Power Transform Revisited: Numerically Stable, and Federated

TL;DR: Numerically stable power transforms with an extension to federated learning.

AISTATS 2026

Xuefeng Xu [1], Graham Cormode [2]

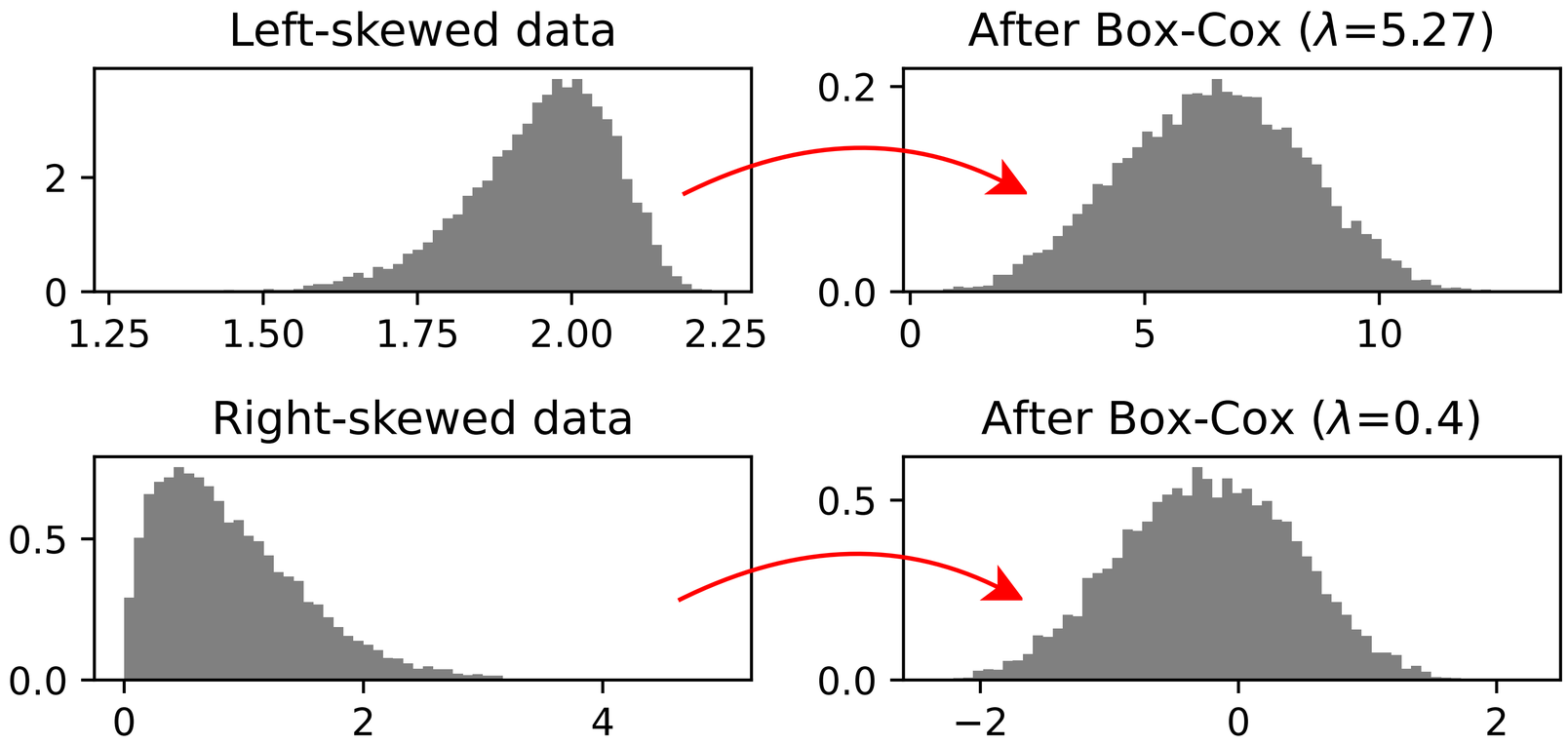
[1] University of Warwick, [2] University of Oxford

Contact: xuefeng.xu@warwick.ac.uk

https://xuefeng-xu.github.io/powertf.html

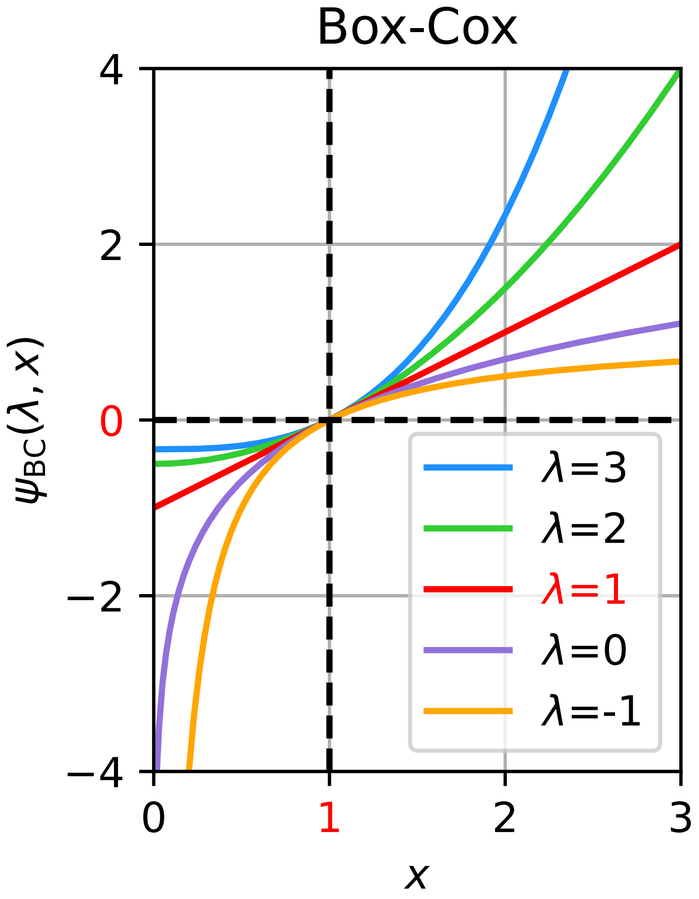
Power Transform

- A parametric method (λ) that transform data into Gaussian-like distribution



Box-Cox Transform

- Requires $x > 0$: $\psi(\lambda, x) = \begin{cases} (x^\lambda - 1)/\lambda & \text{if } \lambda \neq 0, \\ \ln x & \text{if } \lambda = 0. \end{cases}$
- Minimize Negative Log Likelihood: $\lambda^* = \min_\lambda \mathrm{NLL}(\lambda, x)$
- $\mathrm{NLL}(\lambda, x) = (1 - \lambda) \sum_i^n \ln x_i + \frac{n}{2} \ln \mathrm{Var}[\psi(\lambda, x)]$
- NLL is strictly convex —> guarantees a unique minimum
- Extension: Yeo-Johnson Transform handles $x < 0$



Box, G. E. P. and Cox, D. R. (1964). An analysis of transformations. JRSS B.

Kouider, E. and Chen, H. (1995). Concavity of boxcox log-likelihood function. Statistics & Probability Letters.

Why Revisit?

- A widely use statistical method
- Numerical instabilities persist
- Simple inputs trigger failures
- Cannot resolve by quick fixes
- Existing softwares are affected
- Prior work doesn't address it

```
import numpy as np
from scipy.special import boxcox
from scipy.optimize import minimize_scalar

def nll(lmb, x):
    logvar = np.log(np.var(boxcox(x, lmb)))
    return (1 - lmb) * np.sum(np.log(x)) \
            + len(x) / 2 * logvar

x = [10, 10, 10, 9.9]
lmb = minimize_scalar(nll, args=(x,))
> RuntimeWarning: overflow encountered ...
```

Numerical Instabilities

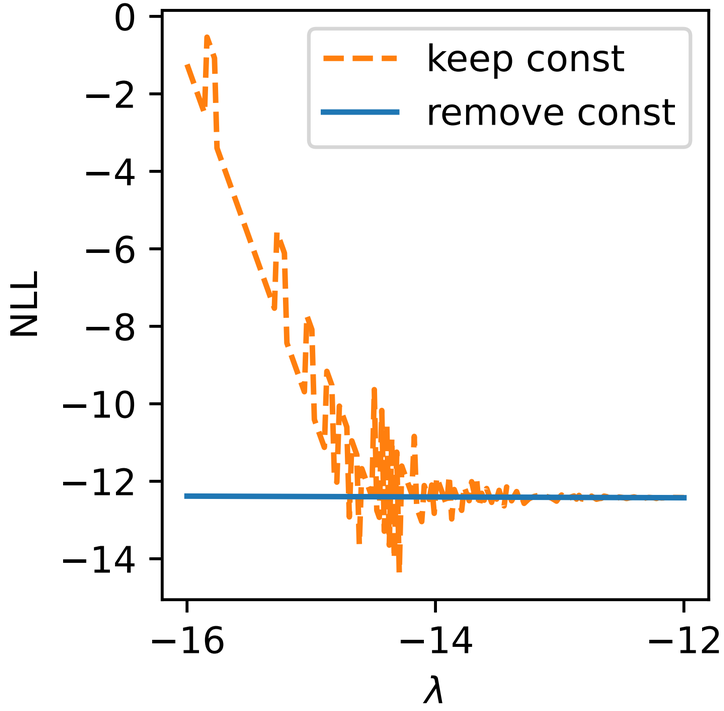
- **Challenge 1:** Catastrophic Cancellation
  - Loss of precision when subtracting floating-point values
- **Challenge 2:** Numerical Overflow
  - Floating-point representations have bounded range
- **Challenge 3:** Federated Aggregation
  - Instability in distributed variance computation
- **Our contribution:** Addressing all three issues!

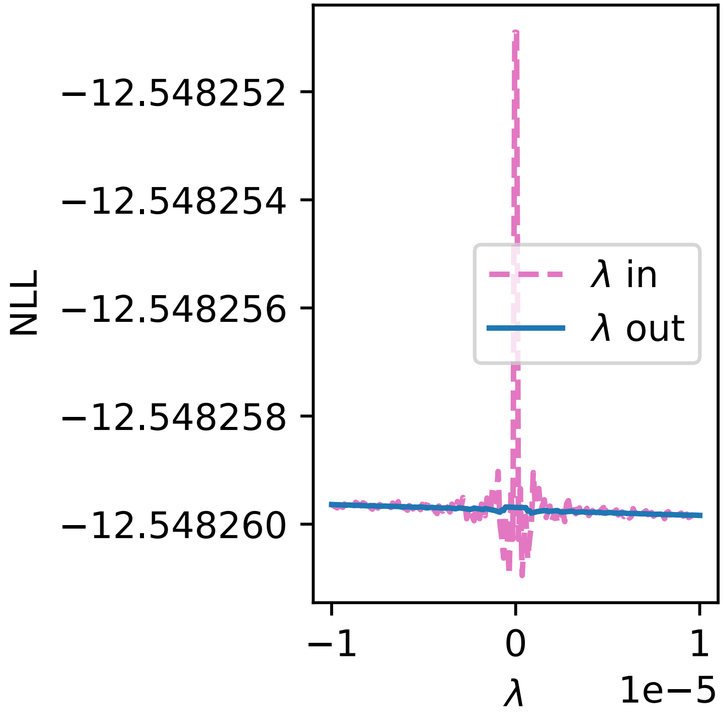
Challenge 1: Catastrophic Cancellation

- $\mathrm{NLL}(\lambda, x) = (1-\lambda)\sum_{i}^{n} \ln x_i + \frac{n}{2} \ln \mathrm{Var}[\psi(\lambda, x)]$
- Numerical issues arise when computing $\ln \mathrm{Var}[\psi(\lambda, x)]$ for $\lambda \neq 0$
- **Eq. (1):** Original formulation —> $\ln \mathrm{Var}[\psi(\lambda, x)] = \ln \mathrm{Var}[(x^{\lambda} - 1)/\lambda]$
- **Eq. (2):** Remove constant term —> $\ln \mathrm{Var}[\psi(\lambda, x)] = \ln \mathrm{Var}(x^{\lambda}/\lambda)$
- **Eq. (3):** Factor out the λ term —> $\ln \mathrm{Var}[\psi(\lambda, x)] = \ln \mathrm{Var}(x^{\lambda}) - 2 \ln |\lambda|$

Solution 1: Carefully Choose Formulations

- Remove constant term and factor out λ works best





Takeaway 1: Equivalent formulas can behave differently, careful analysis is essential.

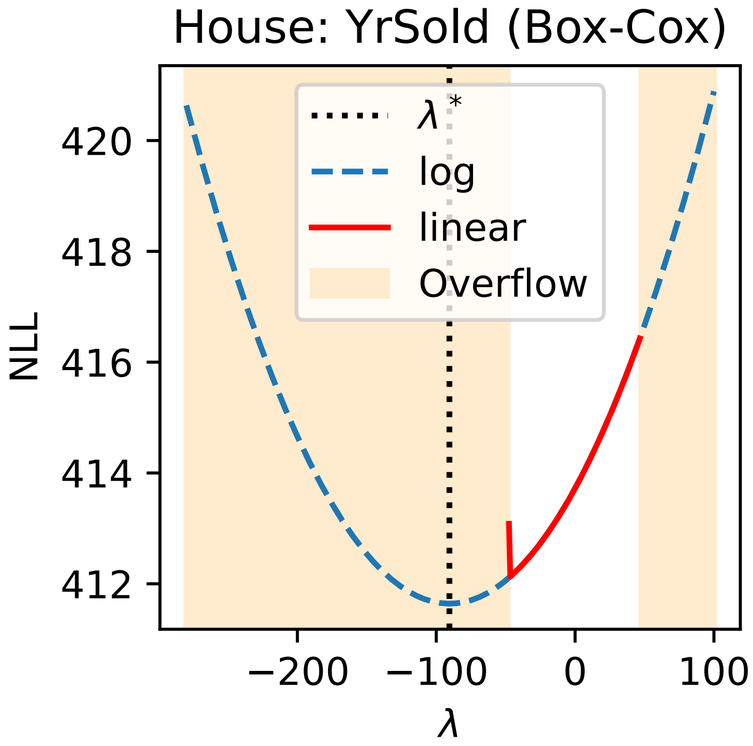
Challenge 2: Numerical Overflow

- Real-world data can already be problematic; adversarial data makes it worse
- Extreme skewness and small variance amplify instability
- Increase floating-point precision is insufficient
- Optimization is affected: derivative-based (unstable) vs. derivative-free (stable)

Overflow	Adversarial Data	λ^*	NLL	$\psi(\lambda^*, x)$	Max Value (Precision)
Positive	[10, 10, 10, 9.9]	357.55	14.18	9.96e+354	1.80e+308 (Double)
Negative	[0.1, 0.1, 0.1, 0.101]	-361.15	32.62	-3.87e+358	
Positive	[10, 10, 10, 9.999]	35933.3	32.61	5.85e+35928	1.19e+4932 (Quadruple)
Negative	[0.1, 0.1, 0.1, 0.10001]	-35936.9	51.03	-2.30e+35932	

Solution 2: Log-domain Computation

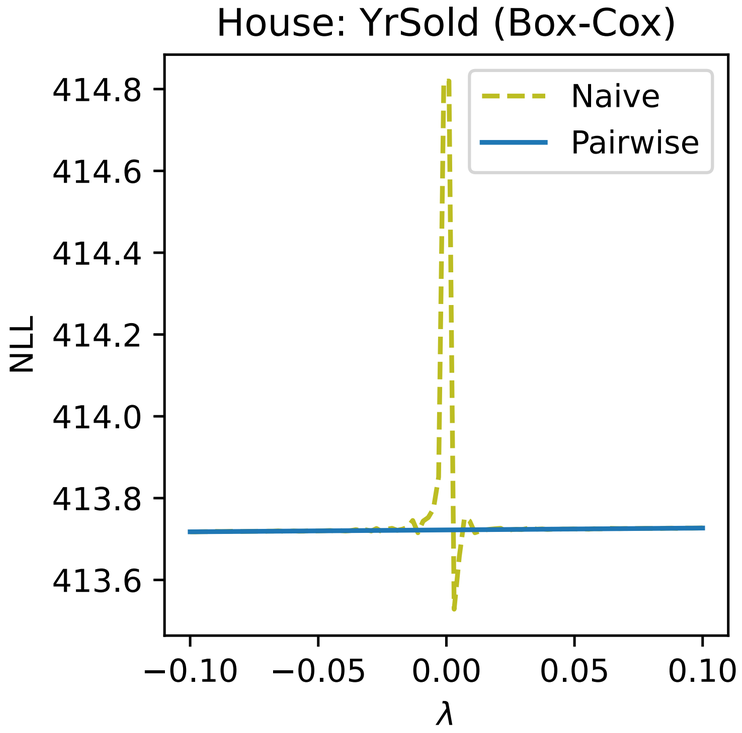
- Goal: Compute $\ln \mathrm{Var}[\psi(\lambda, x)]$ without overflow
- Linear-domain pipeline:
  - $x \rightarrow \psi(\lambda, x) \rightarrow \mathrm{Var}[\psi(\lambda, x)] \rightarrow \ln \mathrm{Var}[\psi(\lambda, x)]$
- Log-domain pipeline:
  - $x \rightarrow \ln x \rightarrow \ln \psi(\lambda, x) \rightarrow \ln \mathrm{Var}[\psi(\lambda, x)]$
  - The final step uses the Log-Sum-Exp trick



Takeaway 2: Use log-domain computation when handling extreme values.

Challenge 3: Federated Aggregation

- One-pass variance computation over distributed data
- Naive method: client send $(n, \sum_i x_i, \sum_i x_i^2)$
- Server aggregate n, $\sum_i x_i$, and $\sum_i x_i^2$ via summation
- $\mathrm{Var}(x) = \frac{1}{n}\sum_i x_i^2 - \frac{1}{n^2}\left(\sum_i x_i\right)^2$
- Numerical instability: smooth curves has spikes

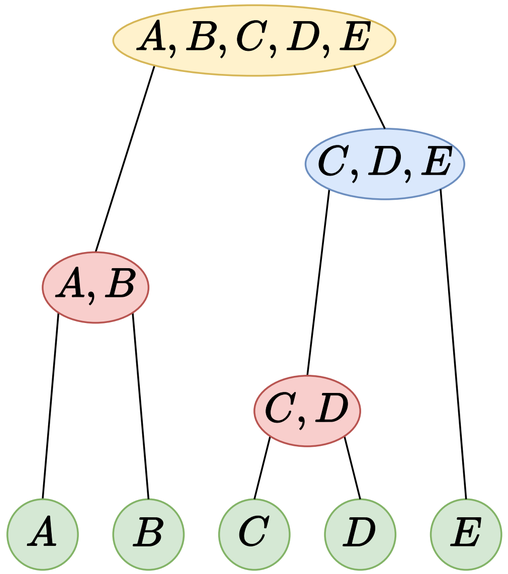


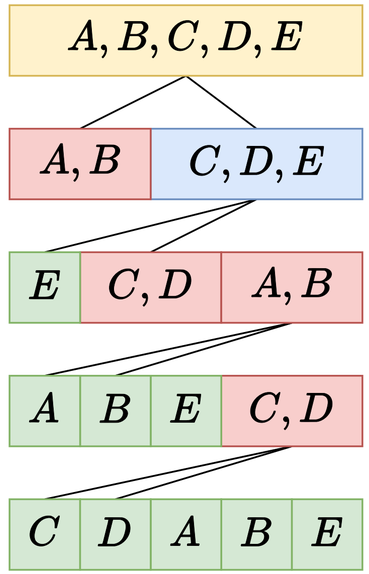
Marchand, T., Muzellec, B., B´eguier, C., Ogier du Terrail, J., and Andreux, M. (2022).

Securefedyj: a safe feature gaussianization protocol for federated learning. NeurIPS.

Solution 3: Pairwise algorithm

- Client send $(n, \bar{x}, s)$, where $s = \sum_i (x_i - \bar{x})^2$
- $n^{(AB)} = n^{(A)} + n^{(B)}$
- $\bar{x}^{(AB)} = \bar{x}^{(A)} + (\bar{x}^{(B)} - \bar{x}^{(A)}) \cdot \frac{n^{(B)}}{n^{(AB)}}$
- $s^{(AB)} = s^{(A)} + s^{(B)} + (\bar{x}^{(B)} - \bar{x}^{(A)})^2 \cdot \frac{n^{(A)} n^{(B)}}{n^{(AB)}}$
- Combine with log-domain computation for full stability





Takeaway 3: Avoid textbook one-pass variance formula in numerical computing.

Chan, T. F., Golub, G. H., and LeVeque, R. J. (1982). Updating formulae and a pairwise algorithm for computing sample variances. COMPSTAT.

SciPy and Scikit-learn

- SciPy: Improved implementation of the following modules
  - scipy.stats.boxcox_normmax: Compute optimal transform parameter
  - scipy.stats.boxcox_llf and scipy.stats.yeojohnson_llf: Log-likelihood functions
  - scipy.special.boxcox and scipy.special.boxcox1p: Transform function
  - scipy.special.inv_boxcox and scipy.special.inv_boxcox1p: Inverse transform
- Scikit-learn:
  - Now relies on SciPy instead of maintaining a separate implementation

Conclusion

- Numerical issues are pervasive in scientific computing
- Naive implementations can lead to severe instability
- We provide a principled treatment for power transforms
- Our techniques generalize to a broader class of methods

https://xuefeng-xu.github.io/powertf.html