Power Transform Revisited: Numerically Stable, and Federated

Xuefeng Xu
University of Warwick

Graham Cormode
University of Oxford

Abstract

Power transforms are popular parametric methods for making data more Gaussian-like, and are widely used as preprocessing steps in statistical analysis and machine learning. However, we find that direct implementations of power transforms suffer from severe numerical instabilities, which can lead to incorrect results or even crashes. In this paper, we provide a comprehensive analysis of the sources of these instabilities and propose effective remedies. We further extend power transforms to the federated learning setting, addressing both numerical and distributional challenges that arise in this context. Experiments on real-world datasets demonstrate that our methods are both effective and robust, substantially improving stability compared to existing approaches.

1 INTRODUCTION

Power transforms are widely used to stabilize variance and reduce skewness, and are described in textbooks as foundational tools for data preprocessing. Among them, the Box-Cox (Box and Cox, 1964) and Yeo-Johnson (Yeo and Johnson, 2000) transforms are the most prominent, and are frequently employed in statistical analysis and machine learning pipelines (Ruppert, 2001; Fink, 2009). Despite their popularity, directly implementing these mathematical functions leads to serious numerical instabilities, producing erroneous outputs or even causing program crashes. This is not a theoretical concern: as we show in Table 1, simple inputs can easily provoke such failures.

The issue has been identified in prior work such as the MASS package (Venables and Ripley, 2002), but only partial solutions have been proposed. Recently, this issue was raised by Marchand et al. (2022). Unfortunately, their analysis includes some incorrect claims and unsound solutions. As shown in Figure 4, their method fails to identify optimal parameters and can crash even on simple datasets. A very recent study by Barron (2025) also mentions numerical issues, but their remedy relies on simple replacement of the numerical function, which remains insufficient to ensure stability. Consequently, this foundational question has been unanswered until now.

In this paper, we provide a comprehensive analysis of numerical instabilities in power transforms. Our solution includes log-domain computation, careful reformulation of critical expressions, and constraints on extreme parameter values, all of which are essential to ensure robust numerical behavior. Guided by our theoretical analysis, we also construct adversarial datasets that systematically trigger overflows under different floating-point precisions. Such data can occur naturally in real-world settings (e.g., Figure 13 in the Appendix) or be deliberately introduced by adversarial clients in federated learning scenarios.

We further extend power transforms to the federated setting, where distributed data introduces unique challenges. We design a numerically stable one-pass variance aggregation method for computing the negative log-likelihood (NLL), enabling robust and efficient optimization of transformation parameters across multiple clients with minimal communication overhead.

Extensive experiments on real-world datasets validate the effectiveness of our approach. Compared to existing implementations, including those of Marchand et al. (2022), our methods consistently achieve superior stability in both centralized and federated settings.

Our contributions are:

1. We conduct a comprehensive analysis of numerical instabilities in power transforms, together with recipes for constructing adversarial datasets that expose these weaknesses.
2. We propose remedies addressing each source of in-

Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300.

stability, yielding numerically stable algorithms in both centralized[1] and federated learning settings.

3. We perform extensive experimental validation on real-world datasets, demonstrating the effectiveness and robustness of our methods.

The rest of the paper is organized as follows. Section 2 reviews preliminaries. Section 3 explains the root causes of instability, while Section 4 presents our remedies. Section 5 extends the approach to federated learning. Experimental results are presented in Section 6, further issues are discussed in Section 7, and Section 8 concludes.

2 PRELIMINARIES

2.1 Power Transform

The Box-Cox (BC) transformation (Box and Cox, 1964) is a widely used power transform requiring strictly positive inputs ($x > 0$):

$$\psi_{\mathrm{BC}}(\lambda, x) = \begin{cases} \frac{x^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0, \\ \ln x & \text{if } \lambda = 0. \end{cases} \tag{1}$$

The Yeo-Johnson (YJ) transformation (Yeo and Johnson, 2000) extends BC to accommodate both positive and negative values:

$$\psi_{\mathrm{YJ}}(\lambda, x) = \begin{cases} \frac{(x+1)^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0, x \geq 0, \\ \ln(x+1) & \text{if } \lambda = 0, x \geq 0, \\ \frac{(-x+1)^{2-\lambda} - 1}{\lambda - 2} & \text{if } \lambda \neq 2, x < 0, \\ -\ln(-x+1) & \text{if } \lambda = 2, x < 0. \end{cases} \tag{2}$$

The parameter λ controls the shape of the transformation; when $\lambda \neq 1$, the mapping is nonlinear and alters the distribution. Figure 1 shows the transformation curves for various λ, while Figure 2 illustrates Box-Cox's effect on left- and right-skewed data.

The parameter λ is typically estimated by minimizing the negative log-likelihood (NLL):

$$\mathrm{NLL}_{\mathrm{BC}} = (1-\lambda)\sum_{i=1}^{n} \ln x_i + \frac{n}{2} \ln \sigma^2_{\psi_{\mathrm{BC}}}, \tag{3}$$

$$\mathrm{NLL}_{\mathrm{YJ}} = (1-\lambda)\sum_{i=1}^{n} \operatorname{sgn}(x_i) \ln(|x_i| + 1) + \frac{n}{2} \ln \sigma^2_{\psi_{\mathrm{YJ}}}, \tag{4}$$

where $\sigma^2_\psi = \mathrm{Var}[\psi(\lambda, x)]$. Both NLL functions are strictly convex (Kouider and Chen, 1995; Marchand et al., 2022), guaranteeing a unique optimum. *SciPy* (Virtanen et al., 2020) estimates λ^* using Brent's method (Brent, 2013), a derivative-free optimizer with superlinear convergence.

[1]Our implementation has been merged into *SciPy*.

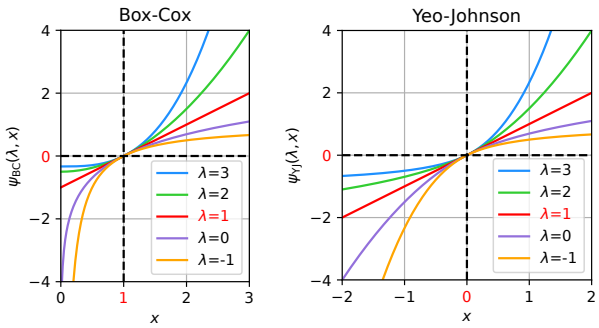


Figure 1: Transformation functions for different λ.

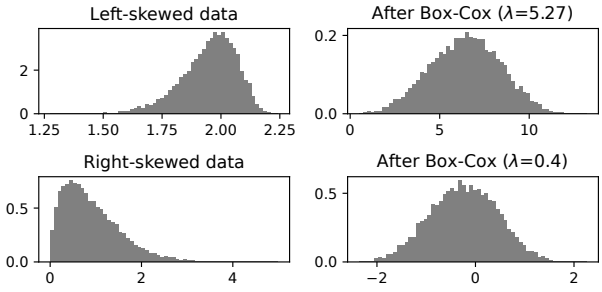


Figure 2: Box-Cox transformation on left- (top) and right-skewed (bottom) data.

2.2 Numerically Stable Variance Computation

Variance is computed as $\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$, where $\bar{x}$ is the sample mean. This requires two passes over the data, which is inefficient for large datasets or streaming data. The equivalent one-pass form,

$$\frac{1}{n}\sum_{i=1}^{n} x_i^2 - \frac{1}{n^2}\left(\sum_{i=1}^{n} x_i\right)^2, \tag{5}$$

is well known to be numerically unstable (Ling, 1974; Chan et al., 1983).

A numerically stable one-pass approach (Chan et al., 1982) maintains $(n, \bar{x}, s)$ triples, where $s = \sum_{i=1}^{n}(x_i - \bar{x})^2$. Merging two partial aggregates $(n^{(A)}, \bar{x}^{(A)}, s^{(A)})$ and $(n^{(B)}, \bar{x}^{(B)}, s^{(B)})$ is done as:

$$n^{(AB)} = n^{(A)} + n^{(B)}, \tag{6}$$

$$\bar{x}^{(AB)} = \bar{x}^{(A)} + (\bar{x}^{(B)} - \bar{x}^{(A)}) \cdot \frac{n^{(B)}}{n^{(AB)}}, \tag{7}$$

$$s^{(AB)} = s^{(A)} + s^{(B)} + (\bar{x}^{(B)} - \bar{x}^{(A)})^2 \cdot \frac{n^{(A)} n^{(B)}}{n^{(AB)}}. \tag{8}$$

This method can be applied sequentially, but a tree-structured aggregation (Figure 3) minimizes error accumulation and improves numerical stability.

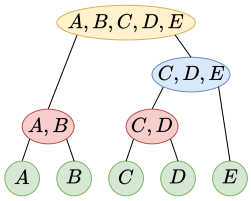


Figure 3: Tree-structured variance aggregation.

2.3 Federated Learning

Federated learning (Kairouz et al., 2021) is a distributed paradigm where multiple clients collaboratively train a model without sharing raw data, thereby preserving privacy. Each client computes local updates from its own data and transmits them to a central server, which aggregates the information to update the global model. Communication efficiency is a key concern, as excessive exchanges can be costly; methods such as FedAvg (McMahan et al., 2017) are commonly used to reduce communication overhead.

3 UNDERSTANDING NUMERICAL INSTABILITIES

This section examines numerical instabilities in power transforms. We begin by explaining why such instabilities matter and how they can disrupt optimization. We then introduce adversarial datasets for testing implementation robustness.

Numerical instabilities in power transforms arise in the computation of the NLL functions defined in Equations (3) and (4). These instabilities can disrupt optimization, producing suboptimal or even incorrect results. In practice, this means that the output of the method becomes unusable, causing bugs or failures in downstream tasks. Moreover, these issues cannot be resolved by simple quick fixes, as the instability is fundamental to the underlying computation and requires careful redesign for robust operation. As illustrated in Figure 4, the Exponential Search method (Marchand et al., 2022)[2] fails even on simple datasets. The true optimum λ^* can be quite large in magnitude, causing numerical overflow when evaluating the power functions, e.g., 0.1^{-361} or 10^{358}. As a result, ExpSearch will crash and return values that diverge significantly from the true optimum.

To better understand these issues, we first summarize key properties of the BC transformation $\psi(\lambda, x)$ in Theorem 3.1; the proof is intricate and deferred to Appendix A. Analogous results for YJ can be found in Yeo (1997); Yeo and Johnson (2000).

[2]There is a mismatch between the sign formula (Equation 3) and the `ExpUpdate` method (Algorithm 2) in (Marchand et al., 2022). The sign formula is $\partial \mathrm{NLL}_{\mathrm{YJ}}/\partial\lambda$ (not $\partial \ln \mathcal{L}_{\mathrm{YJ}}/\partial\lambda$); therefore, $\Delta = 1$ in `ExpUpdate` should instead be $\Delta = -1$. Alternatively, one could add a negative sign in Equation 3 and keep `ExpUpdate` unchanged.

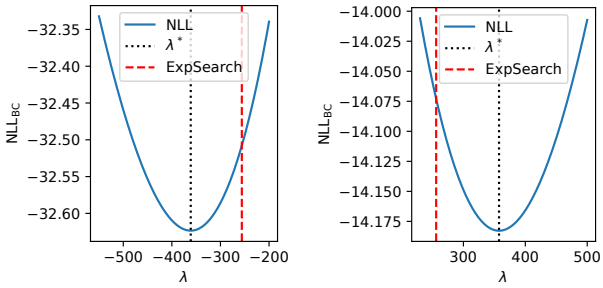


Figure 4: ExpSearch fails to find the true optimum λ^*. Left: data $[0.1, 0.1, 0.1, 0.101]$, $\lambda^* \approx -361$; Right: data $[10, 10, 10, 9.9]$, $\lambda^* \approx 358$.

Theorem 3.1. *The Box-Cox transformation $\psi(\lambda, x)$ defined in (1) has the following properties:*

1. *$\psi(\lambda, x) \geq 0$ for $x \geq 1$, and $\psi(\lambda, x) < 0$ for $x < 1$.*
2. *$\psi(\lambda, x)$ is convex in x for $\lambda > 1$ and concave in x for $\lambda < 1$.*
3. *$\psi(\lambda, x)$ is a continuous function of (λ, x).*
4. *If $\psi^{(k)} = \partial^k \psi(\lambda, x)/\partial\lambda^k$ then, for $k \geq 1$:*

$$\psi^{(k)} = \begin{cases} [x^\lambda (\ln x)^k - k\psi^{(k-1)}]/\lambda & \text{if } \lambda \neq 0, \\ (\ln x)^{k+1}/(k+1) & \text{if } \lambda = 0. \end{cases}$$

   $\psi^{(k)}$ is continuous in (λ, x) and $\psi^{(0)} \equiv \psi(\lambda, x)$.
5. *$\psi(\lambda, x)$ is increasing in both λ and x.*
6. *$\psi(\lambda, x)$ is convex in λ for $x > 1$ and concave in λ for $x < 1$.*

These properties help pinpoint instability sources and guide the construction of adversarial datasets that trigger numerical overflow (Table 1) under various floating-point precisions. Section 4 presents a numerically stable formulation to address these issues. In summary, there is a simple recipe that can lead to numerical overflow:

Staying on One Side of Zero Points. By Theorem 3.1.1, $\psi(\lambda, 1) = 0$ for all λ. Thus, $x = 1$ should be avoided. Choosing all $x > 1$ ensures $\psi(\lambda, x) > 0$ and may lead to *positive* overflow; similarly, choosing all $x < 1$ can yield *negative* overflow. Datasets containing both $x > 1$ and $x < 1$ may be less likely to

Table 1: Adversarial datasets causing negative or positive overflow under different floating-point precisions.

Transf.	Overflow	Adversarial Data	λ^*	NLL	Extreme $\psi(\lambda^*, x)$	Max Value
BC	Negative	[0.1, 0.1, 0.1, 0.109]	-41.70	23.91	-1.20e+40	
	Positive	[10, 10, 10, 9.2]	43.10	5.79	2.90e+41	3.40e+38
YJ	Negative	[-10, -10, -10, -9.1]	-40.10	5.32	-1.65e+42	(Single)
	Positive	[10, 10, 10, 9.1]	42.10	5.32	1.65e+42	
BC	Negative	[0.1, 0.1, 0.1, 0.101]	-361.15	32.62	-3.87e+358	
	Positive	[10, 10, 10, 9.9]	357.55	14.18	9.96e+354	1.80e+308
YJ	Negative	[-10, -10, -10, -9.9]	-391.49	14.18	-1.51e+407	(Double)
	Positive	[10, 10, 10, 9.9]	393.49	14.18	1.51e+407	
BC	Negative	[0.1, 0.1, 0.1, 0.10001]	-35936.9	51.03	-2.30e+35932	
	Positive	[10, 10, 10, 9.999]	35933.3	32.61	5.85e+35928	1.19e+4932
YJ	Negative	[-10, -10, -10, -9.999]	-39524.8	32.61	-2.29e+41158	(Quadruple)
	Positive	[10, 10, 10, 9.999]	39526.8	32.61	2.292e+41158	
BC	Negative	[0.1, 0.1, 0.1, 0.100001]	-359353.0	60.24	-2.74e+359347	
	Positive	[10, 10, 10, 9.9999]	359349.0	41.82	6.99e+359343	1.61e+78913
YJ	Negative	[-10, -10, -10, -9.9999]	-395283.0	41.82	-6.47e+411640	(Octuple)
	Positive	[10, 10, 10, 9.9999]	395285.0	41.82	6.47e+411640	

trigger overflow, and constructing simple adversarial examples in this regime is more challenging.

Extreme Skewness. For $\lambda > 1$, $\psi(\lambda, x)$ is convex and increasing in x (Theorem 3.1.2 and 3.1.5), stretching the right tail more than the left. Thus, left-skewed data tends to push $\lambda > 1$ toward positive overflow after transformation; conversely, right-skewed data tends to push $\lambda < 1$ toward negative overflow. Extreme skewness drives λ far from 1.

Small Variance. Tightly clustered data also drives λ to extreme values, as the transformation must expand interpoint distances to approach normality. For $x > 1$, $\psi(\lambda, x)$ is convex and increasing in λ (Theorem 3.1.5 and 3.1.6), so large λ favors positive overflow; for $x < 1$, it is concave and increasing, so small λ favors negative overflow. Combining extreme skewness with small variance (achieved by inserting duplicate values) is particularly effective at producing overflow.

As we show experimentally in Section 6, such conditions naturally occur in real datasets or can be deliberately created by adversarial clients to poison training.

4 NUMERICALLY STABLE POWER TRANSFORM

This section presents a numerically stable approach for power transforms, addressing different sources of instability and introducing corresponding remedies.

Modern computers use finite-precision floating-point arithmetic (e.g., IEEE 754 standard (IEEE, 2019)), which limits the range of representable values. As shown in Table 1, power transforms can generate extreme values that exceed these limits, resulting in numerical overflow. Therefore, direct computation of the NLL functions is prone to instability.

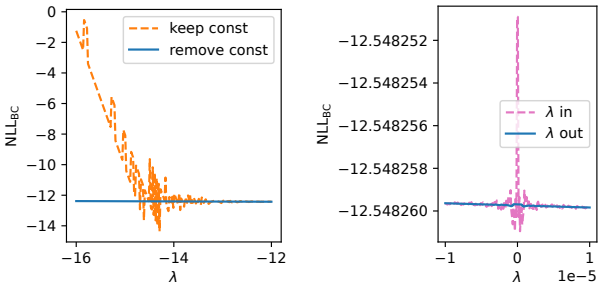
Beyond the representation issue, the choice of optimization method also plays a critical role. In particular, derivative-based methods, such as Exponential Search (Marchand et al., 2022), are prone to instability, whereas derivative-free methods (e.g., Brent's method) can achieve stable optimization. We detail these observations below.

Derivative-based methods are unstable. Consider Exponential Search, which requires the first-order derivative of the NLL function. For the Box-Cox transformation, this derivative is

$$\frac{\partial \mathrm{NLL}_{\mathrm{BC}}}{\partial \lambda} = \frac{1}{\sigma^2_{\psi_{\mathrm{BC}}}} \left(\sum_{i=1}^{n} \psi_{\mathrm{BC}}(\lambda, x_i) \cdot \psi^{(1)}_{\mathrm{BC}}(\lambda, x_i) - \frac{1}{n} \sum_{i=1}^{n} \psi_{\mathrm{BC}}(\lambda, x_i) \cdot \sum_{i=1}^{n} \psi^{(1)}_{\mathrm{BC}}(\lambda, x_i) \right) - \sum_{i=1}^{n} \ln x_i \quad (9)$$

Here, terms such as $\psi_{\mathrm{BC}}(\lambda, x)$, $\psi^{(1)}_{\mathrm{BC}}(\lambda, x)$ (Theorem 3.1.4), and $\sigma^2_{\psi_{\mathrm{BC}}}$ can easily overflow and cannot be computed reliably. Consequently, derivative-based optimization is not numerically stable.

Derivative-free methods are stable. By contrast, derivative-free methods rely only on evaluating the NLL function itself, which can be stabilized. In particular, the NLL function involves only $\ln \sigma^2_{\psi}$, which



(a) Eq. (10) vs. (11) (b) Eq. (11) vs. (12)

Figure 5: Comparison of NLL curves using different equations. Data used: [10, 10, 10, 9.9].

can be represented in floating-point format and efficiently computed using log-domain methods (Haberland, 2023) (see Appendix B).

Modifying the Variance Computation. While log-domain computation mitigates overflow, additional instabilities remain. Specifically, the computation of $\ln \sigma^2_{\psi_{\mathrm{BC}}}$ requires careful reformulation:

$$\ln \sigma^2_{\psi_{\mathrm{BC}}} = \ln \mathrm{Var}[(x^\lambda - 1)/\lambda] \tag{10}$$
$$= \ln \mathrm{Var}(x^\lambda/\lambda) \tag{11}$$
$$= \ln \mathrm{Var}(x^\lambda) - 2\ln|\lambda| \tag{12}$$

Equation (11) removes the constant $-1/\lambda$, which avoids catastrophic cancellation (Weckesser, 2019). For example, with $\lambda < -14$ and $x > 1$, we have $x^\lambda \to 0$, and the subtraction $x^\lambda - 1$ leads to severe precision loss. Therefore, computing the variance after the transformation becomes very unstable. The same occurs for large λ with $x < 1$. This instability, illustrated in the left panel of Figure 5, shows large fluctuations in the NLL curve when keeping the constant term, which will disrupt optimization whenever the evaluated λ lies in such regions.

Further, as $\lambda \to 0$, computing x^λ/λ yields extreme values, destabilizing the variance computation. Factoring out λ, as in (12), restores stability. The right panel of Figure 5 demonstrates this improvement, where the NLL curve becomes smooth after factoring out λ.

Bounding extreme values. Finally, the optimal λ^* may still yield extreme transformed values (Table 1). To prevent this, we impose constraints during optimization. Since the transformation is monotone in both λ and x (Theorem 3.1.5), we restrict the transformed data to lie within $[-y_B, y_B]$ by bounding λ according to $x_{\max}$ and $x_{\min}$:

$$\begin{aligned} \min_{\lambda} \quad & \mathrm{NLL}_{\mathrm{BC}}(\lambda, x) \\ \text{s.t.} \quad & \lambda \le \psi^{-1}_{\mathrm{BC}}(x_{\max}, y_B) \quad \text{if } x_{\max} > 1, \\ & \lambda \ge \psi^{-1}_{\mathrm{BC}}(x_{\min}, -y_B) \text{ if } x_{\min} < 1. \end{aligned} \tag{13}$$

Here, ψ^{-1}_{BC} is the inverse Box-Cox transform, expressed via the Lambert W function (Corless et al., 1996):

$$\psi^{-1}_{\mathrm{BC}}(x, y) = -\tfrac{1}{y} - \tfrac{1}{\ln x} W\left(-\tfrac{x^{-1/y}\ln x}{y}\right). \tag{14}$$

Since the Lambert W function has two real branches ($k = 0$ for $W(x) \ge -1$, and $k = -1$ for $W(x) \le -1$), we use the $k = -1$ branch in overflow cases ($x > 1, \lambda \gg 1$ or $x < 1, \lambda \ll 1$), where

$$W\left(-\tfrac{x^{-1/y}\ln x}{y}\right) = -(\lambda + \tfrac{1}{y})\ln x \tag{15}$$
$$\approx -\lambda \ln x = -\ln(x^\lambda) \ll -1. \tag{16}$$

The same approach applies to the Yeo-Johnson transformation (see Appendix C). The default bound y_B can be set to the maximum representable floating-point value to prevent overflow after transformation. Practitioners may select a smaller bound based on domain-specific requirements or constraints on downstream models.

5 FEDERATED POWER TRANSFORM

This section extends power transforms to the federated learning setting. We begin by explaining the challenges of federated NLL evaluation, then show how textbook variance computation can cause numerical instability in this context. Finally, we introduce a numerically stable variance aggregation method that enables reliable NLL evaluation under federation.

In federated learning, the objective is to find the global optimum λ^* that minimizes the NLL across multiple clients, each holding its own local dataset. Standard federated optimization methods such as FedAvg do not apply here: each client may have a different local optimum λ^*_j, and averaging them does not in general recover the global optimum. For heterogeneous data distributions, the averaged value may diverge significantly from the true global solution. Consequently, the server must evaluate the NLL function on aggregated statistics and apply a derivative-free optimizer (e.g., Brent's method) to locate the global optimum.

To reduce communication rounds, one-pass variance computation is preferred. However, this approach introduces severe numerical instabilities, which directly affect both NLL evaluation and the optimization of

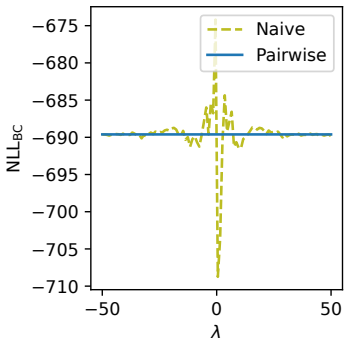


Figure 6: Comparison of NLL curves using the naive and pairwise aggregation. Data are synthetic Gaussian samples from $\mathcal{N}(10^4, 10^{-3})$ with 100 points.

Algorithm 1 Variance Aggregation

Input: Queue Q containing $(n^{(j)}, \bar{y}^{(j)}, s^{(j)})$
while $|Q| > 1$ **do**
    Dequeue $(n^{(A)}, \bar{y}^{(A)}, s^{(A)})$, $(n^{(B)}, \bar{y}^{(B)}, s^{(B)})$
    Compute $(n^{(AB)}, \bar{y}^{(AB)}, s^{(AB)})$ using **Pairwise**
    Enqueue $(n^{(AB)}, \bar{y}^{(AB)}, s^{(AB)})$ back into Q
end while
return the final $(N, \bar{Y}, S)$

Pairwise:
Input: $(n^{(A)}, \bar{y}^{(A)}, s^{(A)})$ and $(n^{(B)}, \bar{y}^{(B)}, s^{(B)})$
$n^{(AB)} \leftarrow n^{(A)} + n^{(B)}$
$\delta \leftarrow \bar{y}^{(B)} - \bar{y}^{(A)}$
$\bar{y}^{(AB)} \leftarrow \bar{y}^{(A)} + \delta \cdot \frac{n^{(B)}}{n^{(AB)}}$
$s^{(AB)} \leftarrow s^{(A)} + s^{(B)} + \delta^2 \cdot \frac{n^{(A)} n^{(B)}}{n^{(AB)}}$
return $(n^{(AB)}, \bar{y}^{(AB)}, s^{(AB)})$

λ. To illustrate this, we compare two approaches: the naive one-pass method (Equation (5), also used in Marchand et al. (2022)) and our numerically stable aggregation procedure (Algorithm 1). As shown in Figure 6, the naive method produces large fluctuations in the NLL curve, disrupting optimization. In contrast, our stable approach yields smooth curves and enables reliable optimization.

For numerically stable federated NLL computation with Box-Cox, each client sends four values to the server: the local sum c, sample size n, mean $\bar{y}$, and sum of squared deviations s for the transformed data (see line 5 in client part). The server aggregates these $(n, \bar{y}, s)$ triplets using the queue-based procedure in Algorithm 1 (line 4 in server part), which ensures numerical stability compared to naive sequential aggregation.

We also incorporate the numerically stable strategies from Section 4, namely log-domain computation and

Algorithm 2 Federated NLL Computation (Box-Cox)

Client Part:
1: **Input:** λ and local data x_i (size n)
2: $c \leftarrow \sum_i \ln x_i$
3: $y_i \leftarrow \begin{cases} x_i^\lambda & \text{if } \lambda \neq 0, \\ \ln x_i & \text{if } \lambda = 0. \end{cases}$
4: $\bar{y} \leftarrow \frac{1}{n}\sum_i y_i$ and $s \leftarrow \sum_i (y_i - \bar{y})^2$
5: **Send:** $(c, n, \bar{y}, s)$ to server

Server Part:
1: **Input:** λ
2: Collect $(c^{(j)}, n^{(j)}, \bar{y}^{(j)}, s^{(j)})$ from clients
3: Enqueue $(n^{(j)}, \bar{y}^{(j)}, s^{(j)})$ into Q
4: Compute $(N, \bar{Y}, S)$ from Q using Algorithm 1
5: **if** $\lambda \neq 0$ **then**
6: $\ln S \leftarrow \ln S - 2\ln|\lambda|$
7: **end if**
8: $\ln \sigma_\psi^2 \leftarrow \ln S - \ln N$
9: $\mathrm{NLL_{BC}} \leftarrow (1-\lambda)\sum_j c^{(j)} + \frac{N}{2}\ln\sigma_\psi^2$
10: **return** $\mathrm{NLL_{BC}}$

modified variance formulations (see line 3 in client part and line 6 in server part). The case of Yeo-Johnson is more involved, since it handles both positive and negative inputs; details are deferred to Appendix D.

6 EMPIRICAL EVALUATION

Our experiments contain two parts: (1) evaluating the effect of power transforms on downstream tasks, and (2) testing the numerical stability of our methods. Code is available at `https://github.com/xuefeng-xu/powertf`.

6.1 Downstream Effectiveness

We first evaluate the impact of power transforms on downstream classification tasks using three datasets: Adult (Becker and Kohavi, 1996), Bank (Moro et al., 2012), and Credit (Yeh, 2009). Dataset statistics are shown in Table 3. Each dataset is split into 80% training and 20% testing. Since features may contain negative values, we apply the Yeo-Johnson transformation to each feature, estimating λ^* from the training set. We compare against two baselines: (1) Standardization (STD: zero mean and unit variance), and (2) Raw data without any transformation.

We then train three classifiers on the transformed features: Linear Discriminant Analysis (LDA), Logistic Regression (LR), and XGBoost (XGB). Since LDA assumes normally distributed inputs, power transforms are expected to improve performance. Figure 7 shows

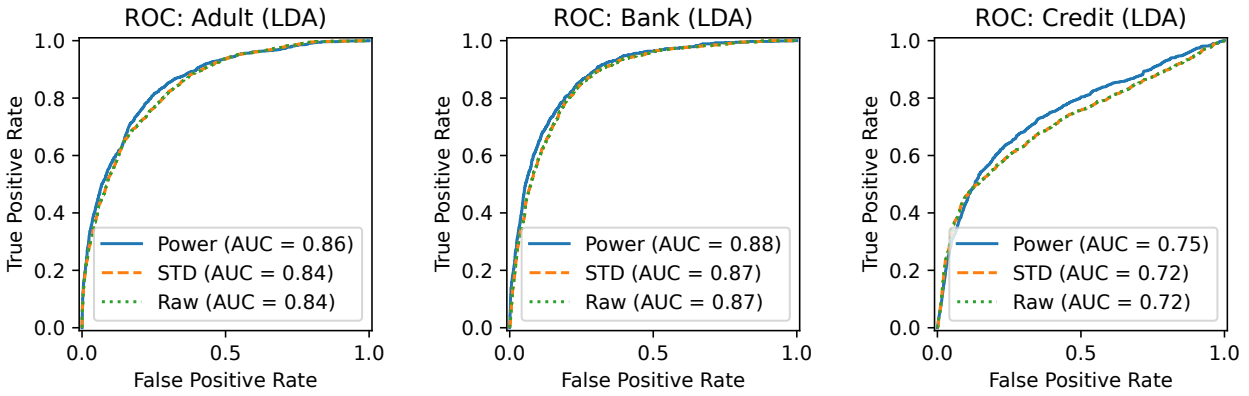


Figure 7: ROC curves after applying different transforms (LDA).

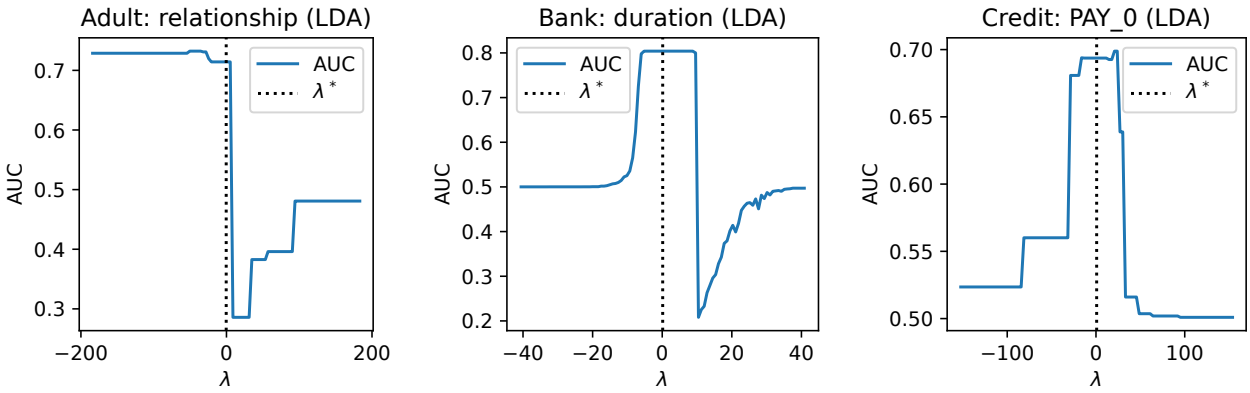


Figure 8: Effect of varying λ on AUC scores (LDA).

ROC curves and AUC scores on the test set (additional plots are in Figure 11 in the Appendix). Power transforms consistently outperform the baselines, although the improvements are modest. Notably, for XGBoost, the performance differential between using raw and preprocessed data is negligible. A recent study by Eftekhari and Papyan (2025) applied power transforms to hidden layers of deep neural networks, showing that improved Gaussianity leads to higher classification accuracy. This further confirms the effectiveness of power transforms.

We next study the effect of varying λ on AUC scores. This tests whether the estimated λ^* is indeed optimal for downstream tasks. Since each dataset has multiple features, we select the feature with the highest mutual information with the label and ignore other features and apply the Yeo-Johnson transform with varying λ. Figures 8 shows AUC as a function of λ (additional results are in Figure 12 in the Appendix).

Interestingly, λ^* (marked with a vertical dashed line) does not always yield the absolute highest AUC (e.g., Adult dataset). However, it consistently provides competitive results across datasets, while deviating from it often reduces performance. This highlights the importance of numerically stable estimation of λ^*, as instability can significantly degrade downstream outcomes.

6.2 Numerical Experiments

We now test numerical stability of power transforms on four datasets: Blood (Yeh, 2008), Cancer (Wolberg et al., 1995), Ecoli (Nakai, 1996), and House (Montoya and DataCanary, 2016). The first three were previously identified by Marchand et al. (2022) as unstable cases; House is a new addition with features exhibiting extreme skewness. We benchmark three methods: (1) ExpSearch, a derivative-based optimization method (Marchand et al., 2022) vs. our derivative-free approach (based on Brent's method); (2) Linear-domain computation with our proposed log-domain method; and (3) Naive one-pass variance computation vs. our pairwise computation in federated learning.

Table 2 summarizes dataset statistics and nine fea-

Table 2: Dataset statistics and unstable features detected.

Datasets	# Row	# Col	# Inst.	Transf.	ExpSearch	Linear
Blood	748	4	1	BC	—	—
				YJ	—	Monetary
Cancer	569	31	3	BC	—	—
				YJ	—	ID, area1, area3
Ecoli	336	8	2	BC	—	—
				YJ	lip, chg	lip, chg
House	1460	80	3	BC	YearRemodAdd, YrSold	YrSold
				YJ	Street, YearRemodAdd, YrSold	YrSold

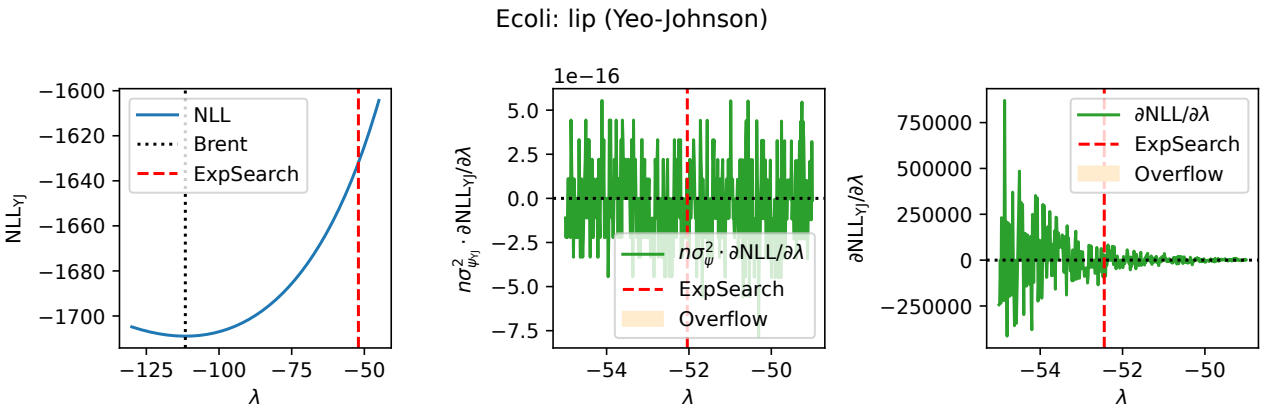


Figure 9: Comparison of Brent's method and ExpSearch.

tures we identified with numerical issues. Histograms in Figure 13 (in the Appendix) show these features are highly skewed, sometimes binary, and often extremely imbalanced. These match our analysis in Section 3, and align with our recipe for constructing adversarial datasets (Table 1). Importantly, this shows that such pathological data naturally occur in practice, underscoring the need for stability in power transforms.

Next, we observe that seven features cause ExpSearch to fail. Instead of finding λ^*, it returns either a boundary of the search interval or an arbitrary value. Figures 9 and 14 (in the Appendix) illustrate this. The left plot shows the true NLL curve and the optimum λ^* (found by Brent's method). ExpSearch's λ is far from optimal. The middle and right plots show the modified derivative used by ExpSearch versus the true derivative. Both are either unstable (returning arbitrary values) or overflow (returning boundary solutions). This explains ExpSearch's failures.

We then compare linear-domain versus log-domain computation. Eight features exhibit failures in the linear domain. Figure 15 (in the Appendix) shows that linear-domain NLL curves often overflow, preventing discovery of the optimum if λ^* lies in the overflow region. By contrast, log-domain computation yields smooth and stable curves, reliably identifying λ^*. This confirms the robustness of our approach.

Finally, we evaluate federated NLL computation. We split each dataset into 100 clients and compare our variance aggregation method with naive one-pass variance computation. Figures 10 and 16 (in the Appendix) show that our method produces smooth NLL curves, while the naive method introduces spikes that can disrupt optimization. This highlights the necessity of numerically stable aggregation in federated settings.

7 DISCUSSION

Runtime performance is an important consideration for practical deployment. Our stable recipe in Section 4 modifies the NLL computation and is indeed slower than the naive implementation due to the use of log-domain variance computation, as shown in Table 4 (Appendix). However, for scientific software, numerical correctness must take priority over raw speed. Moreover, the runtime remains well below human-perceptible latency in all cases, making it

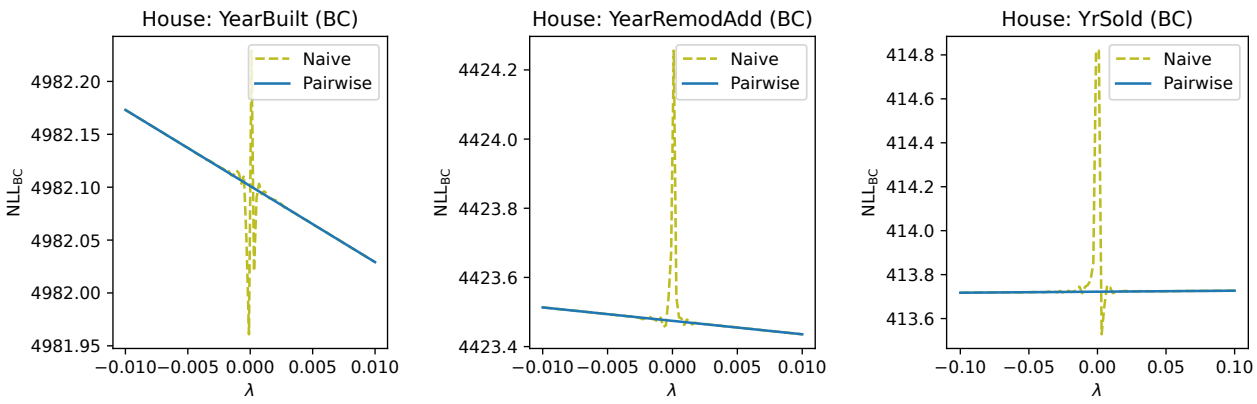


Figure 10: Comparison of pairwise and naive variance aggregation.

acceptable for most practical applications. In federated settings, our current implementation of variance aggregation (Algorithm 1) processes clients sequentially. This can be further optimized by parallelizing the pairwise aggregation steps, thereby reducing overall latency without affecting numerical stability.

Communication cost is a common concern in federated learning, consisting of two components: (1) the size of each message per round and (2) the total number of communication rounds. In our method, each client only needs to send four numbers to the server, while the server sends back a single number, making the per-round communication negligible. For the number of rounds, we adopt Brent's method, which converges superlinearly. In practice, convergence typically requires 20-30 rounds, depending on the dataset, which is acceptable in most settings. One possible extension is to reduce the number of rounds by increasing message size, since the server can evaluate the NLL at multiple points per round. For example, after identifying a bracket that contains the minimum, a grid search could be applied. As shown in Figure 17 (in the Appendix), communication rounds can be reduced to under 10 with a grid size of 1K. This trade-off between message size and number of rounds can be tuned for specific applications.

Privacy is a key concern in federated learning. Approaches such as Secure Aggregation (Bonawitz et al., 2017), Trusted Execution Environments (TEE) (Sabt et al., 2015), and Secure Multiparty Computation (SMPC) (Lindell, 2020) can ensure that only aggregated statistics are revealed to the server, not individual client data. Our method uses pairwise variance aggregation for numerical stability. To integrate with SMPC, the division operations in Equations (7) and (8) can be handled by treating the sample count n as a public value, which is standard in federated learning and also adopted by Marchand et al. (2022). Preserving privacy during iterative optimization without revealing intermediate results in each round is a stronger requirement and we leave it for future work.

Client sampling is a common technique in federated learning to handle offline or dropout clients. When a random subset of clients is selected each round, the NLL estimate becomes biased. However, normalizing the NLL by the subsample size recovers unbiasedness without affecting λ optimization. Specifically, the Box-Cox NLL $(1-\lambda)\sum_i \ln x_i + \frac{n}{2}\ln\sigma^2_{\psi_{\mathrm{BC}}}$ divided by n yields unbiased estimates of both the log-mean term and log-variance term under uniform random sampling. Thus, the normalized NLL can be reliably estimated even with random client subsampling.

8 CONCLUSION

Numerical issues have long been a challenge in scientific computing, as mathematically equivalent expressions can yield vastly different results under finite-precision arithmetic. In this paper, we addressed the numerical instability of power transforms, a widely used technique for data normalization. We conducted a detailed analysis of the sources of instability and proposed numerically stable approaches that combines log-domain computation, reformulated expressions, and bounding strategies. We further extended power transforms to the federated learning setting, introducing a numerically stable variance aggregation method suitable for distributed data. Our empirical results demonstrate the effectiveness and robustness of our methods in both centralized and federated scenarios. We believe that our work not only makes power transforms more reliable in practice, but also provides insights that can be applied to a broader range of numerical stability problems in scientific computing.

Acknowledgments

We thank Matt Haberland for helpful discussions and feedback on our implementation in the GitHub pull request to *SciPy*. We also thank the anonymous reviewers for their valuable comments. XX is supported by a scholarship from the Department of Computer Science, University of Warwick. GC is supported in part by EPSRC grant EP/V044621/1.

References

Barron, J. T. (2025). A power transform. *CoRR*, abs/2502.10647.

Becker, B. and Kohavi, R. (1996). Adult. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5XW20.

Bonawitz, K. A., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., and Seth, K. (2017). Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1175–1191. ACM.

Box, G. E. P. and Cox, D. R. (1964). An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, 26(2):211–243.

Brent, R. P. (2013). *Algorithms for Minimization Without Derivatives*. Dover Publications, Incorporated.

Chan, T. F., Golub, G. H., and LeVeque, R. J. (1982). Updating formulae and a pairwise algorithm for computing sample variances. In *COMPSTAT 1982 5th Symposium held at Toulouse 1982*, pages 30–41, Heidelberg. Physica-Verlag HD.

Chan, T. F., Golub, G. H., and Leveque, R. J. (1983). Algorithms for computing the sample variance: Analysis and recommendations. *The American Statistician*, 37(3):242–247.

Corless, R. M., Gonnet, G. H., Hare, D. E. G., Jeffrey, D. J., and Knuth, D. E. (1996). On the lambertw function. *Advances in Computational Mathematics*, 5(1):329–359.

Eftekhari, D. and Papyan, V. (2025). On the importance of gaussianizing representations. In *Forty-second International Conference on Machine Learning*.

Fink, E. L. (2009). The faqs on data transformation. *Communication Monographs*, 76(4):379–397.

Haberland, M. (2023). Bug: fix overflow in stats.yeojohnson. `https://github.com/scipy/scipy/pull/18852#issuecomment-1657858886`. Accessed: August 14, 2025.

IEEE (2019). Ieee standard for floating-point arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Nitin Bhagoji, A., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., D'Oliveira, R. G. L., Eichner, H., El Rouayheb, S., Evans, D., Gardner, J., Garrett, Z., Gascón, A., Ghazi, B., Gibbons, P. B., Gruteser, M., Harchaoui, Z., He, C., He, L., Huo, Z., Hutchinson, B., Hsu, J., Jaggi, M., Javidi, T., Joshi, G., Khodak, M., Konecný, J., Korolova, A., Koushanfar, F., Koyejo, S., Lepoint, T., Liu, Y., Mittal, P., Mohri, M., Nock, R., Özgür, A., Pagh, R., Qi, H., Ramage, D., Raskar, R., Raykova, M., Song, D., Song, W., Stich, S. U., Sun, Z., Suresh, A. T., Tramèr, F., Vepakomma, P., Wang, J., Xiong, L., Xu, Z., Yang, Q., Yu, F. X., Yu, H., and Zhao, S. (2021). Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210.

Kouider, E. and Chen, H. (1995). Concavity of box-cox log-likelihood function. *Statistics & Probability Letters*, 25(2):171–175.

Lindell, Y. (2020). Secure multiparty computation. *Communications of the ACM*, 64(1):86–96.

Ling, R. F. (1974). Comparison of several algorithms for computing sample means and variances. *Journal of the American Statistical Association*, 69(348):859–866.

Marchand, T., Muzellec, B., Béguier, C., Ogier du Terrail, J., and Andreux, M. (2022). Securefedyj: a safe feature gaussianization protocol for federated learning. In *Advances in Neural Information Processing Systems*, volume 35, pages 36585–36598. Curran Associates, Inc.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and Arcas, B. A. y. (2017). Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR.

Montoya, A. and DataCanary (2016). House prices - advanced regression techniques. `https://kaggle.com/competitions/house-prices-advanced-regression-techniques`. Kaggle.

Moro, S., Rita, P., and Cortez, P. (2012). Bank Marketing. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5K306.

Nakai, K. (1996). Ecoli. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5388M.

Ruppert, D. (2001). Statistical analysis, special problems of: Transformations of data. In *International Encyclopedia of the Social & Behavioral Sciences*, pages 15007–15014. Pergamon, Oxford.

Sabt, M., Achemlal, M., and Bouabdallah, A. (2015). Trusted execution environment: What it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 57–64.

Venables, W. N. and Ripley, B. D. (2002). *Modern Applied Statistics with S*. Springer, New York, fourth edition. ISBN 0-387-95457-0.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17(3):261–272.

Weckesser, W. (2019). Bug: stats: Fix boxcox_llf to avoid loss of precision. `https://github.com/scipy/scipy/pull/10072`. Accessed: August 14, 2025.

Wolberg, W., Mangasarian, O., Street, N., and Street, W. (1995). Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5DW2B.

Yeh, I.-C. (2008). Blood Transfusion Service Center. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5GS39.

Yeh, I.-C. (2009). Default of Credit Card Clients. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C55S3H.

Yeo, I. and Johnson, R. A. (2000). A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959.

Yeo, I.-K. (1997). *A new family of power transformations to reduce skewness or approximate normality*. PhD thesis, The University of Wisconsin-Madison.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Not Applicable]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

A PROOF OF THEOREM 3.1

1. For $x \geq 1$, we have:
$$\begin{cases} x^\lambda - 1 \geq 0 & \text{if } \lambda > 0, \\ x^\lambda - 1 \leq 0 & \text{if } \lambda < 0. \end{cases} \tag{17}$$
When $\lambda = 0$, $\ln x \geq 0$ for $x \geq 1$. Hence $\psi(\lambda, x) \geq 0$ for all λ whenever $x \geq 1$. Similarly, for $0 < x < 1$, we have:
$$\begin{cases} x^\lambda - 1 < 0 & \text{if } \lambda > 0, \\ x^\lambda - 1 > 0 & \text{if } \lambda < 0. \end{cases} \tag{18}$$
When $\lambda = 0$, $\ln x < 0$ for $0 < x < 1$. Hence $\psi(\lambda, x) < 0$ for all λ whenever $0 < x < 1$.

2. The second order partial derivative of ψ with respect to x is:
$$\frac{\partial^2 \psi(\lambda, x)}{\partial x^2} = \begin{cases} (\lambda - 1)x^{\lambda - 2} & \text{if } \lambda \neq 0, \\ -1/x^2 & \text{if } \lambda = 0. \end{cases} \tag{19}$$
Therefore, $\frac{\partial^2 \psi(\lambda, x)}{\partial x^2} > 0$ when $\lambda > 1$ and $\frac{\partial^2 \psi(\lambda, x)}{\partial x^2} < 0$ when $\lambda < 1$.

3. It's clear that $\psi(\lambda, x)$ is continuous for λ and x except $\lambda = 0$. We just need to prove it's continuous at $\lambda = 0$. By L'Hopital's rule, we have:
$$\lim_{\lambda \to 0} \frac{x^\lambda - 1}{\lambda} = \lim_{\lambda \to 0} \frac{x^\lambda \ln x}{1} = \ln x. \tag{20}$$

4. We prove this by induction. Let $k = 1$, then for $\lambda \neq 0$:
$$\psi^{(1)}(\lambda, x) = \frac{x^\lambda \lambda \ln x - (x^\lambda - 1)}{\lambda^2} = \frac{x^\lambda \ln x - \psi^{(0)}(\lambda, x)}{\lambda}. \tag{21}$$
Assume that this hold for $k = n$ where $n \geq 1$, then for $k = n + 1$ and $\lambda \neq 0$:
$$\psi^{(n+1)}(\lambda, x) = \frac{\partial}{\partial \lambda} \frac{x^\lambda (\ln x)^n - n\psi^{(n-1)}(\lambda, x)}{\lambda} \tag{22}$$
$$= \frac{[x^\lambda (\ln x)^{n+1} - n\psi^{(n)}(\lambda, x)]\lambda - [x^\lambda (\ln x)^n - n\psi^{(n-1)}(\lambda, x)]}{\lambda^2} \tag{23}$$
$$= \frac{x^\lambda (\ln x)^{n+1} - (n+1)\psi^{(n)}(\lambda, x)}{\lambda}. \tag{24}$$
For $\lambda = 0$, recall the Taylor expansion of x^λ at $\lambda = 0$ is:
$$x^\lambda = 1 + \lambda \ln x + \frac{(\lambda \ln x)^2}{2!} + \frac{(\lambda \ln x)^3}{3!} + \cdots = \sum_{m=0}^{\infty} \frac{(\lambda \ln x)^m}{m!}. \tag{25}$$
Thus,
$$\psi(\lambda, x) = \frac{x^\lambda - 1}{\lambda} = \ln x + \frac{\lambda (\ln x)^2}{2!} + \frac{\lambda^2 (\ln x)^3}{3!} + \cdots = \sum_{m=1}^{\infty} \frac{\lambda^{m-1} (\ln x)^m}{m!}. \tag{26}$$
Therefore, the k-th order partial derivative of ψ with respect to λ is:
$$\frac{\partial^k \psi(\lambda, x)}{\partial \lambda^k} = \sum_{m=k+1}^{\infty} \frac{(m-1)! \lambda^{m-k-1} (\ln x)^m}{m!} = \sum_{m=k+1}^{\infty} \frac{\lambda^{m-k-1} (\ln x)^m}{m}. \tag{27}$$
Finally, at $\lambda = 0$, we have:
$$\left. \frac{\partial^k \psi(\lambda, x)}{\partial \lambda^k} \right|_{\lambda = 0} = \frac{(\ln x)^{k+1}}{k+1}. \tag{28}$$
This completes the induction.

5. The partial derivative of ψ with respect to x is

$$\frac{\partial \psi(\lambda, x)}{\partial x} = \begin{cases} x^{\lambda-1} & \text{if } \lambda \neq 0, \\ 1/x & \text{if } \lambda = 0. \end{cases} \tag{29}$$

so $\frac{\partial \psi(\lambda,x)}{\partial x} > 0$. Therefore, ψ is increasing in x.

The partial derivative of ψ with respect to λ is

$$\frac{\partial \psi(\lambda, x)}{\partial \lambda} = \begin{cases} \frac{x^\lambda(\ln x^\lambda - 1)+1}{\lambda^2} & \text{if } \lambda \neq 0, \\ (\ln x)^2/2 & \text{if } \lambda = 0. \end{cases} \tag{30}$$

Let $y = x^\lambda > 0$ and $f_1(y) = y(\ln y - 1) + 1$, we have $f_1'(y) = \ln y$, $f_1''(y) = 1/y > 0$. Thus $f_1(y)$ has the unique minimum at $y = 1$ and $f_1(y) > f_1(1) = 0$. Thus $\frac{\partial \psi(\lambda,x)}{\partial \lambda} > 0$. Therefore, ψ is increasing in λ.

6. The second order partial derivative of ψ with respect to λ is

$$\frac{\partial^2 \psi(\lambda, x)}{\partial \lambda^2} = \begin{cases} \frac{x^\lambda[(\ln x^\lambda)^2 - 2\ln x^\lambda + 2] - 2}{\lambda^3} & \text{if } \lambda \neq 0, \\ (\ln x)^3/3 & \text{if } \lambda = 0. \end{cases} \tag{31}$$

Let $y = x^\lambda > 0$ and $f_2(y) = y[(\ln y)^2 - 2\ln y + 2] - 2$, we have $f_2'(y) = (\ln y)^2 > 0$ and $f_2(1) = 0$. Thus $f_2(y) > 0$ when $y > 1$ and $f_2(y) < 0$ when $y < 1$ since $f_2(y)$ is increasing in y.

The relationship between x, λ and $y, f_2(y)$ are as follows

$$\left.\begin{aligned} x > 1, \lambda > 0 &\Rightarrow y > 1, f_2(y) > 0 \\ x > 1, \lambda < 0 &\Rightarrow y < 1, f_2(y) < 0 \end{aligned}\right\} \Rightarrow f_2(y)/\lambda^3 > 0$$
$$\left.\begin{aligned} 0 < x < 1, \lambda < 0 &\Rightarrow y > 1, f_2(y) > 0 \\ 0 < x < 1, \lambda > 0 &\Rightarrow y < 1, f_2(y) < 0 \end{aligned}\right\} \Rightarrow f_2(y)/\lambda^3 < 0 \tag{32}$$

Therefore, $\frac{\partial^2 \psi(\lambda,x)}{\partial \lambda^2} > 0$ when $x > 1$ and $\frac{\partial^2 \psi(\lambda,x)}{\partial \lambda^2} < 0$ when $0 < x < 1$.

B LOG-DOMAIN COMPUTATION

Log-domain computation refers to performing numerical operations in the logarithmic space rather than on raw values. This technique is particularly effective at avoiding numerical overflow and underflow, especially when working with exponential functions, as in the case of power transforms. Central to this approach is the *Log-Sum-Exp* (LSE) function:

$$\text{LSE}(x_1, \ldots, x_n) = \ln \sum_{i=1}^n \exp(x_i) = \ln \sum_{i=1}^n \exp(x_i - c) + c \tag{33}$$

where $c = \max_i x_i$. This ensures numerically stable computation by shifting values before exponentiation.

Using the LSE operator, the logarithmic mean is computed as

$$\ln \mu = \ln \left(\frac{1}{n} \sum_{i=1}^n x_i \right) = \text{LSE}(\ln x_1, \ldots, \ln x_n) - \ln n \tag{34}$$

Similarly, the logarithmic variance is expressed as

$$\ln \sigma^2 = \ln \sum_{i=1}^n (x_i - \mu)^2 - \ln n = \text{LSE}\left(2\ln(x_1 - \mu), \ldots, 2\ln(x_n - \mu)\right) - \ln n \tag{35}$$

The computation of $\ln(x_i - \mu)$ requires extra care. A stable way is to rewrite the difference using the LSE trick (Haberland, 2023):

$$\ln(x_i - \mu) = \ln\left(\exp(\ln x_i) + \exp(\ln \mu + \pi i)\right) = \text{LSE}(\ln x_i, \ln \mu + \pi i) \tag{36}$$

where the πi term is the imaginary part that handles sign differences for negative values.

C NUMERICALLY STABLE YEO-JOHNSON

To extend numerical stability to the Yeo-Johnson transformation, we must handle both positive and negative inputs. Unlike Box-Cox, the constant term cannot be eliminated when both positive and negative values are present. We therefore consider the following two cases separately:

1. Data entirely positive (or entirely negative): Apply log-domain computation as in Box-Cox, removing the constant term and factoring out λ (for positive) or $(\lambda - 2)$ (for negative).
2. Data contains both positive and negative values: Use the full piecewise definition of Yeo-Johnson (Equation (2)) with log-domain computation, but do not remove the constant term. In practice, this case does not exhibit instability.

As in the Box-Cox case, extreme values of λ can be constrained during optimization. Here, we use $x = 0$ as the reference point since $\psi_{\mathrm{YJ}}(\lambda, 0) = 0$:

$$\begin{aligned} \min_{\lambda} \quad & \mathrm{NLL}_{\mathrm{YJ}}(\lambda, x) \\ \text{s.t.} \quad & \lambda \le \psi_{\mathrm{YJ}}^{-1}(x_{\max}, y_B) \quad \text{if } x_{\max} > 0, \\ & \lambda \ge \psi_{\mathrm{YJ}}^{-1}(x_{\min}, -y_B) \text{ if } x_{\min} < 0. \end{aligned} \tag{37}$$

Here ψ_{YJ}^{-1} is the inverse Yeo-Johnson transformation, expressed using the Lambert W function:

$$\psi_{\mathrm{YJ}}^{-1}(x, y) = \begin{cases} -\frac{1}{y} - \frac{1}{\ln(x+1)} W\left(-\frac{(x+1)^{-1/y}\ln(x+1)}{y}\right) & \text{if } x \ge 0, \\ 2 - \frac{1}{y} + \frac{1}{\ln(1-x)} W\left(\frac{(1-x)^{1/y}\ln(1-x)}{y}\right) & \text{if } x < 0. \end{cases} \tag{38}$$

For overflow cases, we employ the $k = -1$ branch of the Lambert W function. As for $x > 0$ and $\lambda \gg 1$:

$$W\left(-\frac{(x+1)^{-1/y}\ln(x+1)}{y}\right) = -(\lambda + \tfrac{1}{y})\ln(x+1) \approx -\lambda \ln(x+1) \ll -1. \tag{39}$$

For $x < 0$ and $\lambda \ll 1$:

$$W\left(\frac{(1-x)^{1/y}\ln(1-x)}{y}\right) = (\lambda + \tfrac{1}{y} - 2)\ln(1-x) \approx (\lambda - 2)\ln(1-x) \ll -1. \tag{40}$$

D FEDERATED NLL COMPUTATION FOR YEO-JOHNSON

In the federated setting, the Yeo-Johnson transformation requires additional care due to its piecewise definition for positive and negative inputs. The main challenges are: (1) clients need to use different formulas to achieve numerical stability based on their local data (all-positive, all-negative, or mixed); (2) the server must correctly aggregate statistics from these heterogeneous clients.

First, instead of reporting only the total number of samples, each client separately transmits the counts of positive and negative values. This enables the server to identify whether the client's dataset is all-positive, all-negative, or mixed, and to aggregate contributions accordingly.

Second, depending on the case, clients apply different formulas for the transformed data. In the all-positive (or all-negative) case, constant term can be safely omitted. For mixed data, however, the full piecewise definition of the transform function must be used to preserve correctness of the variance.

Finally, the server aggregates the statistics and computes the variance of the transformed data. It then applies a correction step to adjust the mean, compensating for constant term that clients may have omitted. The complete procedure is summarized in Algorithm 3.

E SUPPLEMENTARY TABLES AND PLOTS

Algorithm 3 Federated NLL Computation (Yeo-Johnson)

Client Part:

Input: λ and local data x_i (size n, with positive size n^+ and negative size n^-)
$c \leftarrow \sum_i \operatorname{sgn}(x_i) \ln(|x_i| + 1)$
if $n^- = 0$ **then** ▷ All positive
    $y_i \leftarrow \begin{cases} (x_i + 1)^\lambda & \text{if } \lambda \neq 0, \\ \ln(x_i + 1) & \text{if } \lambda = 0. \end{cases}$
else if $n^+ = 0$ **then** ▷ All negative
    $y_i \leftarrow \begin{cases} (-x_i + 1)^{2-\lambda} & \text{if } \lambda \neq 2, \\ -\ln(-x_i + 1) & \text{if } \lambda = 2. \end{cases}$
else ▷ Mixed data
    $y_i \leftarrow \psi_{\mathrm{YJ}}(\lambda, x_i)$ (Equation (2))
end if
$\bar{y} \leftarrow \frac{1}{n} \sum_i y_i$ and $s \leftarrow \sum_i (y_i - \bar{y})^2$
Send: $(c, n^+, n^-, \bar{y}, s)$ to server

Server Part:

Input: λ
Collect $(c^{(j)}, n^{+(j)}, n^{-(j)}, \bar{y}^{(j)}, s^{(j)})$ from clients
if $n^{-(j)} = 0$ **then** ▷ All positive
    Enqueue $(n^{+(j)}, \bar{y}^{(j)}, s^{(j)})$ into Q^+
else if $n^{+(j)} = 0$ **then** ▷ All negative
    Enqueue $(n^{-(j)}, \bar{y}^{(j)}, s^{(j)})$ into Q^-
else ▷ Mixed data
    Enqueue $(n^{+(j)} + n^{-(j)}, \bar{y}^{(j)}, s^{(j)})$ into $Q^\pm$
end if
Compute $(N^+, \bar{Y}^+, S^+)$ from Q^+ using Algorithm 1
if $\lambda \neq 0$ **then**
    $\ln S^+ \leftarrow \ln S^+ - 2 \ln |\lambda|$
end if
Compute $(N^-, \bar{Y}^-, S^-)$ from Q^- using Algorithm 1
if $\lambda \neq 2$ **then**
    $\ln S^- \leftarrow \ln S^- - 2 \ln |2 - \lambda|$
end if
Compute $(N^\pm, \bar{Y}^\pm, S^\pm)$ from $Q^\pm$ using Algorithm 1
if ($N^- > 0$ or $N^\pm > 0$) and $\lambda \neq 0$ **then** ▷ Add constant term back for mixed data
    $\bar{Y}^+ \leftarrow (\bar{Y}^+ - 1)/\lambda$
else if ($N^+ > 0$ or $N^\pm > 0$) and $\lambda \neq 2$ **then**
    $\bar{Y}^- \leftarrow (\bar{Y}^- - 1)/(\lambda - 2)$
end if
Enqueue $(N^+, \bar{Y}^+, S^+)$, $(N^-, \bar{Y}^-, S^-)$, $(N^\pm, \bar{Y}^\pm, S^\pm)$ into Q
Compute $(N, \bar{Y}, S)$ from Q using Algorithm 1
$\ln \sigma_\psi^2 \leftarrow \ln S - \ln N$
$\mathrm{NLL}_{\mathrm{YJ}} \leftarrow (1 - \lambda) \sum_j c^{(j)} + \frac{N}{2} \ln \sigma_\psi^2$
return $\mathrm{NLL}_{\mathrm{YJ}}$

Table 3: Dataset statistics.

Datasets	# Row	# Col
Adult	33K	14
Bank	45K	16
Credit	30K	24

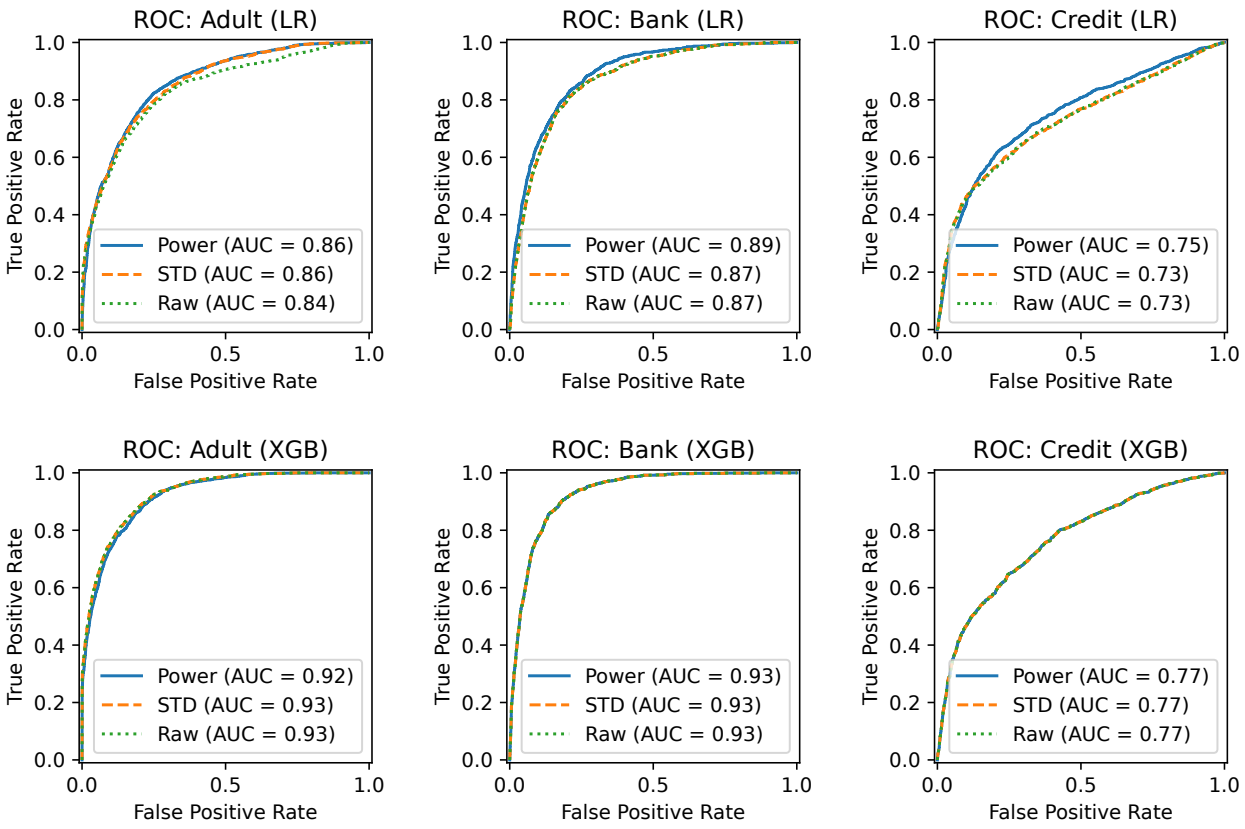


Figure 11: ROC curves after applying different transforms (LR and XGB).

Table 4: Runtime comparison between naive and our method for Box-Cox NLL computation.

Data size	Time (Naive)	Time (Ours)
1K	13.10 µs ± 391.62 ns	273.88 µs ± 819.93 ns
10K	51.70 µs ± 2.54 µs	1.70 ms ± 13.29 µs
100K	604.00 µs ± 10.06 µs	18.13 ms ± 183.97 µs
1M	6.57 ms ± 60.04 µs	194.75 ms ± 1.31 ms

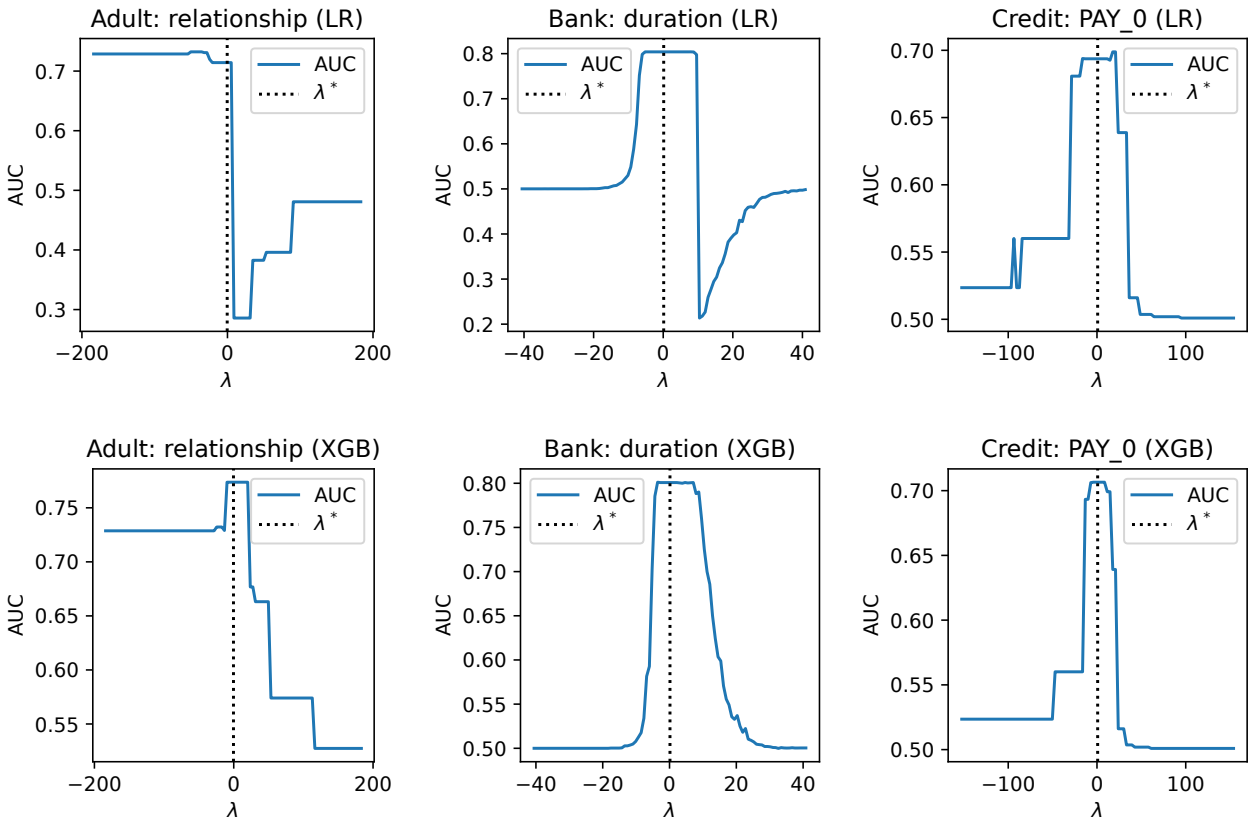


Figure 12: Effect of varying λ on AUC scores (LR and XGB).

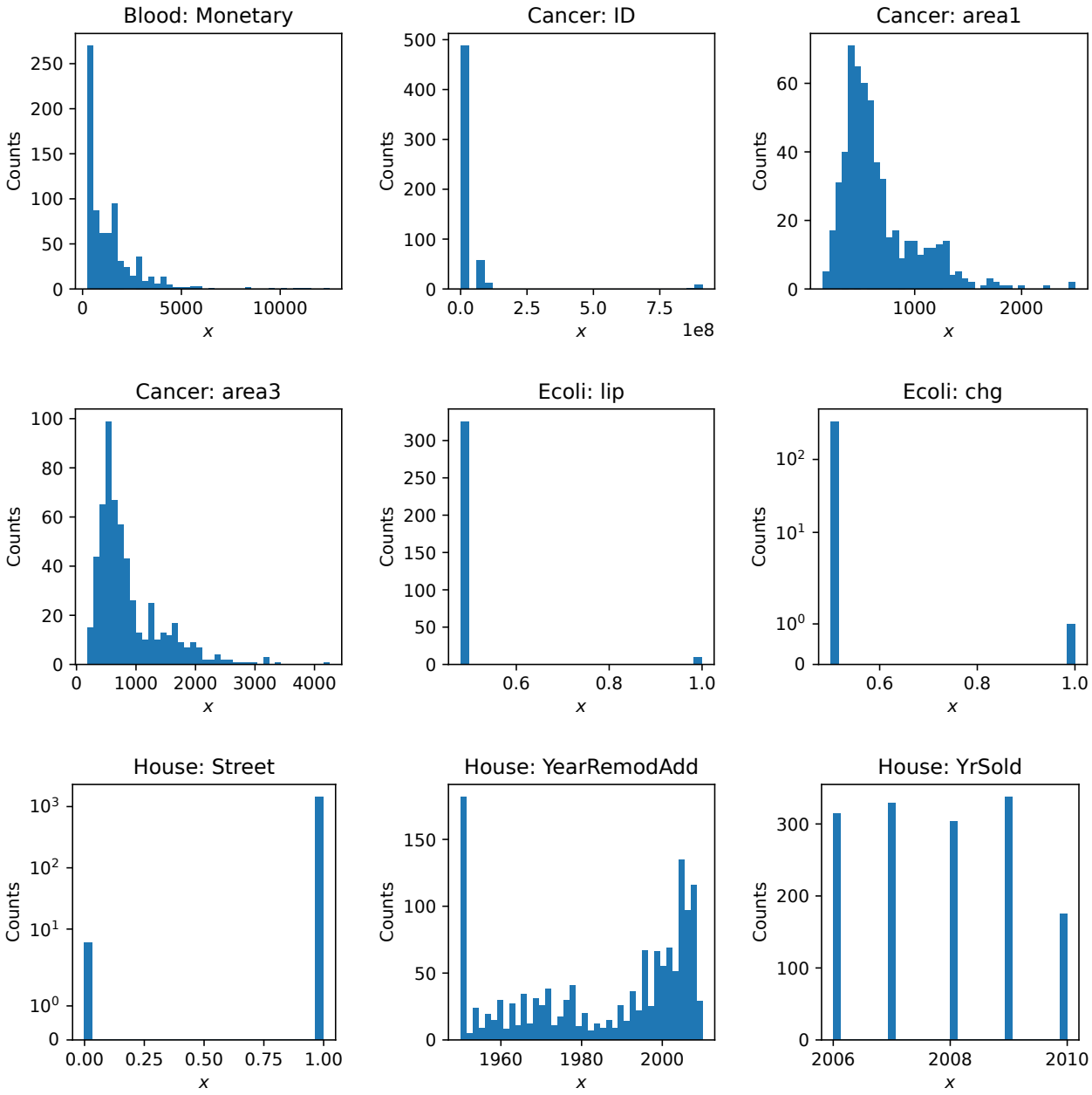


Figure 13: Histogram of features that have numerical issues.

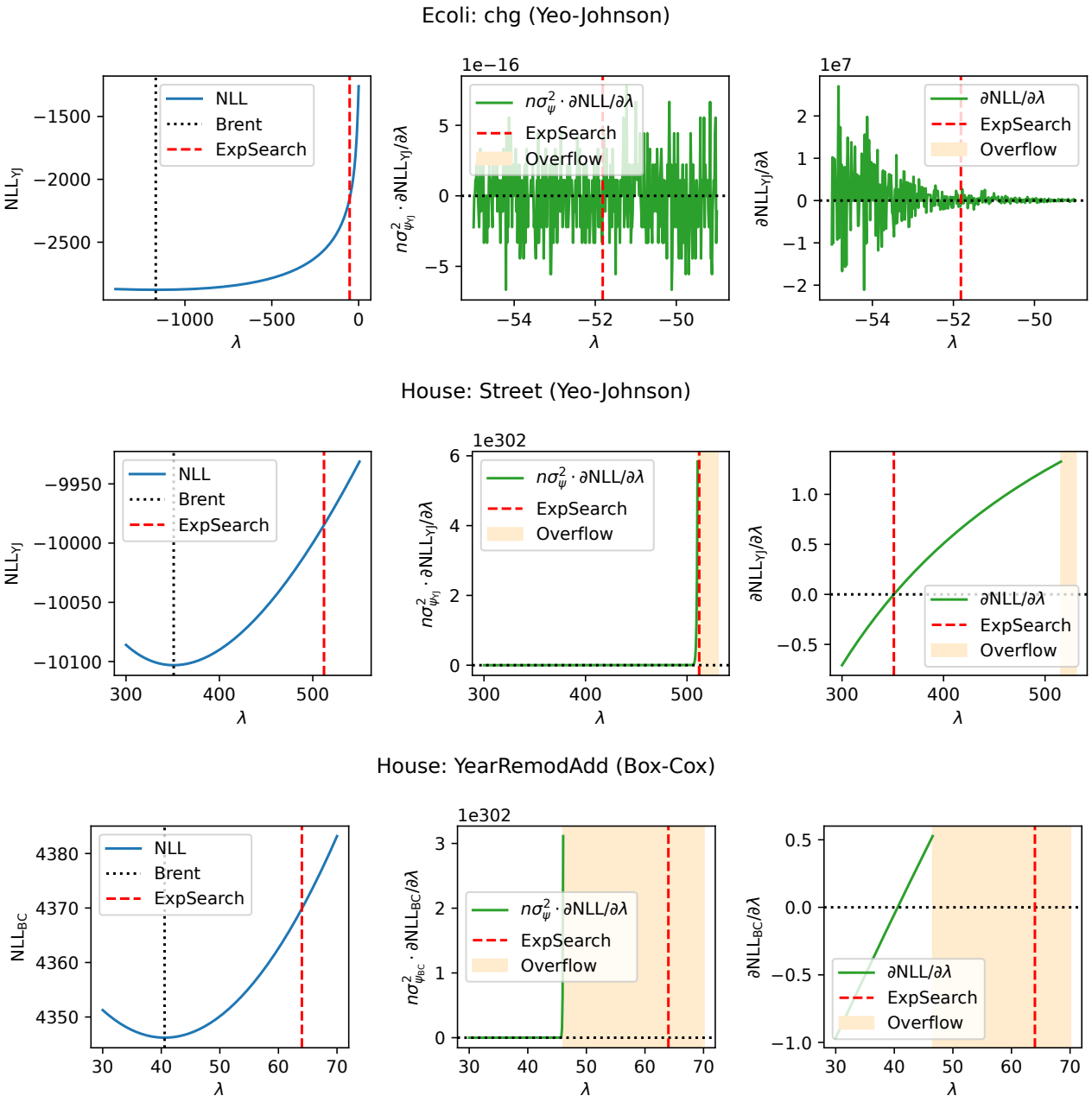


Figure 14: Comparison of Brent's method and ExpSearch.

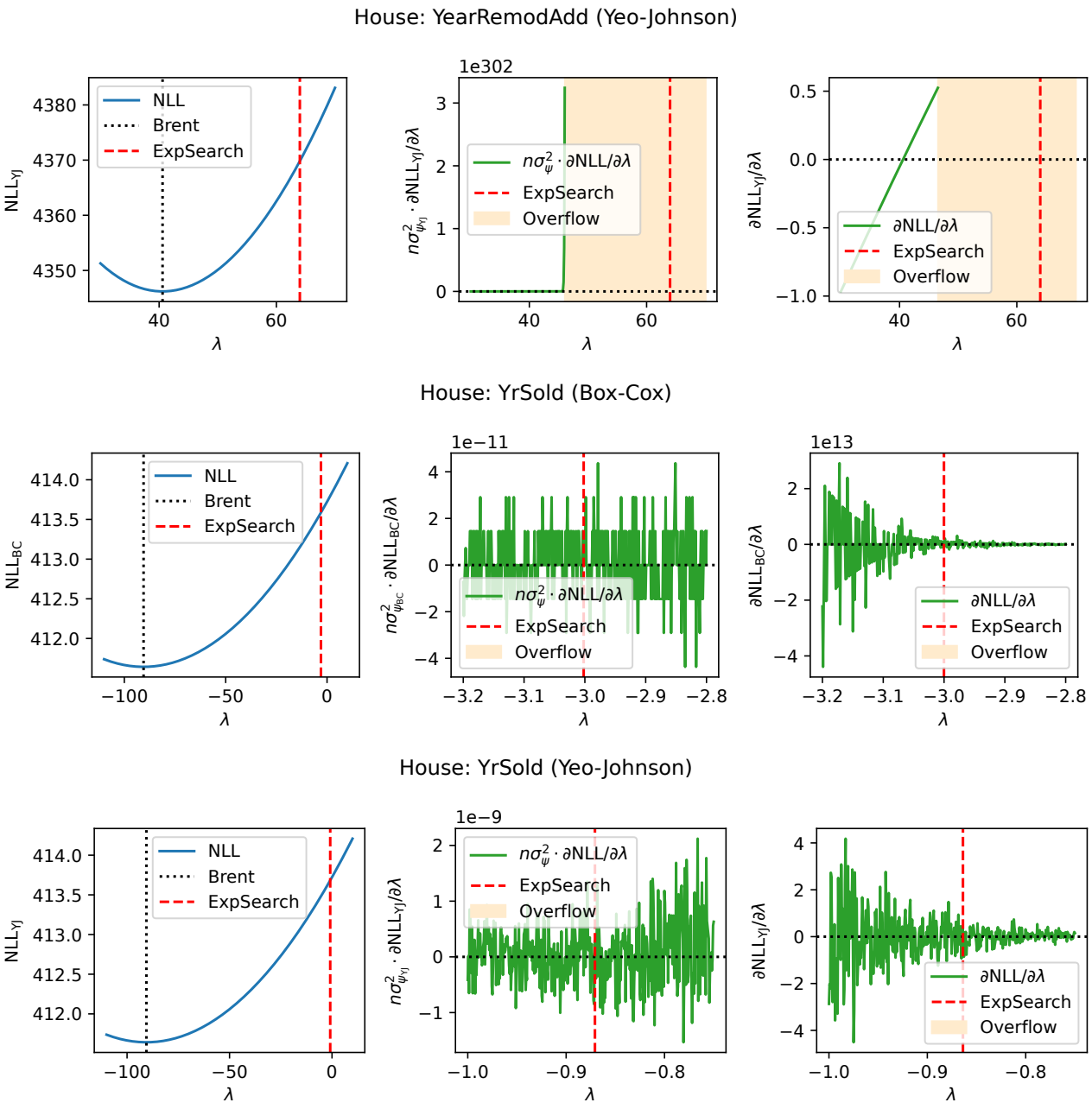


Figure 14: Comparison of Brent's method and ExpSearch.

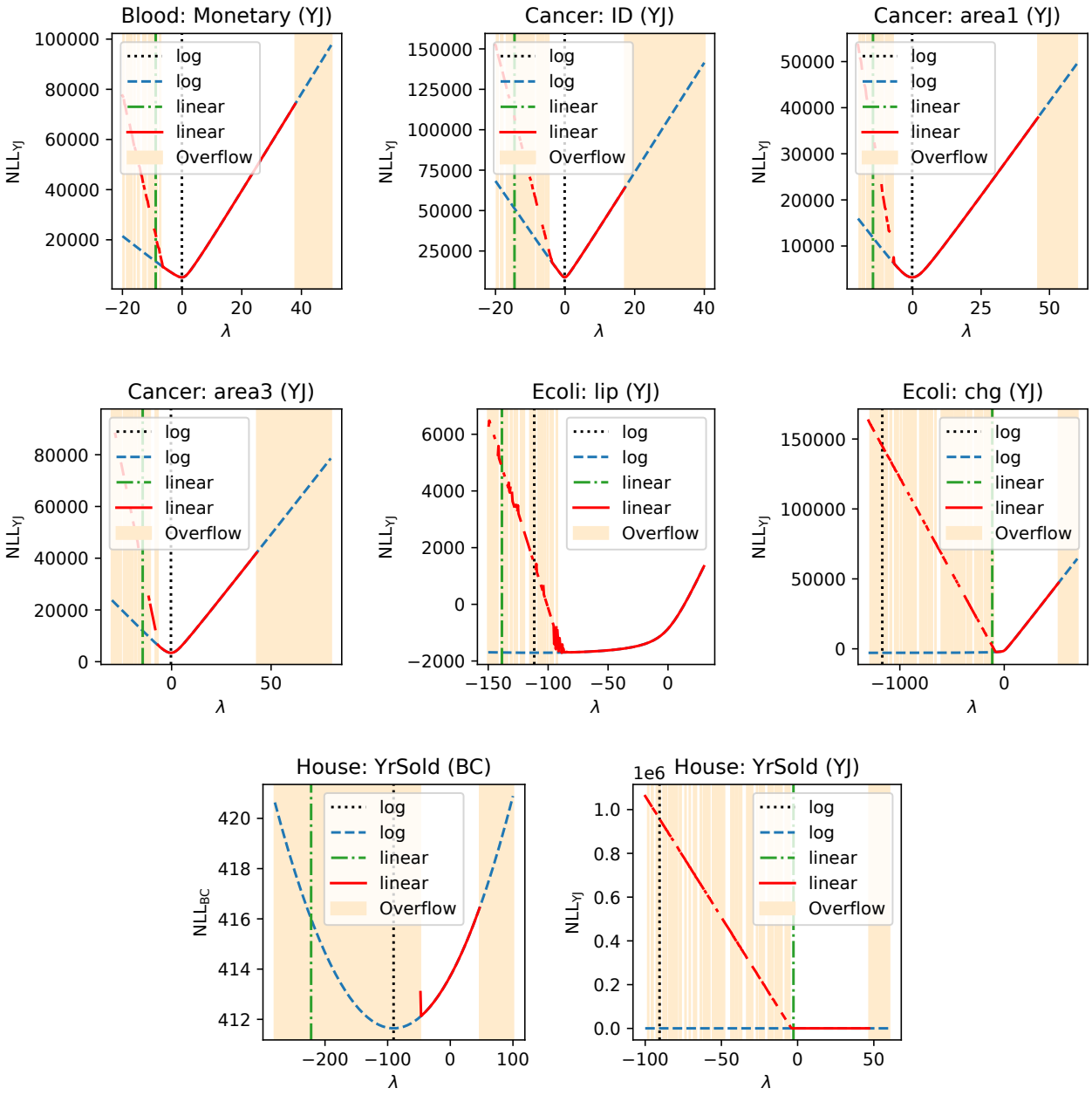


Figure 15: Comparison of log and linear domain computation.

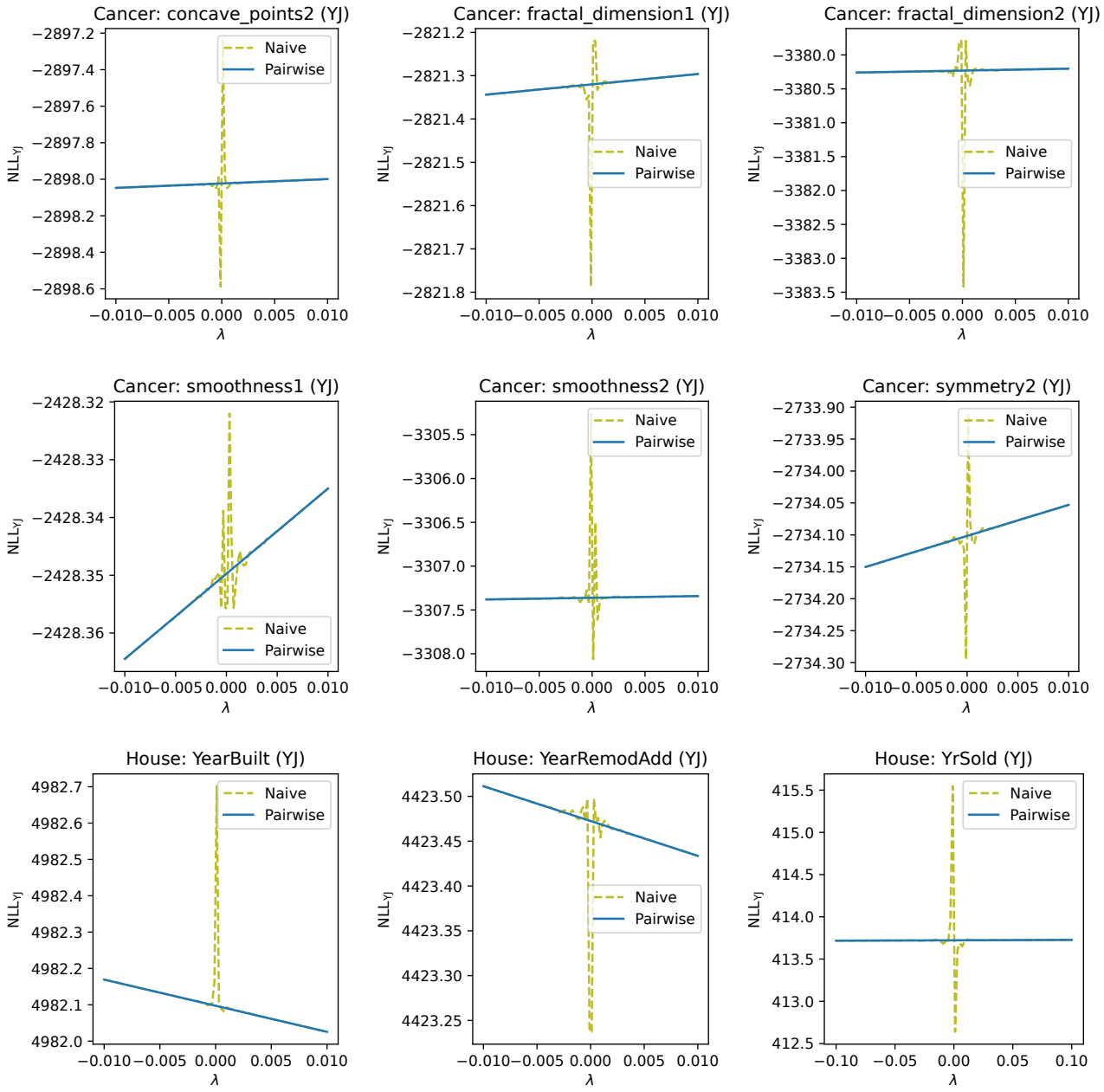


Figure 16: Comparison of pairwise and naive variance aggregation.

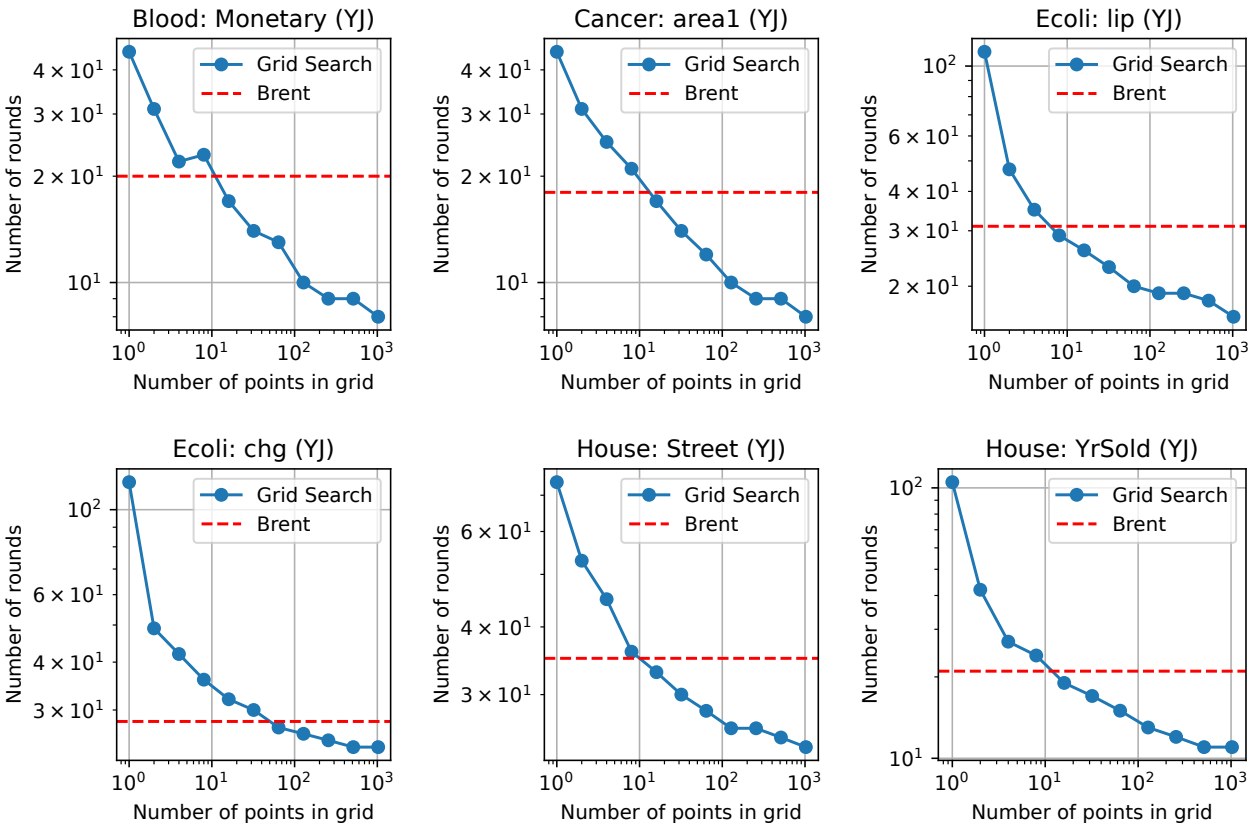


Figure 17: Number of communication rounds vs. grid size.