (see Appendix D.1).

We focus on the swap 1-OPT solutions (i.e., $|S_1| = |S_2| = 1$). The order of checking features in S_1 for possible swaps is key to improving the efficiency. Instead of checking features in S_1 sequentially based on feature indices (Dedieu et al., 2021), we dynamically order these features via a priority queue. We provide an example in Figure 2 to illustrate the key difference between the two approaches.

Suppose we have an initial solution with support on features $1, 3, 7, 9, 11,$ and 15, and features 3 and 9 are suboptimal. We can swap feature 3 with feature 5 and feature 9 with feature 10 to get a lower total loss. The first method checks features sequentially and always starts from the first index in the support after a successful swap. The algorithm terminates if we have checked all features without making any swaps. This method implicitly assumes that each feature in the support has an equal probability of having a successful swap. However, a feature that has not been swapped for many iterations is likely to be important and therefore unlikely to be swapped in the near future. It is better to check more promising features first.

To achieve this, we record how many times a feature has failed to swap. The features are ranked in ascending order of the number of failure times. Features that have never been checked are kept at the top of our priority queue. This local search process terminates when all features have been evaluated (i.e., the full priority queue) without making a successful swap. This accelerates the process to reach a swap 1-OPT solution.

7 EXPERIMENTS

Our evaluation answers the following questions: (1) How well do our early pruning technique, priority queue ordering, and proposed exponential loss perform in terms of run time relative to the state-of-the-art? (§7.1) (2) How well do our methods perform in terms of AUC, accuracy, and sparsity relative to state-of-the-art algorithms on simulated and real datasets? (§7.2)

We compare our methods to ℓ_1 regularized logistic regression (LASSO) via the *glmnet* package (Friedman et al., 2010), MCP via the *ncvreg* package (Breheny and Huang, 2011), and L0Learn (Dedieu et al., 2021). We use the fast C++ linear algebra libraries of L0Learn in our implementation. For all datasets, we run 5-fold cross validation and report the mean and standard deviation. Appendix C presents the experimental setup, datasets, and evaluation metrics, and Appendix D presents additional experimental results. Our methods are denoted as LogRegQuad-L0 (logistic loss and quadratic cuts) and Exp-L0 (exponential loss).

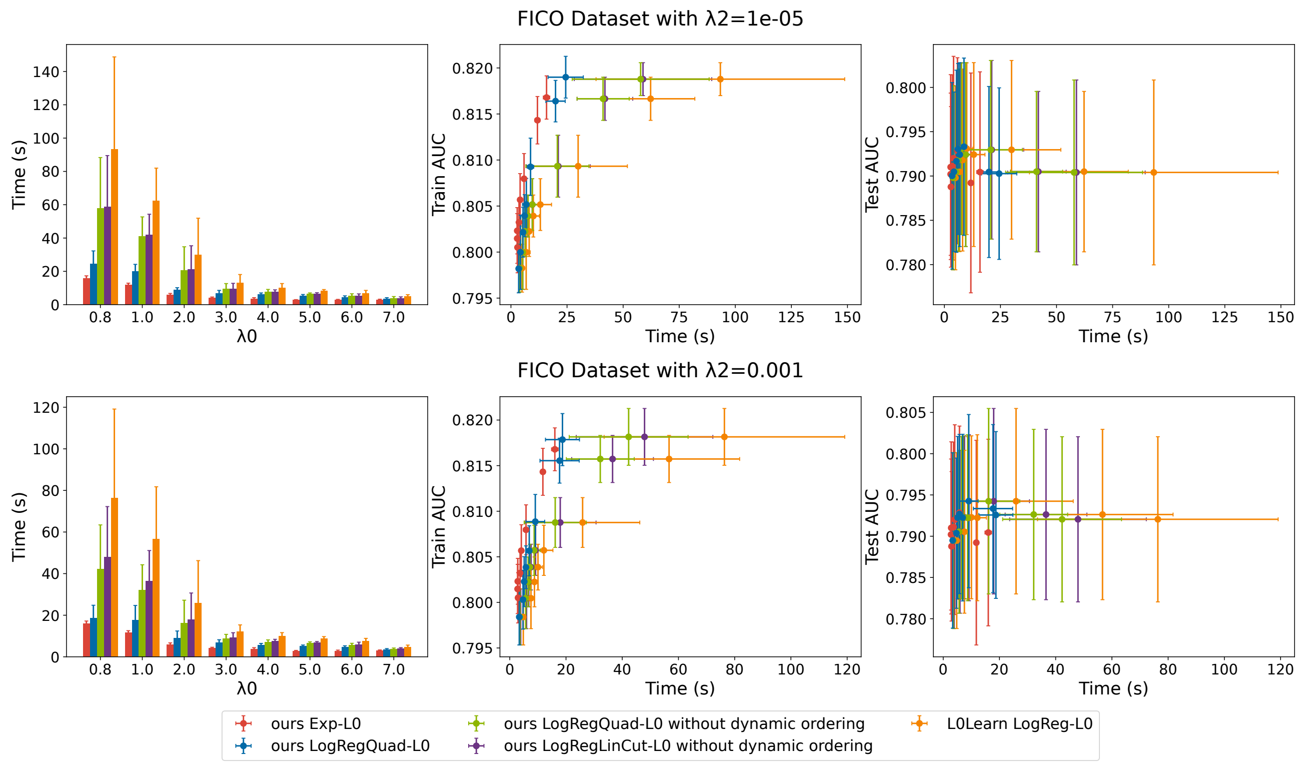

Figure 3: Computational times of different methods. "Exp" stands for exponential loss, "LogReg" stands for logistic loss, "LinCut" stands for linear cuts, and "Quad" stands for quadratic cuts. Note that there is no ℓ_2 penalty for the exponential loss. *Our Exp-L0 method is generally about 4 times faster than L0Learn.* Note that the AUC axes indicate practically similar performance for these particular methods; the training time is what differentiates the methods. Additionally, when the ℓ_2 penalty increases from $\lambda_2 = 1\text{e}-05$ to $\lambda_2 = 0.001$, there is a computational speedup from using the linear cut to the quadratic cut due to the tighter lower bound.

7.1 Computational Efficiency

To examine the impact of the quadratic cuts and dynamic ordering, we first run our algorithm with only quadratic cuts and then enable dynamic ordering on the FICO dataset from the Explainable Machine Learning Challenge (FICO et al., 2018). We also run this experiment using Exp-L0. L0Learn is used as a baseline. (MCP and LASSO use continuous regularization terms, which provides them with a run-time advantage, though these methods do not perform as well, as shown in the next subsections.) The ℓ_0 parameters we used are $\{0.8, 1, 2, 3, 4, 5, 6, 7\}$ and the ℓ_2 parameters used are $\{0.00001, 0.001\}$.

Figure 3 shows the training time and AUC values on the FICO dataset. The methods achieve performance comparable with Chen et al. (2021), who reported best black-box AUC ∼0.8. Our method using only linear cuts (purple bars) runs faster than the baseline (orange bars, L0Learn) for all regularization options. With ℓ_2 regularization coefficient $\lambda_2 = 0.001$, the time is reduced when we switch from using linear cuts to quadratic cuts (green bars) due to the tighter lower bound, as in Figure 1. The training time is further reduced by using both quadratic cuts and dynamic ordering (blue bars, which is LogRegQuad-L0). *Exp-L0 (red bars) is the fastest approach.* Again, this speed-up owes to the analytical line search and fast update.

From the four rightmost subfigures, we find that our improvement in training time does not negatively impact training/test AUC scores, as our methods (red and blue dots) form a "left frontier" with respect to the baseline L0Learn (orange dots). Results for additional datasets are in Appendix D.2.

7.2 Solution Quality

We next evaluate sparsity vs. performance. In addition to AUC on the datasets, we calculate Recovery-F1 score to measure how well we captured the ground truth support (ground truth coefficients $\boldsymbol{w}^*$ are known for simulated datasets). Recovery-F1 score is $\frac{2PR}{P+R}$, where $P = |\text{supp}(\hat{\boldsymbol{w}}) \cap \text{supp}(\boldsymbol{w}^*)|/|\text{supp}(\hat{\boldsymbol{w}})|$ is the precision and $R = |\text{supp}(\hat{\boldsymbol{w}}) \cap \text{supp}(\boldsymbol{w}^*)|/|\text{supp}(\boldsymbol{w}^*)|$ is the recall. supp(·) stands for the support (indices with nonzero coefficients) of a solution. We can use

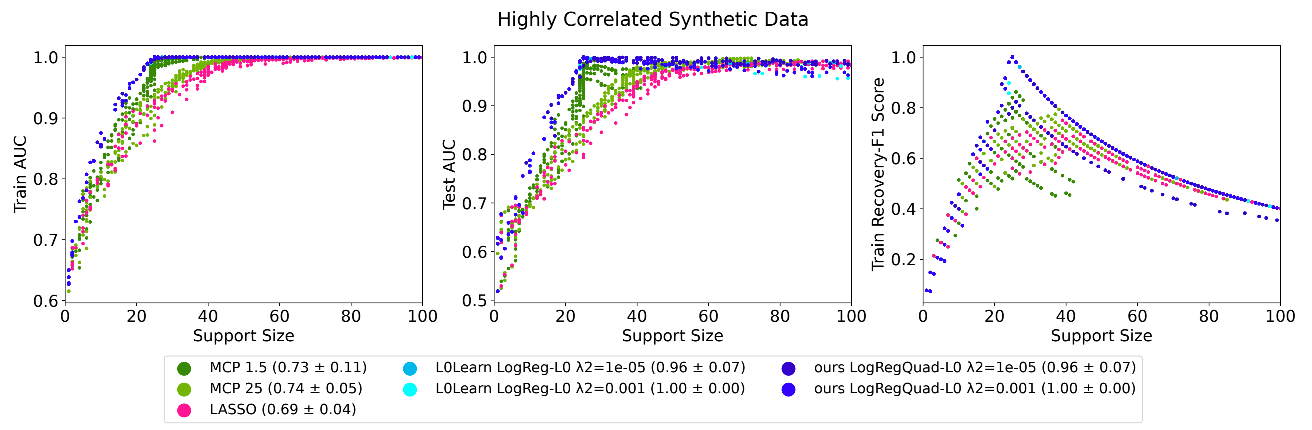

Figure 4: Results from all 5 datasets (each dataset generated by a different random seed) and parameter choices on highly correlated synthetic datasets. The parentheses contain the best Recovery-F1 scores averaged over all 5 datasets. MCP is shown with γ fixed at 1.5 and 25, and all other choices for γ lie between the shown regions. Our methods and L0Learn outperfom MCP and LASSO in terms of the AUC (left and middle), and better recover the true support (right). L0Learn's performance heavily overlaps with our methods. Our methods have a computational advantage over L0Learn as shown in the last section.

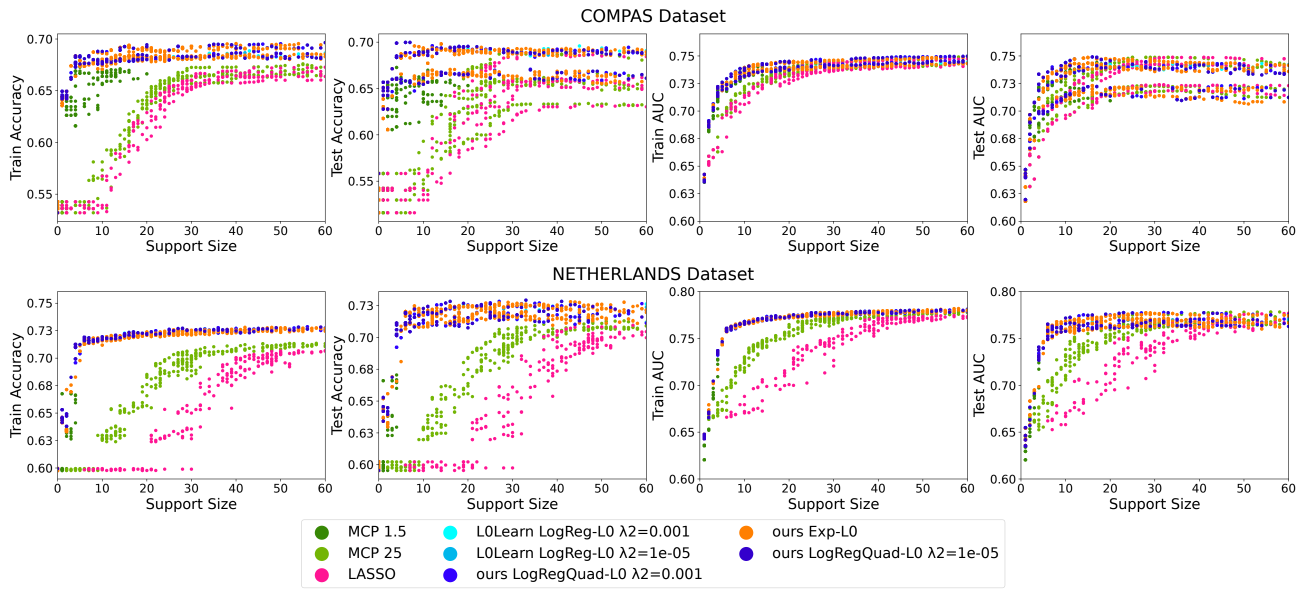

Figure 5: Results from all folds and parameter choices on real datasets: COMPAS and NETHERLANDS. We can see from the first and second columns (training and test accuracies) that MCP and LASSO do not perform well. Our methods and L0Learn (overlapping) outperform all other methods. Our methods are more computationally efficient than L0Learn.

Recovery-F1 score for synthetic data only, since we need to know $\boldsymbol{w}^*$ to calculate it.

Synthetic Data: Figure 4 shows sparsity/AUC tradeoffs and sparsity/Recovery-F1 tradeoffs on a synthetic dataset consisting of highly correlated features. Our methods are generally tied for the best results. LASSO (pink curves) and MCP (green curves) do not fully optimize the AUC, nor recover the correct support. For the full regularization path, the AUC's of L0Learn and our method largely overlap. However, as demonstrated in the previous subsection, our method runs much more quickly than L0Learn.

Since the features for this synthetic dataset are continuous (and we chose not to binarize them), Exp-L0 cannot be applied; its advantage comes from exploiting its analytical line search for binary features.

Real Datasets: Figure 5 shows sparsity-AUC tradeoffs and sparsity-accuracy tradeoffs on the COMPAS and NETHERLANDS datasets. LASSO and MCP do not achieve high prediction accuracy on training and test sets. L0Learn and our proposed methods have higher AUC and accuracy. Again, while L0Learn and our methods are tied for the best performance (which could be the optimal possible performance for this problem), our methods have major advantages in speed. More results are in Appendix D.3.

8 RELATED WORK

Mixed Integer Optimization. There have been many approaches to finding the optimal solution to logistic regression either with an ℓ_0 regularization or cardinality constraint (Sato et al., 2016, 2017; Ustun and Rudin, 2017; Bertsimas and King, 2017; Bertsimas et al., 2021; Sakaue and Marumo, 2019; Ustun and Rudin, 2019). In general, these approaches formulate the problem as a mixed-integer optimization problem (see Bertsekas, 1997; Wolsey and Nemhauser, 1999). The problem can then be solved using branch-and-bound search (see Land and Doig, 2010) or cutting-plane methods (Kelley, 1960; Gilmore and Gomory, 1961, 1963). However, even with the recent advances in hardware and software, MIP solvers are orders of magnitude slower than the methods we consider here and requires relatively large ℓ_2 regularization to work well (Bertsimas et al., 2021; Dedieu et al., 2021).

Gradient-based Heuristic Methods. One of the most widely used methods to promote sparsity is LASSO (Tibshirani, 1996), which relaxes the ℓ_0 penalty to ℓ_1. However, ℓ_1 simultaneously promotes sparsity and shrinks the coefficients, leading to bias. Several new methods obtain solutions under cardinality constraints or ℓ_0 penalty terms. One method is Orthogonal Matching Pursuit (OMP) (Lozano et al., 2011; Elenberg et al., 2018), which greedily selects the next-best feature based on the current support and gradients on coefficients. Other methods include Iterative Hard Thresholding (IHT) (Blumensath and Davies, 2009), coordinate descent (Beck and Eldar, 2013; Patrascu and Necoara, 2015; Dedieu et al., 2021), GraSP (Bahmani et al., 2013), and NHTP (Zhou et al., 2021). These methods enjoy fast computation, but their solutions suffer when the feature dimension is high or features are highly correlated because they can get stuck at local minima (Dedieu et al., 2021).

Local Feature Swaps. Some recent work considers swapping features on a given support. One such example is ABESS (Zhu et al., 2020; Zhang et al., 2021), which ranks features based on their contribution to the loss objective. Then, they swap only unimportant features in the support with features outside the support. Our experiments show that ABESS often returns "nan" values for its coefficients, thus in its current form was not able to be included in our experiments. Another work is L0Learn (Hazimeh and Mazumder, 2020; Dedieu et al., 2021), which exhaustively tries replacing every feature in the support with better features.

To the best of our knowledge, our work is the first where quadratic cuts (or exponential loss) and dynamic ordering have been used for sparse classification.

9 CONCLUSION

We have shown substantial speedups over other techniques for best subset search for probabilistic models with high-quality solutions. Our advances are due to several key ideas: (1) the use of cutting planes and quadratic cuts to form lower bounds, telling us when exploring a feature further is not worthwhile, (2) the use of the exponential loss, which has an analytical form, obviating the manipulations needed for logistic loss, (3) the use of a priority queue with a useful ordering function.

Code Availability

Implementations of the fast sparse classification method discussed in this paper are available at `https://github.com/jiachangliu/fastSparse`.

Acknowledgements

We acknowledge support from the U.S. National Institutes of Health under NIDA grant DA054994-01, and the National Science Foundation under grant DGE-2022040. We also acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

Sohail Bahmani, Bhiksha Raj, and Petros T Boufounos. Greedy sparsity-constrained optimization. *Journal of Machine Learning Research*, 14 (Mar):807–841, 2013.

Amir Beck and Yonina C Eldar. Sparsity constrained nonlinear optimization: Optimality conditions and algorithms. *SIAM Journal on Optimization*, 23(3): 1480–1509, 2013.

Dimitri P Bertsekas. Nonlinear programming. *Journal of the Operational Research Society*, 48(3):334–334, 1997.

Dimitris Bertsimas and Angela King. Logistic regression: From art to science. *Statistical Science*, pages 367–384, 2017.

Dimitris Bertsimas, Jean Pauphilet, and Bart Van Parys. Sparse classification: a scalable discrete optimization perspective. *Machine Learning*, 110(11):3177–3209, 2021.

Thomas Blumensath and Mike E Davies. Iterative hard thresholding for compressed sensing. *Applied and Computational Harmonic Analysis*, 27(3):265–274, 2009.

Patrick Breheny and Jian Huang. Coordinate descent algorithms for nonconvex penalized regression, with applications to biological feature selection. *Annals of Applied Statistics*, 5(1):232–253, 2011.

Chaofan Chen, Kangcheng Lin, Cynthia Rudin, Yaron Shaposhnik, Sijia Wang, and Tong Wang. A holistic approach to interpretability in financial lending: Models, visualizations, and summary-explanations. *Decision Support Systems*, page 113647, 2021.

Ingrid Daubechies, Michel Defrise, and Christine De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 57(11):1413–1457, 2004.

Antoine Dedieu, Hussein Hazimeh, and Rahul Mazumder. Learning sparse classifiers: Continuous and mixed integer optimization perspectives. *Journal of Machine Learning Research*, 22(135):1–47, 2021.

Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. Sequential monte carlo samplers. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(3):411–436, 2006.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

Ethan R Elenberg, Rajiv Khanna, Alexandros G Dimakis, and Sahand Negahban. Restricted strong convexity implies weak submodularity. *The Annals of Statistics*, 46(6B):3539–3568, 2018.

Şeyda Ertekin and Cynthia Rudin. On equivalence relationships between classification and ranking algorithms. *Journal of Machine Learning Research*, 12:2905–2929, 2011.

FICO, Google, Imperial College London, MIT, University of Oxford, UC Irvine, and UC Berkeley. Explainable Machine Learning Challenge. `https://community.fico.com/s/explainable-machine-learning-challenge`, 2018.

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.

Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859, 1961.

Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting stock problem—part ii. *Operations Research*, 11(6):863–888, 1963.

Trevor J Hastie and Robert J Tibshirani. *Generalized additive models*. Routledge, 2017.

Hussein Hazimeh and Rahul Mazumder. Fast best subset selection: Coordinate descent and local combinatorial optimization algorithms. *Operations Research*, 68(5):1517–1537, 2020.

James E Kelley, Jr. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 8(4):703–712, 1960.

Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

Ailsa H Land and Alison G Doig. An automatic method for solving discrete programming problems. In *50 Years of Integer Programming 1958-2008*, pages 105–132. Springer, 2010.

J. Larson, S. Mattu, L. Kirchner, and J. Angwin. How we analyzed the COMPAS recidivism algorithm. *ProPublica*, 2016.

Yin Lou, Jacob Bien, Rich Caruana, and Johannes Gehrke. Sparse partially linear additive models. *Journal of Computational and Graphical Statistics*, 25(4):1126–1140, 2016.

Aurelie Lozano, Grzegorz Swirszcz, and Naoki Abe. Group orthogonal matching pursuit for logistic regression. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 452–460, 2011.

Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

Harsha Nori, Samuel Jenkins, Paul Koch, and Rich Caruana. Interpretml: A unified framework for machine learning interpretability. *arXiv preprint arXiv:1909.09223*, 2019.

Andrei Patrascu and Ion Necoara. Random coordinate descent methods for ℓ_0 regularized convex optimization. *IEEE Transactions on Automatic Control*, 60

(7):1811–1824, 2015.

John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, April 21 1998.

Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. Interpretable machine learning: Fundamental principles and 10 grand challenges. *Statistics Surveys*, 16:1–85, 2022.

Shinsaku Sakaue and Naoki Marumo. Best-first search algorithm for non-convex sparse minimization. *arXiv preprint arXiv:1910.01296*, 2019.

Toshiki Sato, Yuichi Takano, Ryuhei Miyashiro, and Akiko Yoshise. Feature subset selection for logistic regression via mixed integer optimization. *Computational Optimization and Applications*, 64(3):865–880, 2016.

Toshiki Sato, Yuichi Takano, and Ryuhei Miyashiro. Piecewise-linear approximation for feature subset selection in a sequential logit model. *Journal of the Operations Research Society of Japan*, 60(1):1–14, 2017.

Robert E Schapire and Yoav Freund. Boosting: Foundations and algorithms. *Kybernetes*, 2013.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

Nikolaj Tollenaar and PGM Van der Heijden. Which method predicts recidivism best?: a comparison of statistical, machine learning and data mining predictive models. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 176(2):565–584, 2013.

Berk Ustun and Cynthia Rudin. Optimized risk scores. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1125–1134, 2017.

Berk Ustun and Cynthia Rudin. Learning optimized risk scores. *J. Mach. Learn. Res.*, 20:150–1, 2019.

Laurence A Wolsey and George L Nemhauser. *Integer and Combinatorial Optimization*, volume 55. John Wiley & Sons, 1999.

Yanhang Zhang, Junxian Zhu, Jin Zhu, and Xueqin Wang. Certifiably polynomial algorithm for best group subset selection. *arXiv preprint arXiv:2104.12576*, 2021. Code version: December 8, 2021.

Shenglong Zhou, Naihua Xiu, and Hou-Duo Qi. Global and quadratic convergence of newton hard-thresholding pursuit. *J. Mach. Learn. Res.*, 22(12): 1–45, 2021.

Junxian Zhu, Canhong Wen, Jin Zhu, Heping Zhang, and Xueqin Wang. A polynomial algorithm for best-subset selection problem. *Proceedings of the National Academy of Sciences*, 117(52):33117–33123, 2020.

Supplementary Material: Fast Sparse Classification for Generalized Linear and Additive Models

A THEOREMS AND PROOFS

A.1 Thresholding Is Too Conservative

The first theorem shows that thresholding is too conservative. Recall that with the support set fixed (i.e., $\lambda_0 = 0$), the loss can be written as $G(\boldsymbol{w}) = \sum_{i=1}^{n} \log(1 + \exp(-y_i(\boldsymbol{x}_i^T \boldsymbol{w}))) + \lambda_2 \|\boldsymbol{w}\|_2^2$.

Theorem 4.1 *(Thresholding is too conservative.) Let $\boldsymbol{w}^t$ be the current solution at iteration t, w_j^t be the coefficient for the j-th feature, and let w_j^* be the optimal value on the j-th coefficient while keeping all other coefficients fixed to their values at time t. Furthermore, let $\boldsymbol{w}^{t+1} = \boldsymbol{w}^t + \boldsymbol{e}_j(T(j, \boldsymbol{w}^t) - w_j^t)$, where $\boldsymbol{e}_j$ is a vector with 1 on the j-th component and 0 otherwise and $T(j, \boldsymbol{w}^t)$ is the thresholding operation with the support set fixed (i.e., $\lambda_0 = 0$). Then we have the following inequalities:*

$$\begin{aligned}\nabla_j G(\boldsymbol{w}^t) \nabla_j G(\boldsymbol{w}^{t+1}) &\geq 0,\\ (w_j^t - w_j^*)(w_j^{t+1} - w_j^*) &\geq 0,\\ \text{and } G(\boldsymbol{w}^t) &\geq G(\boldsymbol{w}^{t+1}).\end{aligned}$$

Proof.
For notational convenience, let us define two functions:

$$\begin{aligned}F(u) &:= G(\boldsymbol{w}^t) + (u - w_j^t)\nabla_j G(\boldsymbol{w}^t) + \frac{1}{2} L_j (u - w_j^t)^2\\ H(u) &:= G(\boldsymbol{w}^t - w_j^t \boldsymbol{e}_j + u \boldsymbol{e}_j).\end{aligned}$$

Using the notation above, our thresholding operation can be rewritten as $T(j, \boldsymbol{w}) \in \arg\min_u F(u)$. This means $w_j^{t+1} = T(j, \boldsymbol{w})$ minimizes $F(\cdot)$. After the thresholding operation, we update $\boldsymbol{w}$ by $\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - w_j^t \boldsymbol{e}_j + w_j^{t+1} \boldsymbol{e}_j$. Furthermore, we use w_j^* to denote the optimal value that minimizes $H(\cdot)$. Throughout this proof, we assume $\lambda_0 = 0$ because the support set is fixed.

Using the new notation for $F(u)$ and $H(u)$, we have the following expression for their first and second derivatives:

$$\begin{aligned}F'(u) &= \nabla_j G(\boldsymbol{w}^t) + L_j(u - w_j^t), & F''(u) &= L_j\\ H'(u) &= \nabla_j G(\boldsymbol{w}^t - w_j^t \boldsymbol{e}_j + u\boldsymbol{e}_j), & H''(u) &= \nabla_{jj}^2 G(\boldsymbol{w}^t - w_j^t \boldsymbol{e}_j + u\boldsymbol{e}_j)\\ F'(w_j^t) &= H'(w_j^t), & 0 \leq H''(u) &\leq L_j = F''(u).\end{aligned}$$

To get $F'(w_j^t) = H'(w_j^t)$, we plug in $u = w_j^t$ into the formula for $F'(u)$ and $H'(u)$. For the last inequalities, we have $H''(u) \geq 0$ because $H(u)$ is a convex function. In addition, we have $H''(u) \leq L_j$ because L_j is the Lipschitz constant for $H'(u)$ so that $|H'(u + d) - H'(u)| \leq L_j |d|$ and $|H''(u)| = \lim_{d \to 0} |\frac{H'(u+d) - H'(u)}{d}| \leq L_j$.

Note that $F(u)$ is a quadratic upper bound of $H(u)$. First we have that $F - H$ is a convex function because the second derivative of $F - H$ is greater than or equal to 0. Second, the first derivative of $F - H$ at w_j^t is 0. Third, $F - H$ at w_j^t is also 0. These three things mean that $F(u) - H(u) \geq 0$ for any $u \in \mathbb{R}$. Therefore, $F(u)$ is a quadratic upper bound of $H(u)$.

We want to show

$$\begin{aligned}\nabla_j G(\boldsymbol{w}^t) \nabla_j G(\boldsymbol{w}^{t+1}) &\geq 0,\\ (w_j^t - w_j^*)(w_j^{t+1} - w_j^*) &\geq 0,\\ \text{and } G(\boldsymbol{w}^t) &\geq G(\boldsymbol{w}^{t+1}).\end{aligned}$$

Using the new notation, it is equivalent for us to show

$$H'(w_j^t)H'(w_j^{t+1}) \geq 0, \tag{11}$$
$$(w_j^t - w_j^*)(w_j^{t+1} - w_j^*) \geq 0, \tag{12}$$
$$\text{and } H(w_j^t) \geq H(w_j^{t+1}). \tag{13}$$

To show the inequalities above, we discuss three cases: Case 1) $w_j^t < w_j^*$, Case 2) $w_j^t > w_j^*$, and Case 3) $w_j^t = w_j^*$.

Case 1: $w_j^t < w_j^*$

If $w_j^t < w_j^*$, we have $H'(w_j^t) < 0$. This is true because

$$\begin{aligned} H'(w_j^t) &= H'(w_j^*) + \int_{w_j^*}^{w_j^t} H''(u)du \\ &= 0 + \int_{w_j^*}^{w_j^t} H''(u)du \\ &= -\int_{w_j^t}^{w_j^*} H''(u)du < 0. \end{aligned}$$

The last inequality holds because $H''(u) \geq 0$ and $H''(u) > 0$ for some nonzero measurable set in $[w_j^t, w_j^*]$.

Now because w_j^{t+1} minimizes $F(\cdot)$, we have $F'(w_j^{t+1}) = 0$. Using the relationship between $F(\cdot)$ and $H(\cdot)$, we have

$$\begin{aligned} 0 = F'(w_j^{t+1}) &= F'(w_j^t) + \int_{w_j^t}^{w_j^{t+1}} F''(u)du \\ &= H'(w_j^t) + \int_{w_j^t}^{w_j^{t+1}} F''(u)du \\ &\geq H'(w_j^t) + \int_{w_j^t}^{w_j^{t+1}} H''(u)du \\ &= H'(w_j^{t+1}). \end{aligned}$$

Therefore, we have $H'(w_j^{t+1}) \leq 0$. Since $H'(w_j^t) < 0$, we have $H'(w_j^t)H'(w_j^{t+1}) \geq 0$, proving (11) for Case 1.

Let us prove (12) for Case 1. For the sake of contradiction, suppose $w_j^{t+1} > w_j^*$, we have $H'(w_j^{t+1}) > 0$ because

$$\begin{aligned} H'(w_j^{t+1}) &= H'(w_j^*) + \int_{w_j^*}^{w_j^{t+1}} H''(u)du \\ &= 0 + \int_{w_j^*}^{w_j^{t+1}} H''(u)du \\ &= \int_{w_j^*}^{w_j^{t+1}} H''(u)du > 0. \end{aligned}$$

This implies $H'(w_j^t)H'(w_j^{t+1}) < 0$, contradicting the proof of (11) for Case 1 above. Thus, we have $w_j^{t+1} \leq w_j^*$ and $(w_j^t - w_j^*)(w_j^{t+1} - w_j^*) \geq 0$, proving (12) for Case 1.

Lastly, because of the relationship between $F(\cdot)$ and $H(\cdot)$, we have

$$H(w_j^{t+1}) \leq F(w_j^{t+1}) = \min_u F(u) \leq F(w_j^t) = H(w_j^t).$$

This proves our third inequality (13) for Case 1.

Case 2: $w_j^t > w_j^*$

If $w_j^t > w_j^*$, we have $H'(w_j^t) > 0$. The procedure to show this is very similar to what we have shown in Case 1, so we omit it here.

Now because w_j^{t+1} minimizes $F(\cdot)$, we have $F'(w_j^{t+1}) = 0$. Using the relationship between $F(\cdot)$ and $H(\cdot)$, we have

$$\begin{aligned} 0 = F'(w_j^{t+1}) &= F'(w_j^t) + \int_{w_j^t}^{w_j^{t+1}} F''(u)du \\ &= H'(w_j^t) + \int_{w_j^t}^{w_j^{t+1}} F''(u)du \\ &\leq H'(w_j^t) + \int_{w_j^t}^{w_j^{t+1}} H''(u)du \\ &= H'(w_j^{t+1}). \end{aligned}$$

The third line holds true because $w_j^{t+1} < w_j^t$. Therefore, we have $H'(w_j^{t+1}) \geq 0$. Since $H'(w_j^t) > 0$, we have $H'(w_j^t)H'(w_j^{t+1}) \geq 0$, proving the inequality (11) under Case 2.

We proceed to prove (12) for Case 2. Now for the sake of contradiction, suppose $w_j^{t+1} < w_j^*$, we have $H'(w_j^{t+1}) < 0$. Again, the procedure to show this is very similar to what we have shown in Case 1, so we omit it here. This reasoning implies $H'(w_j^t)H'(w_j^{t+1}) < 0$, contradicting the proof of (11) under Case 2 above. Thus, we have $w_j^{t+1} \geq w_j^*$ and $(w_j^t - w_j^*)(w_j^{t+1} - w_j^*) \geq 0$, proving (12) for Case 2.

The procedure to prove (13) under Case 2 identical to (13) under Case 1, so we omit the proof here.

Case 3: $w_j^t = w_j^*$.

In this special case, $w_j^{t+1} = w_j^t$. Thus, the three inequalities (11), (12), and (13) hold trivially. This completes the proof for Theorem 4.1.

To the best of our knowledge, we are the first to show that w_j^{t+1} and w_j^t stay on the same side of w_j^*. This important point is the motivation behind our use of cutting planes and the development of our quadratic lower bound.

A.2 Lower Bound via Cutting Planes

Theorem 4.2 *Suppose $f(x)$ is convex and differentiable on domain $\mathbb{R}$. Let α_1 and α_2 be slopes of tangent lines of $f(x)$ at locations x_1 and x_2. If $\alpha_1\alpha_2 \leq 0$, there is a lower bound on the optimal value $f(x^*)$:*

$$f(x^*) \geq \frac{\alpha_1 f(x_2) - \alpha_2 f(x_1) + \alpha_1\alpha_2(x_1 - x_2)}{\alpha_1 - \alpha_2}.$$

Proof.
Because of the convexity of $f(x)$, we have

$$\begin{aligned} f(x) &\geq f(x_1) + \alpha_1(x - x_1) \quad \text{where } \alpha_1 = f'(x_1) \\ f(x) &\geq f(x_2) + \alpha_2(x - x_2) \quad \text{where } \alpha_2 = f'(x_2). \end{aligned}$$

Notice that the function $f(x)$ sits above two lines $y = f(x_1) + \alpha_1(x - x_1)$ and $y = f(x_2) + \alpha_2(x - x_2)$.

Equating these two lines to find the intersection point $\hat{x}$, we have

$$\begin{aligned} & f(x_1) + \alpha_1(\hat{x} - x_1) = f(x_2) + \alpha_2(\hat{x} - x_2) \qquad (14) \\ \Rightarrow & \hat{x} = \frac{f(x_2) - f(x_1) + \alpha_1 x_1 - \alpha_2 x_2}{\alpha_1 - \alpha_2}. \end{aligned}$$

To find the intersection value $\hat{y}$, we plug in $\hat{x}$ into the left side of (14) and get

$$\begin{aligned}\hat{y} &= f(x_1) + \alpha_1(\hat{x} - x_1) \\ &= \frac{\alpha_1 f(x_2) - \alpha_2 f(x_1) + \alpha_1\alpha_2(x_1 - x_2)}{\alpha_1 - \alpha_2}.\end{aligned}$$

Since the function $f(x)$ sits above the two lines and therefore above the intersection value $\hat{y}$, we have

$$f(x^*) \geq \frac{\alpha_1 f(x_2) - \alpha_2 f(x_1) + \alpha_1\alpha_2(x_1 - x_2)}{\alpha_1 - \alpha_2}.$$

This completes the proof for Theorem 4.2.

A.3 Lower Bound via Quadratic Cuts

Theorem 4.3 *Suppose $f(x) = g(x) + \lambda_2 x^2$, where $g(x)$ is a convex and differentiable function. Then $f(x)$ is strongly convex. Let α_1 be the slope of the tangent line to $f(x)$ at location x_1. Then, there is a lower bound on the optimal value $f(x^*)$:*

$$f(x^*) \geq \mathcal{L}_{low} := f(x_1) - \frac{\alpha_1^2}{4\lambda_2}. \tag{15}$$

Let α_2 be the slope of the tangent line to $f(x)$ at another location x_2. If $\alpha_1\alpha_2 \leq 0$, a lower bound on the optimal value $f(x^*)$ is as follows:

$$f(x^*) \geq \mathcal{L}_{low} := f(\hat{x}) + \alpha_1(\hat{x} - x_1) + \lambda_2(\hat{x} - x_1)^2, \tag{16}$$

where

$$\hat{x} = \frac{-f(x_1) + f(x_2) + \alpha_1 x_1 - \alpha_2 x_2 - \lambda_2(x_1^2 - x_2^2)}{\alpha_1 - \alpha_2 - 2\lambda_2(x_1 - x_2)}.$$

Proof.
Given a convex function $g(x)$, we first show that $f(x) = g(x) + \lambda_2 x^2$ is a strongly convex function before proving the two bounds in Theorem 4.3.

To show that $f(x)$ is a strongly convex function, it is sufficient to show

$$f(y) \geq f(x) + f'(x)(y - x) + \lambda_2(y - x)^2 \tag{17}$$

for any $x, y \in \mathbb{R}$.

Because $g(x)$ is convex, we have

$$g(y) \geq g(x) + g'(x)(y - x).$$

Adding $\lambda_2 y^2$ to both sides, we have

$$g(y) + \lambda_2 y^2 \geq g(x) + g'(x)(y - x) + \lambda_2 y^2.$$

The LHS is $f(y)$. The RHS can be rewritten as

$$\begin{aligned}&g(x) + g'(x)(y - x) + \lambda_2 y^2 \\ &\quad= \quad g(x) + \lambda_2 x^2 + (g'(x) + 2\lambda_2 x)(y - x) + \lambda_2 y^2 - \lambda_2 x^2 - 2\lambda_2 x(y - x) \\ &\quad= \quad f(x) + f'(x)(y - x) + \lambda_2(x - y)^2.\end{aligned}$$

Therefore, $f(x)$ is a strongly convex function.

Because $f(\cdot)$ is strongly convex, where (17) holds for any x and y, given a point x_1 with $\alpha_1 := f'(x_1)$, then our strongly convex function $f(\cdot)$ at any point x is bounded by

$$f(x) \geq f(x_1) + \alpha_1(x - x_1) + \lambda_2(x - x_1)^2.$$

The RHS is a quadratic function of x, with the minimum value achieved at $f(x_1) - \frac{\alpha_1^2}{4\lambda_2}$, so we have

$$f(x) \geq f(x_1) - \frac{\alpha_1^2}{4\lambda_2}.$$

Since the above inequality works for any $x \in \mathbb{R}$, it also works for the optimal value x^*:

$$f(x^*) \geq f(x_1) - \frac{\alpha_1^2}{4\lambda_2}.$$

Therefore, we have proved (15).

Suppose we are given another point x_2 with $\alpha_2 := f'(x_2)$, then $f(x)$ sits above two quadratic equations:

$$\begin{aligned} f(x) &\geq f(x_1) + \alpha_1(x - x_1) + \lambda_2(x - x_1)^2 \\ f(x) &\geq f(x_2) + \alpha_2(x - x_2) + \lambda_2(x - x_2)^2. \end{aligned}$$

Equating these two quadratic equations to find the intersection point $\hat{x}$, we have

$$\begin{aligned} &f(x_1) + \alpha_1(\hat{x} - x_1) + \lambda_2(\hat{x} - x_1)^2 = f(x_2) + \alpha_2(\hat{x} - x_2) + \lambda_2(\hat{x} - x_2)^2 \\ \Rightarrow &\hat{x} = \frac{-f(x_1) + f(x_2) + \alpha_1 x_1 - \alpha_2 x_2 + \lambda_2(x_2^2 - x_1^2)}{\alpha_1 - \alpha_2 - 2\lambda_2(x_1 - x_2)}. \end{aligned}$$

Plugging in the intersection point $\hat{x}$, we can get the intersection value $\hat{y}$, which is a lower bound of $f(x^*)$

$$f(x^*) \geq \hat{y} = f(x_1) + \alpha_1(\hat{x} - x_1) + \lambda_2(\hat{x} - x_1)^2.$$

This completes the proof for (16).

A.4 Derivation for the Exponential Loss

The exponential loss function is defined as $H(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} e^{-y_i f(\mathbf{x}_i)}$, where $f(\boldsymbol{x}_i) = \boldsymbol{w}^T \boldsymbol{x}_i$. Since $\boldsymbol{x}_i$ is a binary vector, s.t. $x_{ij} \in \{-1, 1\}$ and $y_i \in \{-1, 1\}$, let $\boldsymbol{z}_i = y_i \boldsymbol{x}_i$ and $\boldsymbol{z}_i \in \{-1, 1\}^p$. After t iterations, the exponential loss function can be written as:

$$H(\boldsymbol{w}^t) = \frac{1}{n}\sum_{i=1}^{n} e^{-y_i(\sum_{j=1}^{p} w_j^t x_{ij})} = \frac{1}{n}\sum_{i=1}^{n} e^{-(\boldsymbol{w}^t)^T \boldsymbol{z}_i}.$$

We will perform a linesearch, where we optimize coefficient j at iteration t. This linesearch optimization problem for coordinate j is $w_j^{t+1} \in \arg\min_w H([w_1^t, ..., w_{j-1}^t, w, w_{j+1}^t, ...]) + \lambda_0 \|[w_1^t, ..., w_{j-1}^t, w, w_{j+1}^t, ...]\|_0$.

Theorem 5.1 *Let $\boldsymbol{w}^t$ be the coefficient vector at iteration t, $H^t := H(\boldsymbol{w}^t)$ and λ_0 be the regularization constant for the ℓ_0 penalty. For the j-th coordinate, we update the coefficient according to:*

(1) Suppose $w_j^t = 0$. Let $d_- = \sum_{i: z_{ij}=-1} c_i / \sum_{i=1}^{n} c_i$, where $c_i = e^{-(\boldsymbol{w}^t)^T \boldsymbol{z}_i}$. Then, if d_- is within the interval:

$$\left[\frac{1}{2} - \frac{1}{2H^t}\sqrt{\lambda_0(2H^t - \lambda_0)}, \frac{1}{2} + \frac{1}{2H^t}\sqrt{\lambda_0(2H^t - \lambda_0)}\right],$$

then set w_j^{t+1} to 0. Otherwise set $w_j^{t+1} = \frac{1}{2}\ln\frac{1-d_-}{d_-}$.

(2) Suppose $w_j^t \neq 0$. Let $D_- = \sum_{i: z_{ij}=-1} c_i / \sum_{i=1}^{n} c_i$, where $c_i = e^{-(\boldsymbol{w}^t - w_j^t \boldsymbol{e}_j)^T z_i}$. Let $H_{\neg j}^t = H(\boldsymbol{w}^t - w_j^t \boldsymbol{e}_j)$. Then, if D_- is within the interval:

$$\left[\frac{1}{2} - \frac{1}{2H_{\neg j}^t}\sqrt{\lambda_0(2H_{\neg j}^t - \lambda_0)}, \frac{1}{2} + \frac{1}{2H_{\neg j}^t}\sqrt{\lambda_0(2H_{\neg j}^t - \lambda_0)}\right],$$

then set w_j^{t+1} to 0. Otherwise, set $w_j^{t+1} = \frac{1}{2}\ln\frac{1-D_-}{D_-}$.

While these expressions may first appear difficult to calculate, they are not. Like AdaBoost, we make multiplicative updates to the loss at each iteration. Thus, since H^t is easy to calculate, $H^t_{\neg j}$ is also easy to calculate (requiring only a multiplication), and the rest is simple mathematical operations.

Intuitively, using AdaBoost's terminology, the bound states that if the weak learning algorithm produces a stronger weak classifier at that iteration (a classifier whose error rate is away from 1/2), we would keep it. Otherwise, we would not; we would rather set its coefficient to 0. In some sense, this result is reminiscent of iterative thresholding (Daubechies et al., 2004).

Proof.
Case 1: Suppose at iteration t, $w_j^t = 0$ and in the next iteration $t+1$, we evaluate placing feature j into the model, i.e., set $w_j^{t+1} \neq 0$. Then, the decrease in loss should be larger than λ_0, otherwise $w_j^{t+1} = 0$. Suppose we want to add feature j into the model, the loss function is

$$H^{t+1} = \frac{1}{n}\sum_{i=1}^{n} e^{-(\boldsymbol{w}^t)^T \boldsymbol{z}_i - y_i w_j x_{ij}} = \frac{1}{n}\sum_{i=1}^{n} e^{-(\boldsymbol{w}^t)^T \boldsymbol{z}_i - w_j z_{ij}}.$$

We can get an analytical solution for w_j by solving $\frac{\partial H^{t+1}}{\partial w_j} = 0$, which is the same as AdaBoost's update step.

$$\begin{aligned} 0 = \frac{\partial H^{t+1}}{\partial w_j}\Big|_{w_j^*} &= \sum_{i=1}^{n} -z_{ij} e^{-(\boldsymbol{w}^t)^T \boldsymbol{z}_i} e^{-w_j z_{ij}}\Big|_{w_j^*} \\ &= \sum_{i:z_{ij}=1} -e^{-(\boldsymbol{w}^t)^T \boldsymbol{z}_i} e^{-w_j}\Big|_{w_j^*} + \sum_{i:z_{ij}=-1} e^{-(\boldsymbol{w}^t)^T \boldsymbol{z}_i} e^{w_j}\Big|_{w_j^*}. \end{aligned} \tag{18}$$

Multiplying by a normalization constant

$$C = \sum_{i=1}^{n} c_i = \sum_{i=1}^{n} e^{-(\boldsymbol{w}^t)^T \boldsymbol{z}_i},$$

and defining

$$d_+ = \frac{\sum_{i:z_{ij}=1} e^{-(\boldsymbol{w}^t)^T \boldsymbol{z}_i}}{C} \quad \text{and} \quad d_- = \frac{\sum_{i:z_{ij}=-1} e^{-(\boldsymbol{w}^t)^T \boldsymbol{z}_i}}{C},$$

Equation (18) becomes

$$0 = -d_+ e^{-w_j^*} + d_- e^{w_j^*}.$$

Solving this yields:

$$w_j^* = \frac{1}{2}\ln\frac{d_+}{d_-}.$$

Recalling that $d_+ = 1 - d_-$, the lowest possible loss after adding in feature j is thus:

$$H^{t+1} = \mathcal{L}^t \cdot \left((1-d_-)\left(\frac{1-d_-}{d_-}\right)^{-1/2} + d_-\left(\frac{1-d_-}{d_-}\right)^{1/2}\right) = H^t \cdot 2\left((1-d_-)d_-\right)^{1/2}. \tag{19}$$

We have now derived the best possible value for w_j if it were nonzero. However, our objective suffers a penalty of λ_0 from the regularization term whenever w_j is nonzero. Thus, we need to compare the objective with $w_j = 0$ to the regularized objective with (19) as the loss term. If the difference is less than λ_0, it would benefit the objective to set coefficient j to 0 at the next iteration. The condition for setting w_j to 0 is:

$$\begin{aligned} H^t - H^{t+1} = H^t - H^t \cdot 2\left((1-d_-)d_-\right)^{1/2} &\leq \lambda_0. \\ \frac{H^t - \lambda_0}{2H^t} &\leq ((1-d_-)d_-)^{1/2} \\ \left(\frac{H^t-\lambda_0}{2H^t}\right)^2 &\leq (1-d_-)d_- \\ d_-^2 - d_- + \left(\frac{H^t-\lambda_0}{2H^t}\right)^2 &\leq 0. \end{aligned} \tag{20}$$

This is a quadratic equation, permitting solutions in $d_- \in \left(\frac{1}{2} - \frac{1}{2H^t}\sqrt{\lambda_0(2H^t - \lambda_0)}, \frac{1}{2} + \frac{1}{2H^t}\sqrt{\lambda_0(2H^t - \lambda_0)}\right)$.

Therefore, if $d_- \in \left(\frac{1}{2} - \frac{1}{2H^t}\sqrt{\lambda_0(2H^t - \lambda_0)}, \frac{1}{2} + \frac{1}{2H^t}\sqrt{\lambda_0(2H^t - \lambda_0)}\right)$, then set w_j^{t+1} to 0. Otherwise, $w_j^{t+1} = \frac{1}{2}\ln\frac{1-d_-}{d_-}$.

Case 2: Suppose at iteration t, $w_j^t \neq 0$, and in the next iteration $t+1$, we evaluate updating w_j^t. Then the decrease in loss should be larger than $H^t - H_{\neg j}^t + \lambda_0$, otherwise, $w_j^{t+1=0}$. Suppose we want to update w_j at iteration $t+1$, the loss function is

$$H^{t+1} = \frac{1}{n}\sum_{i=1}^{n} e^{-(\boldsymbol{w}^t - w_j^t \boldsymbol{e}_j)^T \boldsymbol{z}_i - w_j z_{ij}} \quad .$$

Similar to the derivation for Case 1, we can get an analytical solution for w_j by solving $\frac{\partial H^{t+1}}{\partial w_j} = 0$.

$$\begin{aligned} 0 = \frac{\partial H^{t+1}}{\partial w_j}\Big|_{w_j^*} &= \sum_{i=1}^{n} -z_{ij} e^{-(\boldsymbol{w}^t - w_j^t \boldsymbol{e}_j)^T \boldsymbol{z}_i} e^{(-w_j z_{ij})}\Big|_{w_j^*} \\ &= \sum_{i:z_{ij}=1} -e^{-(\boldsymbol{w}^t - w_j^t \boldsymbol{e}_j)^T \boldsymbol{z}_i} e^{-w_j}\Big|_{w_j^*} + \\ &\quad \sum_{i:z_{ij}=-1} e^{-(\boldsymbol{w}^t - w_j^t \boldsymbol{e}_j)^T \boldsymbol{z}_i} e^{w_j}\Big|_{w_j^*} \quad . \end{aligned} \tag{21}$$

Similarly, multiplying by a normalization constant C, and defining $D_+ = \sum_{i:z_{ij}=1} e^{-(\boldsymbol{w}^t - w_j^t \boldsymbol{e}_j)^T \boldsymbol{z}_i}/C$ and $D_- = \sum_{i:z_{ij}=-1} e^{-(\boldsymbol{w}^t - w_j^t \boldsymbol{e}_j)^T \boldsymbol{z}_i}/C$.

Then Equation 21 becomes

$$0 = -D_+ e^{-w_j^*} + D_- e^{w_j^*}.$$

Solving this yields:

$$w_j^* = \frac{1}{2}\ln\frac{D_+}{D_-}.$$

The lowest possible loss after updating the coefficient of feature j is

$$H^{t+1} = H_{\neg j}^t \cdot \left(D_+\left(\frac{D_+}{D_-}\right)^{-1/2} + D_-\left(\frac{D_+}{D_-}\right)^{1/2}\right) = H_{\neg j}^t \cdot 2((1-D_-)D_-)^{1/2}. \tag{22}$$

Similarly to Case 1, we need to compare the objective with $w_j = 0$ to the regularized objective with (22) as the loss term. If the difference is less than λ_0, it would benefit the objective to set coefficient j to 0 at the next iteration. The condition for setting w_j to 0 is:

$$H_{\neg j}^t - H^{t+1} = H_{\neg j}^t - H_{\neg j}^t \cdot 2\left((1-D_-)D_-\right)^{1/2} \leq \lambda_0.$$

Using the same derivation as in Equation (20), the solution is in

$$D_- \in \left(\frac{1}{2} - \frac{1}{2H_{\neg j}^t}\sqrt{\lambda_0(2H_{\neg j}^t - \lambda_0)}, \frac{1}{2} + \frac{1}{2H_{\neg j}^t}\sqrt{\lambda_0(2H_{\neg j}^t - \lambda_0)}\right).$$

Therefore, if $D_- \in \left(\frac{1}{2} - \frac{1}{2H_{\neg j}^t}\sqrt{\lambda_0(2H_{\neg j}^t - \lambda_0)}, \frac{1}{2} + \frac{1}{2H_{\neg j}^t}\sqrt{\lambda_0(2H_{\neg j}^t - \lambda_0)}\right)$, then set w_j^{t+1} to 0. Otherwise, $w_j^{t+1} = \frac{1}{2}\ln\frac{1-D_-}{D_-}$.

B PSEUDOCODE

We begin with the presentation of our high-level Algorithm 1 and then elaborate on the novel steps in the following lower-level algorithms.

Shortly, we discuss how TryDeleteOrSwap($\boldsymbol{w}, j, S^c$) is implemented in detail. After that, we discuss its subroutine algorithms TryAddLinCut($\boldsymbol{w}', j', \mathcal{L}_{best}$) and TryAddQuad($\boldsymbol{w}', j', \mathcal{L}_{best}$), as well as their subroutine algorithm FindNewCoefficient($\boldsymbol{w}', j'$). For algorithms TryAddLinCut($\boldsymbol{w}', j', \mathcal{L}_{best}$) and TryAddQuad($\boldsymbol{w}', j', \mathcal{L}_{best}$), we use $f(x) = G(\boldsymbol{w}' + \boldsymbol{e}_{j'}x)$ for notational convenience (assuming $w'_{j'} = 0$; if it is not, notation can be adjusted appropriately). Also for notational convenience, we use one lower bound from classical cutting planes and two lower bounds from quadratic cuts:

$$\begin{aligned}
\text{LinCut}(a, b, f(\cdot)) &= \frac{f'(a)f(b) - f'(b)f(a) + f'(a)f'(b)(a-b)}{f'(a) - f'(b)} \\
\text{QuadCut1}(a, f(\cdot)) &= f(a) - \frac{f'(a)^2}{4\lambda_2} \\
\text{QuadCut2}(a, b, f(\cdot)) &= f(a) + f'(a)(\hat{x} - a) + \lambda_2(\hat{x} - a)^2 \\
\text{with } \hat{x} &= \frac{-f(a) + f(b) + f'(a)a - f'(b)b + \lambda_2(b^2 - a^2)}{f'(a) - f'(b) - 2\lambda_2(a-b)}.
\end{aligned}$$

Algorithm 1 General Algorithm for Swapping Features

Input: coefficients $\boldsymbol{w}$ from a warm start algorithm, $\boldsymbol{c} = \boldsymbol{0}$ is a vector of size p where each c_j for $j < p$ indicates the number of times we failed to find a feature to swap with j.

Output: updated coefficients $\boldsymbol{w}$ that is a swap 1-OPT solution.

```
1:  while True do
2:      Update support $S = \{j | w_j \neq 0\}$.
3:      $\Pi(S)$ = Sort($S$) according to $c_j$ for $j \in S$.    #Sort support in ascending order of the no. of failed swaps
4:      for $j$ in $\Pi(S)$ do
5:          $\boldsymbol{w}'$ = TryDeleteOrSwap($\boldsymbol{w}, j, S^c$)    #$S^c$ is the complement of $S$
6:          if $\boldsymbol{w}' \neq \boldsymbol{w}$ then
7:              Let $\boldsymbol{w} = \boldsymbol{w}'$.    #Swap was successful
8:              Go to line 2.
9:          else
10:             $c_j = c_j + 1$.    #No better feature can replace feature j
11:         end if
12:     end for
13:     Return $\boldsymbol{w}$.    #No single feature can be replaced with better features
14: end while
```

Algorithm 2 TryDeleteOrSwap($\boldsymbol{w}, j, S^c$)

Input: coefficients $\boldsymbol{w}$, feature index j with $w_j \neq 0$, set of feature indices $S^c = \{j' | w_{j'} = 0\}$.
Output: updated coefficients $\boldsymbol{w}'$ with feature j possibly deleted or swapped with feature $j' \in S^c$.

```
1: Calculate the best current loss $\mathcal{L}_{best} = G(\boldsymbol{w})$.
2: Let $\boldsymbol{w}' = \boldsymbol{w}$ and then set $w'_j = 0$.    #Drop feature $j$ from the support
3: if $G(\boldsymbol{w}') \leq \mathcal{L}_{best}$ then
4:     Update $\boldsymbol{w}'$ with support restricted to $S \setminus \{j\}$.    #Dropping feature $j$ leads to smaller loss
5:     Return $\boldsymbol{w}'$.
6: end if
7: Calculate $|\nabla_{S^c} G(\boldsymbol{w}^t)|$ on $S^c$.
8: $\Pi'$ = feature indices in $S^c$ sorted in descending order of $|\nabla_{S^c} G(\boldsymbol{w}')|$.    #Order features to possibly add in
9: for $j' \in \Pi'$ do
10:     Let $\boldsymbol{w}' = \text{TryAddQuad}(\boldsymbol{w}', j', \mathcal{L}_{best})$ if $(\lambda_2 > 0)$.    #or $\boldsymbol{w}' = \text{TryAddLinCut}(\boldsymbol{w}', j', \mathcal{L}_{best})$ if $(\lambda_2 = 0)$
11:     if $\boldsymbol{w}'$ has changed then
12:         Update full vector $\boldsymbol{w}'$ with support restricted to $S \cup \{j'\} \setminus \{j\}$.    #Swapping $j$ with $j'$ decreases loss
13:         Return $\boldsymbol{w}'$.
14:     end if
15: end for
16: Return $\boldsymbol{w}$.    #Since there were no better features, return original $\boldsymbol{w}$
```

Algorithm 3 TryAddLinCut($\boldsymbol{w}', j', \mathcal{L}_{best}$)

Input: coefficients $\boldsymbol{w}'$, feature index j', and current best loss $\mathcal{L}_{best}$.
Output: updated coefficients $\boldsymbol{w}'$.

```
1: Let $a = T(j', \boldsymbol{w}')$, $b = 2T(j', \boldsymbol{w}')$    #Take 2X distance suggested by thresholding operation Eq (4)
2: if $f'(0)f'(b) < 0$ then
3:     Let $c = (a+b)/2$    #Binary search
4:     if $f'(0)f'(c) < 0$ then
5:         Let $b = c$
6:     else
7:         Let $a = c$
8:     end if    #$a$ and $b$ are on opposite sides of $w^*_{j'}$
9:     Get $\mathcal{L}_{low} = \text{LinCut}(a, b, f(\cdot))$
10:     if $\mathcal{L}_{low} \geq \mathcal{L}_{best}$ then
11:         Return $\boldsymbol{w}'$    #Stop considering feature $j'$ and exit early
12:     end if
13:     Let $\hat{w}_{j'} = \text{FindNewCoefficient}(\boldsymbol{w}', j')$.
14:     if $f(\hat{w}_{j'}) < \mathcal{L}_{best}$ then
15:         Return $\boldsymbol{w}'$ with $w'_{j'} = \hat{w}_{j'}$    #Swap feature $j$ with feature $j'$
16:     end if
17:     Return $\boldsymbol{w}'$    #Eliminate considering feature $j'$
18: end if
19: Let $a = 2T(j', \boldsymbol{w}')$, $b = 3T(j', \boldsymbol{w}')$    #Take 3X distance suggested by thresholding operation
20: if $f'(0)f'(b) < 0$ then
21:     Go to line 9.
22: else
23:     Go to line 13.    #Minimum is far from starting point. Swap the feature to see if there's improvement.
24: end if
```

Algorithm 4 TryAddQuad($\boldsymbol{w}', j', \mathcal{L}_{best}$)

Input: coefficients $\boldsymbol{w}'$, feature index j', and current best loss $\mathcal{L}_{best}$
Output: updated coefficients $\boldsymbol{w}'$

```
 1: Get $\mathcal{L}_{low}$ = QuadCut1($0, f(\cdot)$)
 2: if $\mathcal{L}_{low} \geq \mathcal{L}_{best}$ then
 3:     Return $\boldsymbol{w}'$                          #Stop considering feature $j'$ and exit early
 4: end if
 5: Let $a = T(j', \boldsymbol{w}')$, $b = 2T(j', \boldsymbol{w}')$   #Take 2X distance suggested by thresholding operation Eq (4)
 6: if $f'(0)f'(b) < 0$ then
 7:     Let $c = (a + b)/2$                               #Binary search
 8:     Get $\mathcal{L}_{low}$ = QuadCut1($c, f(\cdot)$)
 9:     if $\mathcal{L}_{low} \geq \mathcal{L}_{best}$ then
10:         Return $\boldsymbol{w}'$                      #Stop considering feature $j'$ and exit early
11:     end if
12:     if $f'(0)f'(c) < 0$ then
13:         Let $b = c$
14:     else
15:         Let $a = c$
16:     end if                                            #a and b are on opposite sides of $w^*_{j'}$
17:     Get $\mathcal{L}_{low}$ = QuadCut2($a, b, f(\cdot)$)
18:     if $\mathcal{L}_{low} \geq \mathcal{L}_{best}$ then
19:         Return $\boldsymbol{w}'$                      #Stop considering feature $j'$ and exit early
20:     end if
21:     Let $\hat{w}_{j'}$ = FindNewCoefficient($\boldsymbol{w}', j'$).
22:     if $f(\hat{w}_{j'}) < \mathcal{L}_{best}$ then
23:         Return $\boldsymbol{w}'$ with $w'_{j'} = \hat{w}_{j'}$    #Swap feature $j$ with feature $j'$
24:     end if
25:     Return $\boldsymbol{w}'$                          #Eliminate considering feature $j'$
26: end if
27: Let $a = 2T(j', \boldsymbol{w}')$, $b = 3T(j', \boldsymbol{w}')$   #Take 3X distance suggested by thresholding operation
28: Get $\mathcal{L}_{low}$ = QuadCut1($a, f(\cdot)$)
29: if $\mathcal{L}_{low} \geq \mathcal{L}_{best}$ then
30:     Return $\boldsymbol{w}'$                          #Stop considering feature $j'$ and exit early
31: end if
32: if $f'(0)f'(b) < 0$ then
33:     Go to line 17.
34: else
35:     Get $\mathcal{L}_{low}$ = QuadCut1($b, f(\cdot)$)
36:     if $\mathcal{L}_{low} \geq \mathcal{L}_{best}$ then
37:         Return $\boldsymbol{w}'$                      #Stop considering feature $j'$ and exit early
38:     end if
39:     Go to line 21.    #Minimum is far from starting point. Swap the feature to see if there's improvement.
40: end if
```

Algorithm 5 FindNewCoefficient($\boldsymbol{w}', j'$)

Input: coefficients $\boldsymbol{w}$, coordinate j', iteration steps max_iter=10 (default)
Output: updated coefficient $w_{j'}$ for coordinate j'

```
1: for t in 1, 2, ..., max_iter do
2:     $\boldsymbol{w}' = \boldsymbol{w}' - w_{j'}\boldsymbol{e}_{j'} + T(\boldsymbol{w}', j')\boldsymbol{e}_{j'}$   #Apply the thresholding operation on coordinate $j'$
3: end for
4: Return $w'_{j'}$
```

C EXPERIMENTAL DETAILS

We next present the datasets used in our experiments, our preprocessing steps, and the experimental setup.

C.1 Datasets

We present results using 5 datasets: two synthetic datasets (one in which the features are highly correlated for binary classification and the other in which the features are highly correlated for linear regression), the Fair Isaac (FICO) credit risk dataset (FICO et al., 2018) used for the Explainable ML Challenge, two recidivism datasets: COMPAS (Larson et al., 2016) and Netherlands (Tollenaar and Van der Heijden, 2013). We predict whether an individual will default on a loan for the FICO dataset, which individuals are arrested within two years of release on the COMPAS dataset, and whether defendants have any type of charge within four years on the Netherlands dataset.

Dataset Name	n	p
Highly Correlated (classification)	800	1000
Highly Correlated (regression)	2000	2000
FICO	10459	1917
COMPAS	6907	134
NETHERLANDS	20000	2024

Table 1: Datasets and their number of samples (n) and number of features (p).

C.2 Data Generation and Preprocessing

Synthetic Datasets

Binary Classification: we generate synthetic datasets according to the generation process in L0Learn (Dedieu et al., 2021). We first sample the data features $\boldsymbol{x}_i \in \mathbb{R}^p$ from a multivariate Gaussian distribution $\mathcal{N}(0, \Sigma)$ with mean 0 and covariance matrix Σ. Then, we create the coefficient vector $\boldsymbol{w}$ with k nonzero entries, where $w_i = 1$ if $i \bmod (p/k) = 0$. Lastly, we sample the data labels $y_i \in \{-1, +1\}$ from a Bernoulli distribution $P(y_i = 1 \mid \boldsymbol{x}_i) = \frac{1}{1+\exp(-\boldsymbol{w}^T\boldsymbol{x}_i)}$. In our experiments, we generate 800 training and 160 test samples with feature dimension $p = 1000$. The data are highly correlated with $\Sigma_{ij} = 0.9^{|i-j|}$. Additionally, we set the number of true sparsity $k = 25$. We generate this setting 5 times with 5 different random seeds (in total we have 5 datasets, each with $(800 + 160) = 960$ samples).

Linear Regression: we generate the synthetic dataset according to the generation process in L0Learn (Hazimeh and Mazumder, 2020) as explained in Section 5.3.1. We first sample the data features $\boldsymbol{x}_i \in \mathbb{R}^p$ from a multivariate Gaussian distribution $\mathcal{N}(0, \Sigma)$ with mean 0 and covariance matrix Σ. Then, we create the coefficient vector $\boldsymbol{w}$ with k nonzero entries, where $w_i = 1$ if $i \bmod (p/k) = 0$. Lastly, we sample $y_i = \boldsymbol{x}_i^T\boldsymbol{w} + \epsilon_i$ with ϵ_i generated from a Gaussian distribution $\mathcal{N}(0, \sigma^2)$. The signal-to-noise ratio (SNR) is defined as $\text{SNR} = \frac{\text{Var}(X\boldsymbol{w})}{\text{Var}(\epsilon)} = \frac{\boldsymbol{w}^T\Sigma\boldsymbol{w}}{\sigma^2}$, where each row of X is $\boldsymbol{x}_i$. In our experiments, we generate 2000 data samples with feature dimension $p = 2000$. The data are highly correlated with $\Sigma_{ij} = 0.9^{|i-j|}$ and $\text{SNR} = 5$. Additionally, we set the true sparsity as $k = 100$.

Real Datasets

FICO: We use all continuous features in this dataset. We did not consider missing data values as separate dummy variables.

COMPAS: We selected features *sex, age, juv_fel_count, juv_misd_count, juv_other_count, priors_count*, and *c_charge_degree* and the label *two_year_recid*.

NETHERLANDS: We translated the feature names from Dutch to English and then used features *sex, country of birth, log # of previous penal cases, 11-20 previous case, and >20 previous case, age in years, age at first penal case, offence type*, and the label *recidivism_in_4y*.

For FICO and COMPAS, we convert each continuous variable $x_{\cdot,j}$ into a set of highly correlated dummy variables $\tilde{x}_{\cdot,j,\theta} = \mathbf{1}_{[x_{\cdot,j} \leq \theta]}$, where θ are all unique values that have appeared in feature column j. For NETHERLANDS, we convert continuous variables into a set of dummy variables in the same way except for variables *age in years* (which is real-valued, not integer) and *age at first penal case*. For these two real-valued variables, instead of considering all unique values that have appeared in the feature column, we consider 1000 quantiles.

C.3 Evaluation Platform

All experimental results were run on a 2.40GHz 30M Cache (256GB RAM 48 hyperthreaded cores) Dell R620 with 2 Xeon(R) CPU E5-2695 v2. We ran all experiments using 8 cores per task.

C.4 Software Packages Used

We list all software packages used in this section. Details about hyperparameter selection are in Appendix D.

- ℓ_1 regularized logistic regression: We run ℓ_1 regularized logistic regression using *glmnet* package (Friedman et al., 2010).
- Minimax Concave Penalty (MCP): We run MCP using *ncvreg* package (Breheny and Huang, 2011).
- L0Learn: We run L0Learn using the R implementation from (Dedieu et al., 2021)[1].
- Ours: We build our method based on L0Learn's codebase, so that we could use its preprocessing steps, and pipeline for running the full regularization path of λ_0 values.

There are some other baselines such as GraSP (Bahmani et al., 2013) and NHTP (Zhou et al., 2021). However, previous work (Dedieu et al., 2021) has shown that they have a considerable number of false positives on the synthetic dataset and have large support sizes for their solutions, so we omit running these two baselines.

C.5 Evaluation Metrics

We use the same evaluation metrics used in Dedieu et al. (2021).

- AUC: The area under the ROC curve.
- Accuracy: $1 - \frac{\sum_{i=1}^{n} \mathbb{1}[y_i \neq \hat{y}_i]}{n}$.
- Recovery F1 score: $\frac{2PR}{P+R}$, where $P = |\text{supp}(\hat{\boldsymbol{w}}) \cap \text{supp}(\boldsymbol{w}^*)| / |\text{supp}(\hat{\boldsymbol{w}})|$ is the precision and $R = |\text{supp}(\hat{\boldsymbol{w}}) \cap \text{supp}(\boldsymbol{w}^*)| / |\text{supp}(\boldsymbol{w}^*)|$ is the recall. $\text{supp}(\cdot)$ stands for the support (indices with nonzero coefficients) of a solution. We can only use recovery F1 score for synthetic datasets since we need to know the support of $\boldsymbol{w}^*$ to calculate it.

D ADDITIONAL EXPERIMENTS

We first elaborate on hyperparameters used for different software packages. We then present extra experimental results that were omitted from the main paper due to space constraints.

Collection and Setup: we ran the experiments on the two simulated datasets and 3 real datasets: FICO, COMPAS, and Netherlands. For each dataset, we trained the model using varying configurations. On the simulated classification task, we ran on 5 datasets, each generated by a different random seed. On the real datasets, we performed 5-fold cross validation to measure training time, training accuracy, and test accuracy for each fold.

To get the support versus AUC, accuracy, and F1 score curves, for MCP, the sequence of 100 λ values was set to the default values of ncvreg, and we chose the second parameter γ by using 10 values between 1.5 and 25, where we show results of γ being 1.5 and 25 for each λ. Curves for other γ values are between these extremes. For ℓ_1 regularized logistic regression, the choice of 100 λ values was set to the default sequence chosen by glmnet. For L0Learn, we set the penalty type to "L0L2" and γ to $\{0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10\}$. The regularization

[1]https://github.com/hazimehh/L0Learn

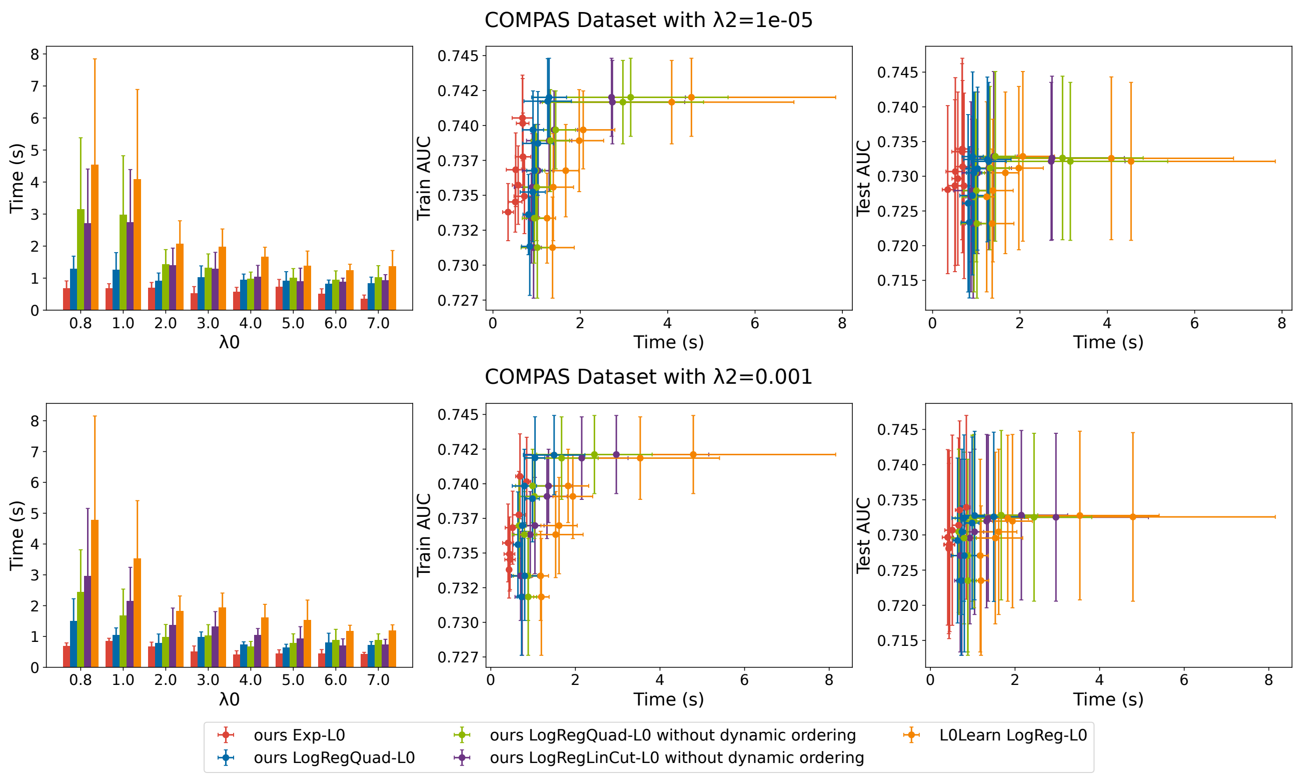

Figure 6: Computational times of different methods. “Exp” stands for exponential loss, “LogReg” stands for logistic loss, and “Quad” stands for quadratic cuts. Note that there is no ℓ_2 penalty for the exponential loss. *Our Exp-L0 method is generally about 4 times faster than L0Learn.* Note that the AUC axes indicate practically similar performance for these particular methods; the training time is what differentiates the methods.

choices for the ℓ_0 term were set to the 100 default values of L0Learn. For our methods, we also set the penalty type and γ (which is λ_2) in the same way as the setting for L0Learn and use the same λ values as in the L0Learn algorithm.

In addition, we set 56 pairs of λ (resp. λ_0) and γ (resp. λ_2) values for comparing the run times obtained by our methods and by L0Learn: $\lambda \in \{0.8, 1, 2, 3, 4, 5, 6, 7\}$ and $\gamma \in \{0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10\}$.

D.1 Run Time Savings for Linear Regression

Although our method is designed for classification problems, the proposed dynamic ordering technique can also speed up the local swap process for linear regression. For the full regularization path with 100 different λ_0 values, the total time difference between local swaps without dynamic ordering and local swaps with dynamic ordering improved computation time by 36% (from 184 seconds to 117 seconds).

D.2 Run Time Savings from First and Second Methods

We show results on time savings from our first method (linear cut, quadratic cut, and dynamic ordering) and our second method (exponential loss). The general trends are: *i*) Using the quadratic cut makes the algorithm faster than the linear cut, i.e., there is more time saved with stronger ℓ_2 regularization. *ii*) Using dynamic ordering and the quadratic cut together makes the algorithm much faster than using the quadratic cut alone. *iii*) When features are binary, using the exponential loss has the greatest computational advantage. These trends are shown fairly uniformly across datasets. Results for each dataset are shown in Figures 6-7.

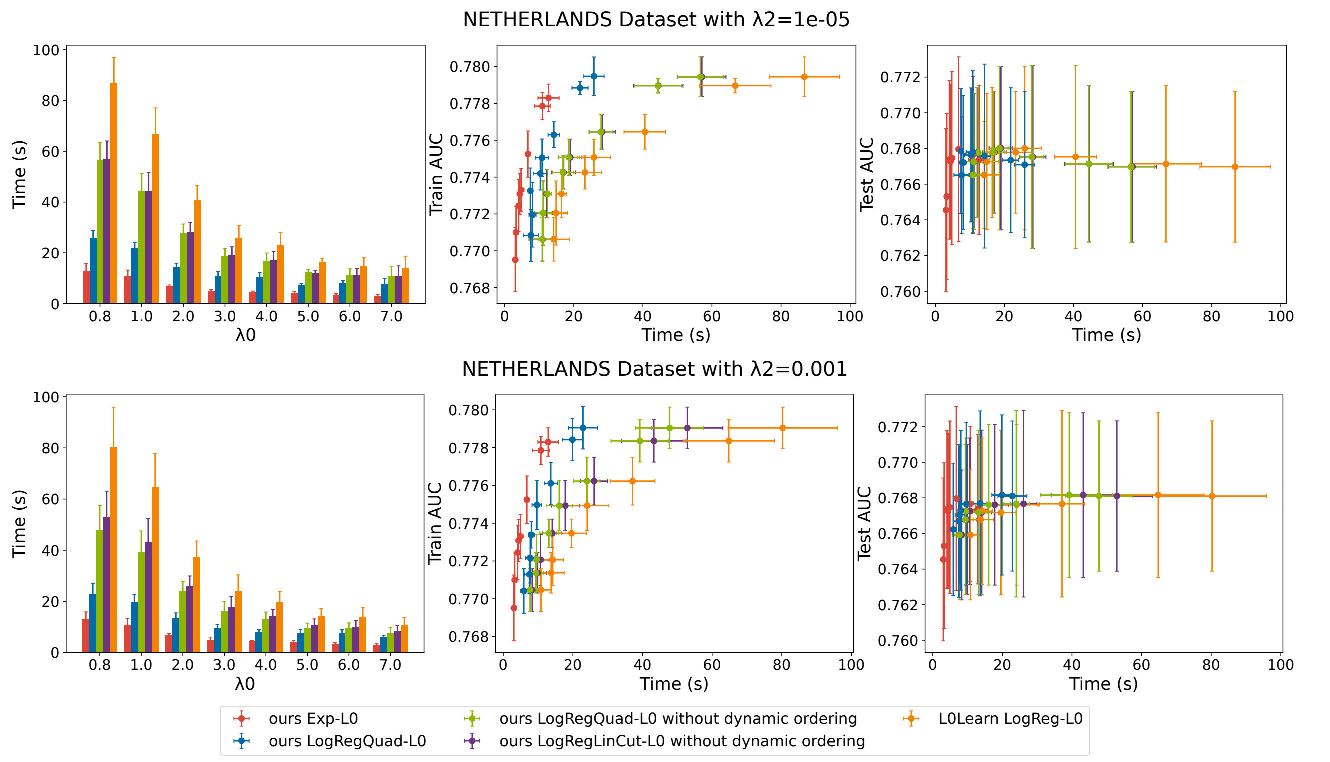

Figure 7: Computational times of different methods. "Exp" stands for exponential loss, "LogReg" stands for logistic loss, and "Quad" stands for quadratic cuts. Note that there is no ℓ_2 penalty for the exponential loss. *Our Exp-L0 method is generally about 4 times faster than L0Learn.* Note that the AUC axes indicate practically similar performance for these particular methods; the training time is what differentiates the methods.

D.3 Support versus AUC, Accuracy, and F1 Score

We provide the full regularization paths for the FICO dataset (see Figure 8). Our first method (quadratic cut + dynamic ordering) and second method (exponential loss) obtain high-quality solutions and their AUC and accuracy curves are similar to those from other methods. Our methods have computational advantages over L0Learn, as shown in Section D.2.

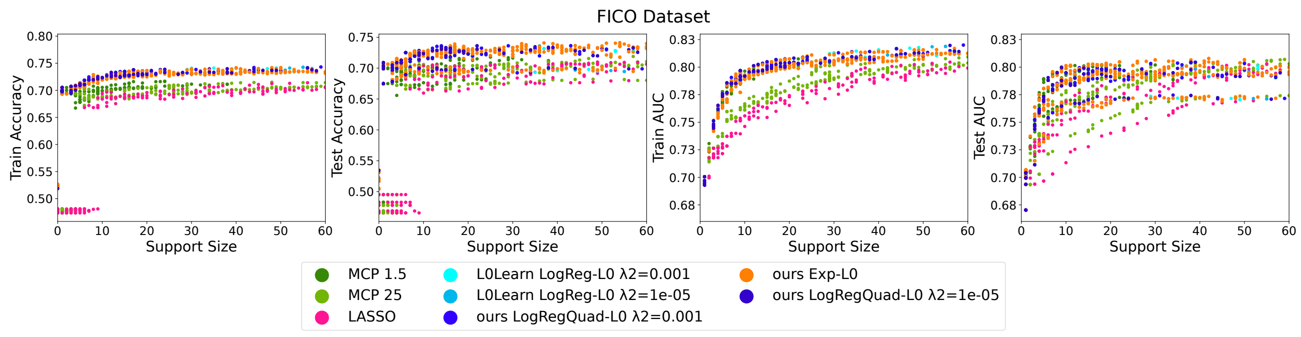

Figure 8: Results on the FICO dataset. See Figure 5 in the main paper for results on the COMPAS and NETHERLANDS datasets. The L0Learn points are mostly overlapping with points from our methods.

To investigate whether a small ℓ_2 regularization would help with the LASSO baseline, we provide a comparison between our method and the the ElasticNet method on the highly correlated synthetic dataset for the classification task. We used the glmnet R package for the ElasticNet baseline. The hyperparameter $\alpha \in [0, 1]$ controls the balance between ℓ_1 and ℓ_2 regularization. The LASSO method corresponds to $\alpha = 1.0$. Besides $\alpha = 1.0$, we also consider $\alpha \in \{0.9, 0.7, 0.5, 0.3, 0.1, 0.001\}$. As shown in Figure 9, a small ℓ_2 regularization term does not improve the LASSO method much. When α decreases, the solution quality degrades. This is potentially because more ℓ_2 regularization leads to non-sparse solutions with small coefficients, neither of which will lead to better performance here.

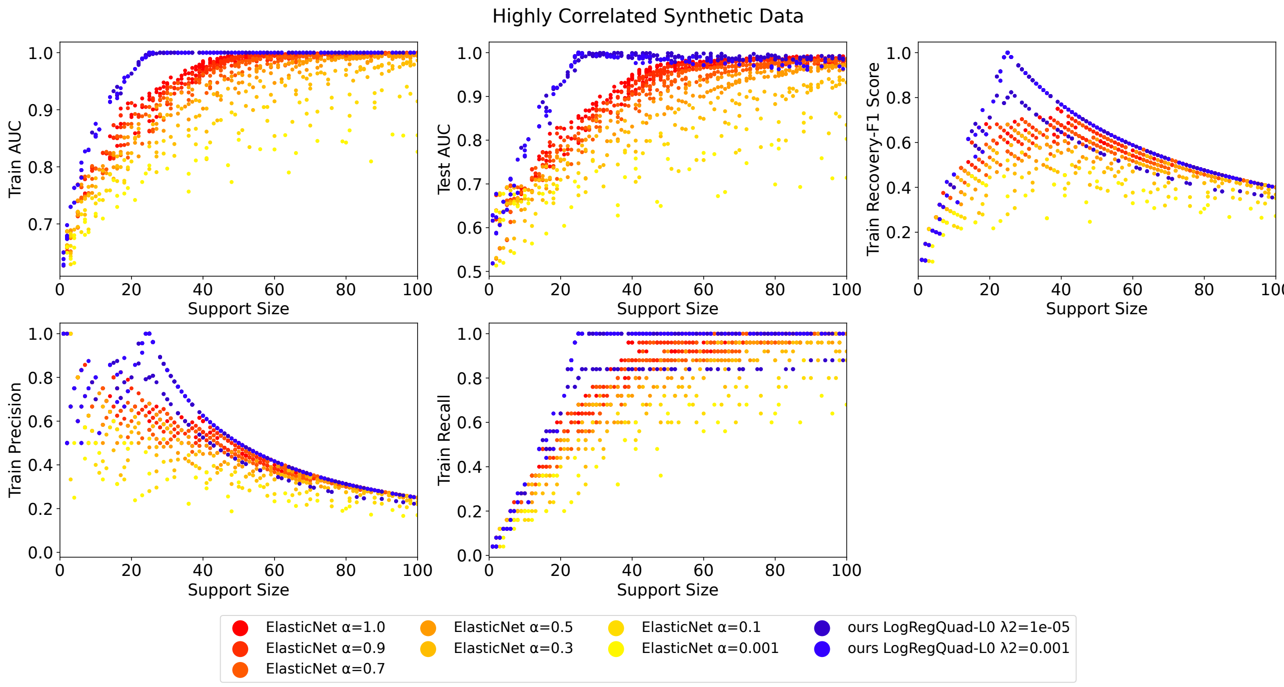

Figure 9: ElasticNet and our methods' results on the highly correlated synthetic dataset.

D.4 Samples of Sparse Models on the FICO and NETHERLANDS datasets

We provide some sample sparse models produced by minimizing the exponential loss and minimizing the logistic loss (quadratic cut + dynamic ordering) on the FICO and NETHERLANDS datasets.

The FICO dataset has 10459 samples and 1917 features. The NETHERLANDS dataset has 20000 samples and 2024 features. All models were developed from the third fold of our 5-fold cross validation split.

FICO Baseline Performance: The sparse models below approximately match the performance of black-box models shown in previous works (Chen et al., 2021). We also ran a GBDT model (Friedman, 2001) with max depth set to be 3 and number of boosting stages set to be 100. The AUC on the training set is 0.8318 ± 0.0028, and the AUC on the test set is 0.7959 ± 0.0133. The result on the test set is comparable to what we have shown in Figure 3. The models are in Figures 10-11.

NETHERLANDS Baseline Performance: The sparse models below approximately match the performance of black-box models. We ran a GBDT model (Friedman, 2001) with max depth set to be 3 and number of boosting stages set to be 100. The AUC on the training set is 0.7850 ± 0.0014, and the AUC on the test set is 0.7696 ± 0.0062. The result on the test set is comparable to what we have shown in Figure 7. (For the NETHERLANDS dataset, ages are collected in terms of months. That is why age thresholds are shown with float numbers.) The models are in Figures 12-13.

FICO model using the exponential loss:

$\lambda_0 = 5$:

$$
\begin{aligned}
score = & -0.2584626 \\
& +0.1825955 \times \mathbf{1}_{A \le 63} + 0.1387806 \times \mathbf{1}_{A \le 70} \\
& +0.2286364 \times \mathbf{1}_{A \le 74} + 0.2569742 \times \mathbf{1}_{A \le 83} && \# \; A: \textit{ExternalRiskEstimate} \\
& +0.1840013 \times \mathbf{1}_{B \le 51} + 0.172138 \times \mathbf{1}_{B \le 75} && \# \; B: \textit{AverageMInFile} \\
& +0.2015039 \times \mathbf{1}_{C \le 13} + 0.1923697 \times \mathbf{1}_{C \le 31} && \# \; C: \textit{NumSatisfactoryTrades} \\
& +0.2654667 \times \mathbf{1}_{D \le 96} && \# \; D: \textit{PercentTradesNeverDelq} \\
& +0.2320259 \times \mathbf{1}_{E \le 33} && \# \; E: \textit{MSinceMostRecentDelq} \\
& +0.1009372 \times \mathbf{1}_{F \le 8} && \# \; F: \textit{NumTotalTrades} \\
& -0.2311165 \times \mathbf{1}_{G \le 46} && \# \; G: \textit{PercentInstallTrades} \\
& -0.7723769 \times \mathbf{1}_{H \le -8} + 0.3636577 \times \mathbf{1}_{H \le 0} && \# \; H: \textit{MSinceMostRecentInqexcl7days} \\
& -0.2762694 \times \mathbf{1}_{I \le 5} && \# \; I: \textit{NumInqLast6M} \\
& -0.1897788 \times \mathbf{1}_{J \le 37} - 0.2742168 \times \mathbf{1}_{J \le 73} && \# \; J: \textit{NetFractionRevolvingBurden} \\
& -0.1038025 \times \mathbf{1}_{K \le 5} - 0.1938047 \times \mathbf{1}_{K \le 7} && \# \; K: \textit{NumRevolvingTradesWBalance}
\end{aligned}
$$

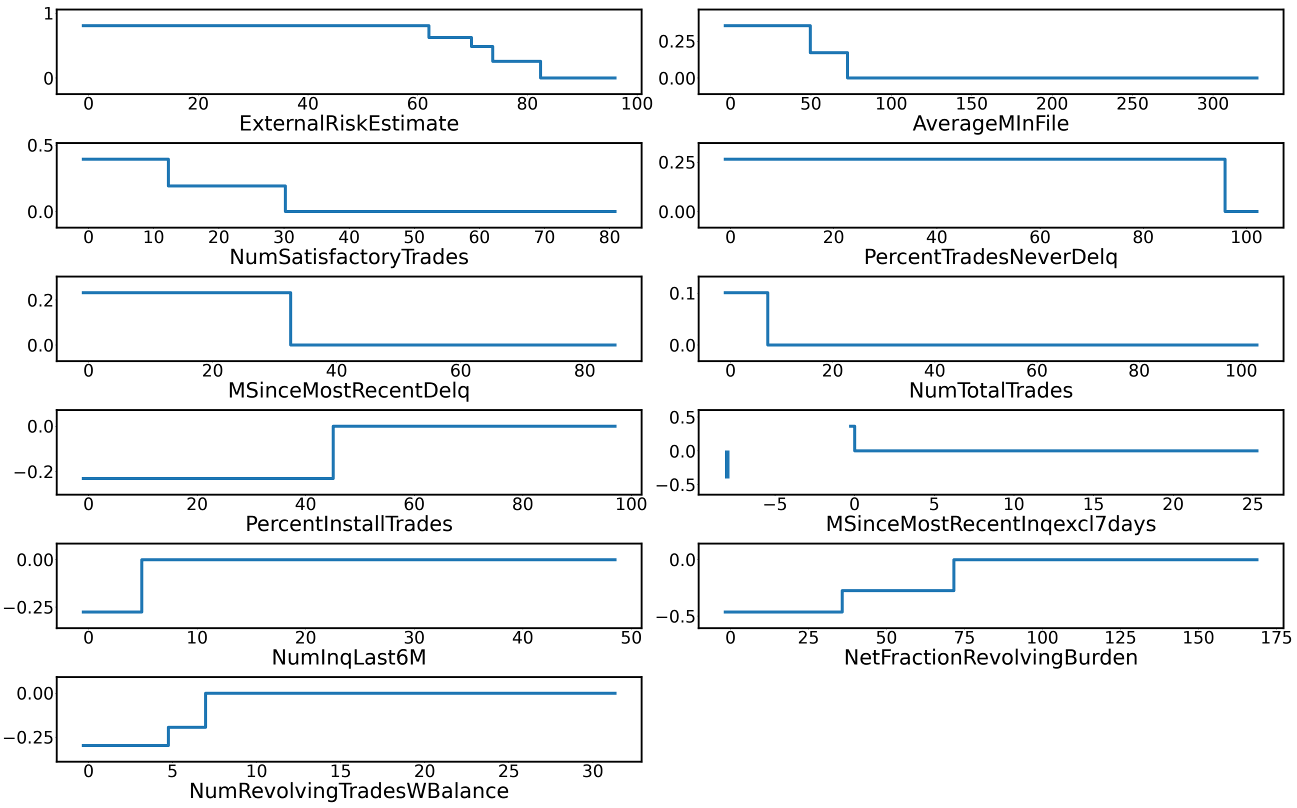

Figure 10: FICO score contributions with the exponential loss and $\lambda_0 = 5$. Training duration is 3.15 seconds. Note that no monotonicity constraints were imposed.

FICO model using the logistic loss (quadratic cut + dynamic ordering):
$\lambda_0 = 5$, $\lambda_2 = 0.001$:

$$\begin{aligned}
score =& 2.805021 \\
& + 0.4071199 \times \mathbf{1}_{A\leq 63} + 0.310368 \times \mathbf{1}_{A\leq 70} \\
& + 0.4604512 \times \mathbf{1}_{A\leq 74} + 0.5471219 \times \mathbf{1}_{A\leq 83} && \# \ A \ \text{:} ExternalRiskEstimate \\
& + 0.408959 \times \mathbf{1}_{B\leq 51} + 0.3283239 \times \mathbf{1}_{B\leq 75} && \# \ B \ \text{:} AverageMInFile \\
& + 0.4225237 \times \mathbf{1}_{C\leq 13} + 0.3396898 \times \mathbf{1}_{C\leq 31} && \# \ C \ \text{:} NumSatisfactoryTrades \\
& + 0.525166 \times \mathbf{1}_{D\leq 96} && \# \ D \ \text{:} PercentTradesNeverDelq \\
& + 0.4427697 \times \mathbf{1}_{E\leq 33} && \# \ E \ \text{:} MSinceMostRecentDelq \\
& - 0.4317725 \times \mathbf{1}_{F\leq 46} && \# \ F \ \text{:} PercentInstallTrades \\
& - 1.576435 \times \mathbf{1}_{G\leq -8} + 0.5045199 \times \mathbf{1}_{G\leq 0} \\
& + 0.2874494 \times \mathbf{1}_{G\leq 1} && \# \ G \ \text{:} MSinceMostRecentInqexcl7days \\
& - 3.97116 \times \mathbf{1}_{H\leq 11} && \# \ H \ \text{:} NumInqLast6M \\
& - 0.3657186 \times \mathbf{1}_{I\leq 37} - 0.5681891 \times \mathbf{1}_{I\leq 73} && \# \ I \ \text{:} NetFractionRevolvingBurden \\
& - 0.4969551 \times \mathbf{1}_{J\leq 7} && \# \ J \ \text{:} NumRevolvingTradesWBalance
\end{aligned}$$

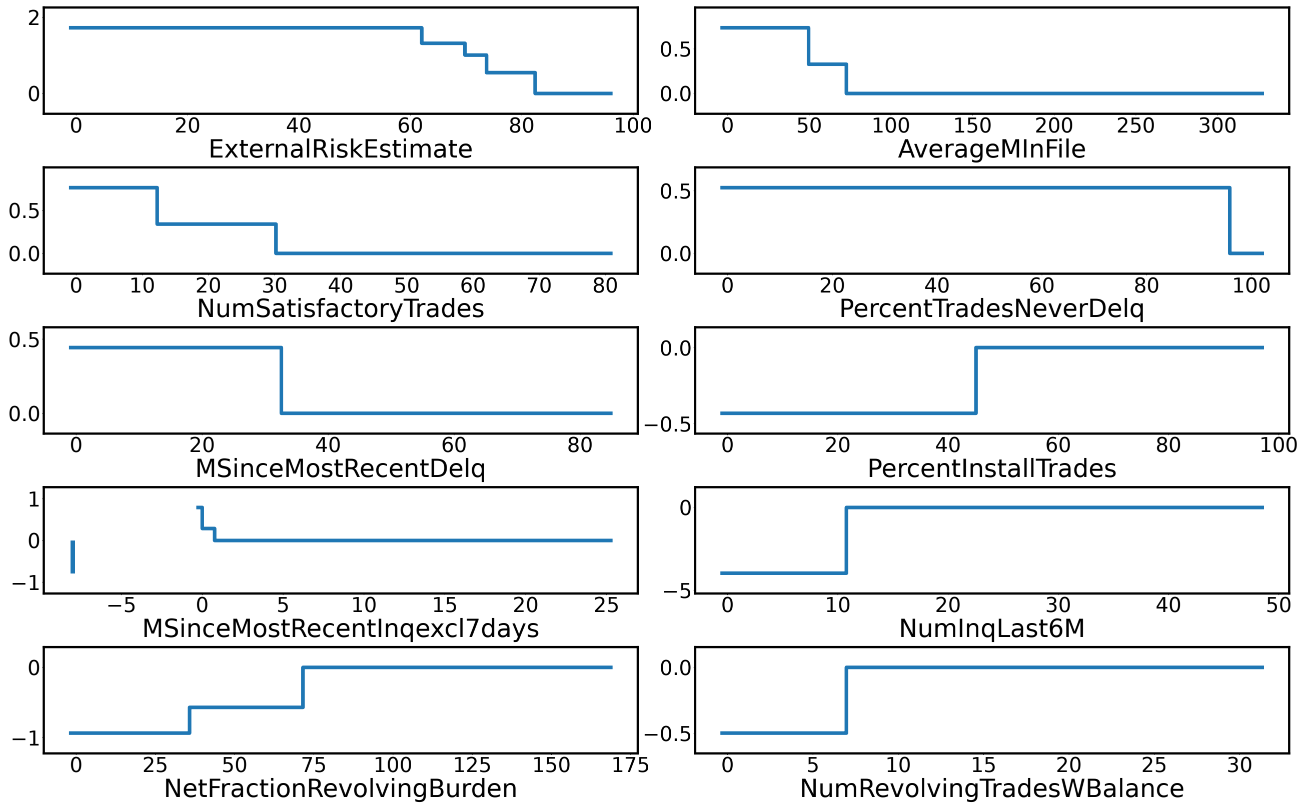

Figure 11: FICO score contributions with the logistic loss and $\lambda_0 = 5, \lambda_2 = 0.001$. Training duration is 5.95 seconds.

NETHERLANDS model using the exponential loss:

$\lambda_0 = 7$:

$$\begin{aligned}
score = & 3.114342 \\
& - 0.1439394 \times \mathbf{1}_{A==female} && \# \ A \ :sex \\
& - 0.4739628 \times \mathbf{1}_{B\leq 0.0} - 0.3336059 \times \mathbf{1}_{B\leq 1.098612289} \\
& - 0.3083761 \times \mathbf{1}_{B\leq 1.609437912} && \# \ B \ :log \ \# \ of \ previous \ penal \ cases \\
& + 0.2887266 \times \mathbf{1}_{C\leq 22.26146475} + 0.2354507 \times \mathbf{1}_{C\leq 28.48266213076} \\
& + 0.1951787 \times \mathbf{1}_{C\leq 39.07432555374} + 0.2304243 \times \mathbf{1}_{C\leq 46.91581109} && \# \ C \ :age \ in \ years \\
& - 0.1674992 \times \mathbf{1}_{D\leq 28.069209540720003} && \# \ D \ :age \ at \ first \ penal \ case \\
& + 0.1526613 \times \mathbf{1}_{E\leq 7.0} && \# \ E \ :offence \ type \\
& - 1.381801 \times \mathbf{1}_{F\leq 0.0} && \# \ F \ :11\text{-}20 \ previous \ case \\
& - 1.856642 \times \mathbf{1}_{G\leq 0.0} && \# \ G \ :>20 \ previous \ case
\end{aligned}$$

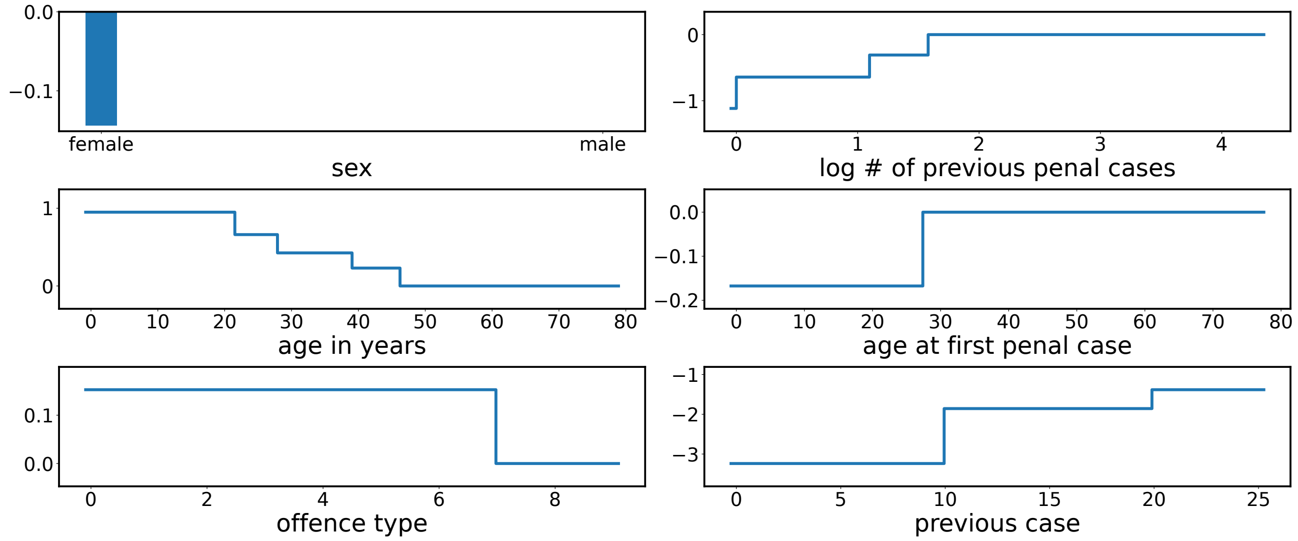

Figure 12: NETHERLANDS score contributions with the exponential loss and $\lambda_0 = 7$. Training duration is 2.73 seconds. Note that there are no monotonicity constraints imposed.

NETHERLANDS model using the logistic loss (quadratic cut + dynamic ordering):
$\lambda_0 = 7$, $\lambda_2 = 0.001$:

$$
\begin{aligned}
score = & 6.259994 \\
& - 0.3092142 \times \mathbf{1}_{A==female} && \# \ A\ :sex \\
& - 0.7374188 \times \mathbf{1}_{B\leq 0.0} - 0.4302188 \times \mathbf{1}_{B\leq 0.693147181} \\
& - 0.2888496 \times \mathbf{1}_{B\leq 1.098612289} - 0.3933033 \times \mathbf{1}_{B\leq 1.386294361} \\
& - 0.5383587 \times \mathbf{1}_{B\leq 1.945910149} && \# \ B\ :log\ \#\ of\ previous\ penal\ cases \\
& + 0.3877289 \times \mathbf{1}_{C\leq 18.94046991832} + 0.5554352 \times \mathbf{1}_{C\leq 23.01017483608} \\
& + 0.4700141 \times \mathbf{1}_{C\leq 31.552317465359998} + 0.6324188 \times \mathbf{1}_{C\leq 43.91512663} && \# \ C\ :age\ in\ years \\
& - 0.2645467 \times \mathbf{1}_{D\leq 27.986380572549994} && \# \ D\ :age\ at\ first\ penal\ case \\
& + 0.2861914 \times \mathbf{1}_{E\leq 7.0} && \# \ E\ :offence\ type \\
& - 2.655844 \times \mathbf{1}_{F\leq 0.0} && \# \ F\ :11\text{-}20\ previous\ case \\
& - 3.605789 \times \mathbf{1}_{G\leq 0.0} && \# \ G\ :>20\ previous\ case
\end{aligned}
$$

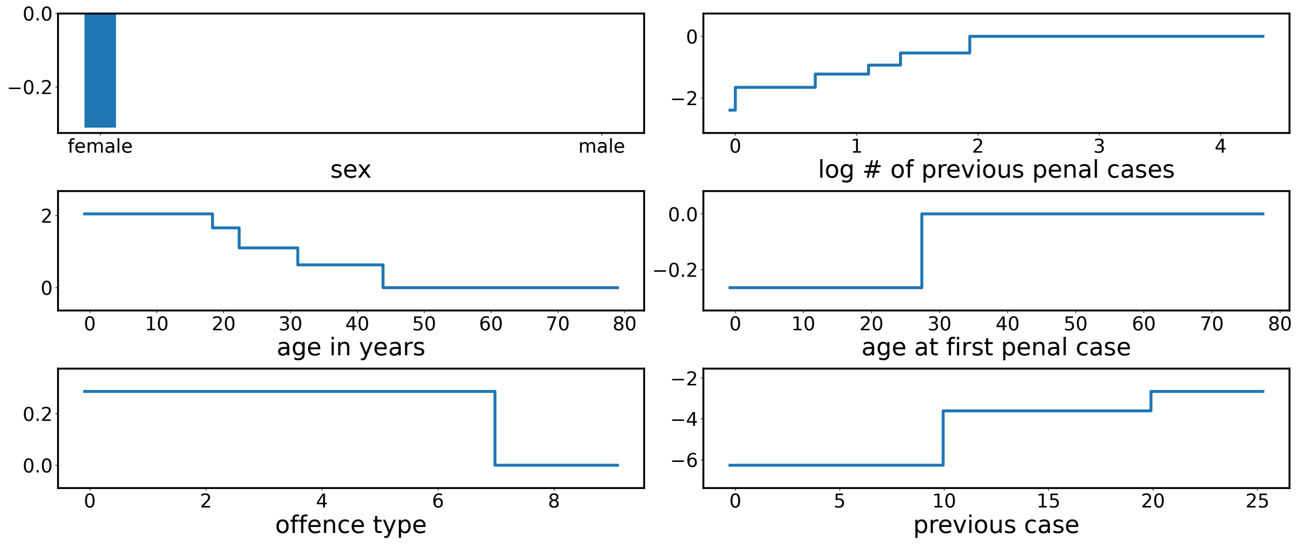

Figure 13: NETHERLANDS model score contributions with logistic loss, $\lambda_0 = 7$, and $\lambda_2 = 0.001$. Training duration is 7.2 seconds.