
Tabular Numeric Stretch Transformation

Zihao Ye Juyong Kim Johnna Sundberg Burak Varici Pradeep Ravikumar
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
{zihaoYe, juyongk, jsundber, bvarici}@andrew.cmu.edu
pradeepr@cs.cmu.edu

Abstract

Tabular data presents unique challenges for deep learning due to its heterogeneous nature, where numeric features exhibit diverse distributions, scales, and statistical properties. Although recent advances have improved how models learn from tabular data, how numeric data are transformed into model-friendly representations remains comparatively underexplored. We introduce the *stretch transformation framework*, which formulates numeric feature preprocessing as an optimization problem to make the target function smoother and thus more learnable. Our framework has two variants: (1) *unsupervised stretch*, which uniformly redistributes feature density via minimax optimization, and (2) *supervised stretch*, which optimizes target-aware numeric feature transformations from the perspective of target-function smoothness by minimizing the target function’s Dirichlet energy in the transformed space. Our theoretical analysis further connects this framework to several popular transformations: unsupervised stretch is closely related to Piecewise Linear Encoding through a shared piecewise-linear geometry and approaches the empirical CDF transformation as the number of bins grows, while supervised stretch becomes closely related to target encoding in the fine-binning limit. Comprehensive experiments on 38 datasets from the TALENT benchmark demonstrate that supervised stretch consistently outperforms all baselines. These results show that explicitly optimizing for target function smoothness is a powerful and under-explored strategy for tabular deep learning.

1 Introduction

Most recent progress in tabular deep learning has focused on improving how models learn from data through better architectures, optimization, or training protocols. A complementary question is how to transform numeric data itself into representations that are easier for neural networks to learn from. This question is important because numeric tabular features can combine heterogeneous quantities, binary flags, counts, percentages, monetary values, and heavy-tailed continuous variables, each with distinct scales and statistical properties [12]. This heterogeneity creates a challenging optimization landscape for neural networks, which typically assume well-conditioned inputs [19].

While tree-based methods naturally handle diverse distributions through split-based decisions, neural networks require careful preprocessing to achieve competitive performance [35, 23]. Thus, the *numeric feature transformation* step is needed to bridge the gap between raw heterogeneous data and model-friendly features. Commonly used transformations include standardization, power transformations [36], quantile transformations [26], and Piecewise Linear Encoding (PLE) [10]. Remarkably, although target information (e.g., labels) is typically available during training, most existing numeric feature transformations are unsupervised. A supervised PLE variant, PLE-T, is a notable exception, but it uses targets only to set bin boundaries, not to optimize the transformation itself from the perspective of target-function learnability.

We introduce the *stretch transformation framework*, which systematically incorporates target information to create more learnable numeric representations. Our key insight is that feature transformation should not merely normalize distributions but make the target function smoother in the transformed space. This reduces the design of numeric feature transformations to a width-allocation problem: how much transformed-space width should be assigned to different regions of a numeric feature? We answer this question via two complementary approaches:

Supervised stretch is, to our knowledge, the first method to formulate target-informed numeric feature transformation explicitly as target-function smoothness optimization. By minimizing the Dirichlet energy of the target function in the transformed space, we derive optimal width allocations that concentrate more resolution in regions where the target varies rapidly. This principled approach achieves strong empirical gains, demonstrating the untapped potential of supervised feature transformation.

Unsupervised stretch provides a robust fallback when target information is unavailable or unreliable. Using minimax optimization, it maximizes the worst-case sample separation, effectively creating a piecewise linear approximation of the empirical CDF transformation. This variant matches or exceeds PLE’s performance while requiring only $O(1)$ memory per feature instead of $O(T)$.

Beyond practical improvements, our framework offers theoretical explanations for existing empirical observations. We show that unsupervised stretch explains why CDF transformation can reduce frequency content and improve learnability [3], despite being label-agnostic. Similarly, supervised stretch reveals deep connections to target encoding, providing a theoretical justification for why target-based transformations can be effective while offering principled regularization through the number of bins T . **In summary, our contributions are:**

- We introduce supervised stretch, a systematic framework that uses target information to optimize numeric feature transformations by maximizing target-function smoothness.
- We propose unsupervised stretch, a memory-efficient alternative to PLE that achieves comparable or superior performance without dimensional expansion.
- We connect our framework to several existing transformations: unsupervised stretch shares a piecewise-linear geometry with PLE and approaches the empirical CDF transformation as the number of bins grows, and supervised stretch becomes closely related to target encoding in the fine-binning limit.
- We empirically validate the framework on 38 datasets from the TALENT benchmark, where supervised stretch consistently outperforms all baselines, with the largest gains in regression tasks.

These results challenge the prevailing paradigm of unsupervised feature transformation in tabular deep learning and open new avenues for supervised preprocessing methods.

2 Related Work

Tabular Machine Learning is a rapidly evolving field [30, 4, 29, 13]. Recent work has proposed various neural architectures for tabular data, including attention-based models [9, 2] and MLP-based models [10, 17, 11], achieving strong performance on standard benchmarks [8]. Beyond architectural innovations, several studies have explored pretraining for large-scale tabular datasets [20], while recent advances in in-context learning have enabled foundation models to perform well on small datasets without task-specific finetuning [15, 16, 27].

Feature Transformations for Tabular Data. Effective preprocessing is crucial for tabular machine learning, as heterogeneous features, such as binary indicators, counts, and heavy-tailed continuous variables with disparate scales, pose distinct challenges for neural networks [12]. This complexity necessitates feature transformation strategies tailored to tabular data. While tree-based models (e.g., Random Forests, Gradient Boosted Trees) are invariant to monotonic transformations [6], neural networks often require standardized inputs to stabilize optimization [19].

Standardization (z-score normalization) remains the most common preprocessing technique, mapping each feature to zero mean and unit variance [26]. but assumes approximate normality and offers no robustness to skewness or outliers. Power transformations address non-normality: the Yeo-Johnson transformation [36] extends Box-Cox [5] to handle negative values, producing more Gaussian-like features, which benefits neural networks and has been widely adopted in tabular ma-

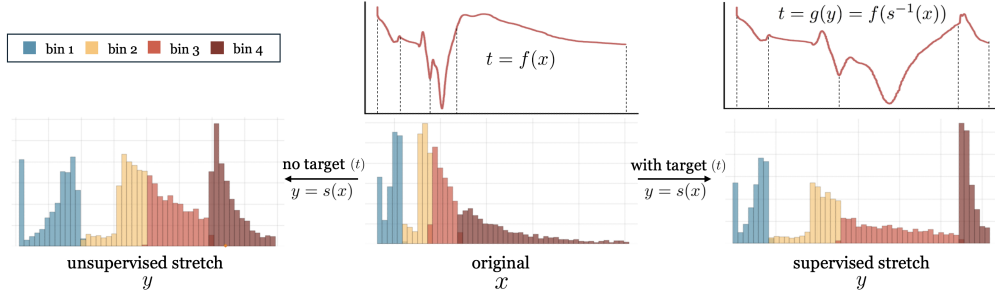


Figure 1: Overview: Stretch Transformation Framework. **(Center)** An original feature x with a complex target function $f(x)$, partitioned into four quantile bins (colors). **(Left)** Unsupervised stretch uniformly redistributes feature density. **(Right)** Supervised stretch uses target information to allocate more *stretch* to bins where the target function varies rapidly (e.g., bin 3). This creates a new target function $g(y)$ in the transformed space that is significantly smoother and thus easier for a neural network to learn.

chine learning [15]. Piecewise linear encoding (PLE) [10] discretizes the distribution into quantile-based bins and encodes each into a vector, which has also proven effective in tabular tasks.

Neural Network Inductive Biases. Despite being universal approximators [7, 18], neural networks exhibit specific inductive biases that can influence their performance in tabular data. For instance, the spectral bias phenomenon, whereby ReLU networks preferentially learn low-frequency functions before high-frequency components, has been well-documented [28, 34]. Interestingly, [3] observe that tabular data contains substantially higher frequency components compared to natural images. Unlike images defined on a fixed pixel grid, numeric features imply continuity, allowing samples to be arbitrarily close to each other. This characteristic creates unbounded high-frequency components that are notoriously difficult for neural networks to learn due to spectral bias. This observation provides one lens through which to understand the empirical performance gap between neural networks and tree-based methods on tabular benchmarks, and motivates preprocessing methods that change not only feature marginal distributions, but also the smoothness of the induced feature-target map.

3 Stretch Feature Transformation

At a high level, stretch transformations preserve the ordering of each numeric feature while reallocating transformed-space length across regions. They can expand regions where the target changes rapidly and compress regions with little target-relevant variation.

Denote a tabular dataset by $\mathcal{D} = \{(\mathbf{x}_i, t_i)\}_{i=1}^n$, where $\mathbf{x}_i \in \mathbb{R}^d$ are features and t_i is the corresponding target.¹ While targets are available during training, feature transformations may leverage this information (supervised) or ignore it (unsupervised), though most existing methods are unsupervised. A feature transformation $s : \mathcal{X} \rightarrow \mathcal{Y}$ preprocesses the raw features, creating the learning pipeline:

$$\mathbf{x} \xrightarrow{s} \mathbf{y} = s(\mathbf{x}) \xrightarrow{g_\theta} \hat{t} = g_\theta(\mathbf{y}) \quad (1)$$

For an invertible transformation with the inverse $h = s^{-1}$, the target function in the transformed space becomes $g(y) = \mathbb{E}[t|y] = f(h(y))$ where $f(x) = \mathbb{E}[t|x]$ is the original target function. The neural network g_θ learns to approximate g in the transformed space. The goal is to find s such that $g = f \circ h$ is easier for neural networks to learn than the original f .

Feature transformations can be characterized along several axes: they may be unsupervised (using only feature statistics, e.g., standardization, quantile transformation) or supervised (leveraging target information); dimension-preserving ($d' = d$) or dimension-expanding ($d' > d$, as in PLE); and marginal, operating on features independently, or joint, modeling feature interactions. These properties can be combined as well, for instance, distribution-shaping followed by standardization.

¹For simplicity, we write t_i as the target. Our framework is directly applicable to both scalar targets (e.g., class labels, single value regression) and multidimensional vector targets ($t_i \in \mathbb{R}^k$).

For numeric features, desirable properties of a transformation s include: (1) *monotonicity*, preserving the natural ordering to avoid arbitrary permutations that could destroy meaningful relationships; (2) *bounded output range*, e.g., $[0, 1]$ or $[-1, 1]$, to ensure inputs are well-scaled for neural networks.

Our stretch transformation framework, introduced shortly, provides both unsupervised and supervised variants that satisfy these requirements through a piecewise linear design. The framework operates marginally on each feature while preserving dimensionality. For the supervised variant, target information is used *only* during transformation design; the learned transformation is then fixed and applied identically to all data.

3.1 Stretch Transformation Setup

We introduce a feature transformation that rescales different regions of the input via a monotone piecewise linear map. For a scalar feature $x \in \mathbb{R}$ with range $[x_{\min}, x_{\max}]$, we first partition the domain into T intervals with boundaries $\mathbf{b} = \{b_0, b_1, \dots, b_T\}$ where $b_0 = x_{\min}$ and $b_T = x_{\max}$. We adopt quantile-based binning as the default partition, ensuring approximately equal sample counts per bin. This process groups samples $\{x_i\}_{i=1}^n$ (assumed to be sorted) into T consecutive bins. We denote the set of sample indices i whose corresponding feature value x_i falls into the t -th bin, $[b_{t-1}, b_t)$, as $\mathcal{I}_t$. The transformation $s : \mathbb{R} \rightarrow [0, 1]$ is then defined by

$$s(x) = c_{t-1} + \frac{x - b_{t-1}}{b_t - b_{t-1}} \cdot w_t, \quad x \in [b_{t-1}, b_t), \quad (2)$$

where $w_t > 0$ is the width allocated to interval t , satisfying $\sum_{t=1}^T w_t = 1$. The term $c_t := \sum_{i=1}^t w_i$ (with $c_0 = 0$) denotes the cumulative width. This form is strictly increasing (hence order-preserving and bijective onto $[0, 1]$). The slope within each interval is

$$s'(x) = \frac{w_t}{b_t - b_{t-1}}, \quad \forall x \in [b_{t-1}, b_t), \quad t \in [T]. \quad (3)$$

Thus, by choosing $\{w_t\}$, we can control the local stretch (large w_t) or squeeze (small w_t) of each region. **The central design problem is how to set the width vector $\{w_t\}_{t=1}^T$, which we derive from a principled optimization objective.**

3.2 Optimization Objective: Dirichlet Energy Minimization

Recall from Eqn. (1) that $g(y) = f(h(y))$ is the target function in the transformed space, where f is vector-valued for classification. A smoother g in the y -space is easier for neural networks to learn due to their spectral bias toward low-frequency functions [34, 28]. Thus, we seek transformations that maximize the smoothness of g . To make this smoothness objective precise, we adopt a kernel-smoothing perspective: a function is smoother if it changes less under small-bandwidth smoothing. We derive this connection below, with the full derivation provided in Section A.1. To quantify smoothness, we consider the kernel correlation with a Gaussian RBF kernel $k_\sigma(y, z) = \exp(-(y - z)^2 / (2\sigma^2))$:

$$\langle g, T_\sigma g \rangle = \int_0^1 \int_0^1 g(y) k_\sigma(y, z) g(z) dy dz, \quad (4)$$

where T_σ is the integral operator associated with k_σ . In the small bandwidth regime ($\sigma \rightarrow 0$), the heat kernel expansion [31, 24] gives $T_\sigma = e^{\frac{\sigma^2}{2} \partial_{yy}} = I + \frac{\sigma^2}{2} \partial_{yy} + O(\sigma^4)$. To isolate the term independent of smoothness, we use the centered version:

$$\langle g, (T_\sigma - I)g \rangle = \langle g, \frac{\sigma^2}{2} \partial_{yy} g \rangle + O(\sigma^4) = -\frac{\sigma^2}{2} \int_0^1 \|g'(y)\|_2^2 dy + O(\sigma^4), \quad (5)$$

where the last equality follows from integration by parts. Therefore, maximizing kernel smoothness is equivalent to minimizing the Dirichlet energy:

$$\min_h \mathcal{E}[g] := \int_0^1 \|g'(y)\|_2^2 dy. \quad (6)$$

Recall that our stretch transformation allocates width w_t to bin t , controlling how much of the transformed space $[0, 1]$ is dedicated to each region. Given samples $\{x_i\}_{i=1}^n$ with transformed positions $y_i = s(x_i)$, using the piecewise-linear interpolant through the transformed samples gives

the following finite-sample discrete Dirichlet energy:

$$\mathcal{E}_{\text{disc}} = \sum_{i=1}^{n-1} \frac{\|\Delta f_i\|_2^2}{\Delta y_i}, \quad \text{where} \quad \Delta f_i := f(x_{i+1}) - f(x_i), \quad \Delta y_i := y_{i+1} - y_i. \quad (7)$$

This objective depends on both the target increments Δf_i and the spacings in the transformed space Δy_i . The spacings Δy_i are determined by our width allocations: within bin t with width w_t , the spacings sum to w_t . The key question then is how to choose $\{w_t\}$ to minimize this energy, which we answer in two distinct settings of unsupervised and supervised.

3.3 Unsupervised Stretch: Uniform Allocation

Without the target information, we cannot directly evaluate Δf_i in Eqn. (7). Instead, we adopt a minimax principle and minimize the worst-case Dirichlet energy over all target functions with bounded local variation. Specifically, assuming $\|\Delta f_i\|_2^2 \leq C$ for all i , the worst-case Dirichlet energy for spacing allocation $\{\Delta y_i\}$ becomes

$$\max_{\{\Delta f_i\}} \sum_{i=1}^{n-1} \frac{\|\Delta f_i\|_2^2}{\Delta y_i} \quad \text{s.t.} \quad \|\Delta f_i\|_2^2 \leq C, \quad \forall i. \quad (8)$$

The maximizer is straightforwardly $\|\Delta f_i\|_2^2 = C$ for all i , leading to the optimization problem (see Appendix A.2 for detailed derivation):

$$\min_{\{\Delta y_i\}} \sum_{i=1}^{n-1} \frac{1}{\Delta y_i} \quad \text{s.t.} \quad \sum_{i=1}^{n-1} \Delta y_i = 1. \quad (9)$$

Within our binning framework with T quantile bins containing $n_t \approx n/T$ samples each, the optimal allocation is then uniform both within and across bins, yielding:

$$w_t^* = \frac{1}{T}, \quad \forall t \in \{1, \dots, T\}. \quad (10)$$

Connection to empirical CDF. With this uniform allocation, unsupervised stretch becomes a piecewise linear approximation of the empirical CDF transformation. As $T \rightarrow n$, our method converges to the exact empirical CDF, which maps each sample to its normalized rank. This insight provides a theoretical justification for the empirical finding in Beyazit et al. [3] that CDF transformation reduces frequency content and improves learnability. By maximizing the minimum spacing between samples, we effectively smooth the target function, reducing the high-frequency components. However, as noted in Beyazit et al. [3], reducing high-frequency components does not guarantee a better performance, since excessive smoothing can eliminate important signals. The parameter T provides a natural trade-off: a small T preserves more of the original distribution’s structure, while a large T approaches full CDF transformation. This controllable interpolation between preserving distributional features and achieving uniform density is a key advantage of our framework.

Connection to Piecewise Linear Encoding (PLE). Although unsupervised stretch and PLE [10] appear fundamentally different—scalar versus T -dimensional outputs—they share an underlying geometric structure. PLE maps a sample x in bin t to:

$$\text{PLE}(x) = [1, \dots, 1, \frac{x - b_{t-1}}{b_t - b_{t-1}}, 0, \dots, 0] \in \mathbb{R}^T. \quad (11)$$

This creates a piecewise linear path in $\mathbb{R}^T$ from origin to $(1, \dots, 1)$. The arc length along this path is

$$L(x) = (t-1) + \frac{x - b_{t-1}}{b_t - b_{t-1}} = T \cdot \text{Unsupervised-Stretch}(x). \quad (12)$$

Thus, unsupervised stretch precisely parameterizes the PLE manifold by normalized arc length. Both achieve identical quantile-based density redistribution, differing only by coordinate representation. This equivalence explains their similar empirical performance (Section 4.2), while unsupervised stretch offers significant computational advantages: $O(1)$ versus $O(T)$ memory per feature and no dimensional expansion.

3.4 Supervised Stretch: Target-Informed Allocation

When target information is available, we can optimize the discrete Dirichlet energy in Eqn. (7) directly. However, substituting the piecewise linear map proves to be numerically unstable, as the

resulting objective is highly sensitive to the original feature spacings Δx_i (see Appendix A.3 for a detailed discussion). To overcome this, we instead optimize a robust lower bound on the energy.

We partition the samples into T bins. For bin t with width $w_t = \sum_{i \in \mathcal{I}_t} \Delta y_i$, let $S_t := \sum_{i \in \mathcal{I}_t} \|\Delta f_i\|_2$ be the bin-wise total variation of the target. For fixed $w_t > 0$, minimizing the within-bin contribution of Eqn. (7) over $\{\Delta y_i\}_{i \in \mathcal{I}_t}$ subject to $\sum_{i \in \mathcal{I}_t} \Delta y_i = w_t$ yields a lower bound:

$$\sum_{i \in \mathcal{I}_t} \frac{\|\Delta f_i\|_2^2}{\Delta y_i} \geq \frac{S_t^2}{w_t}, \quad (13)$$

with equality at $\Delta y_i \propto \|\Delta f_i\|_2$ within each bin. Then, the global optimization over $\{w_t\}$ becomes

$$\min_{\{w_t > 0\}} \sum_{t=1}^T \frac{S_t^2}{w_t} \quad \text{s.t.} \quad \sum_{t=1}^T w_t = 1. \quad (14)$$

This convex problem has the closed-form solution:

$$w_t^* = \frac{S_t}{\sum_{u=1}^T S_u}. \quad (15)$$

Intuitively, in the transformed space, we allocate a larger space to the regions where the target function varies rapidly, effectively equalizing the slope magnitudes across bins and creating a smoother target function in the transformed space.

Connection to target encoding. Supervised stretch connects naturally to target encoding when $T = n$ (one bin per unique value). In this limit, the optimal width for the interval $[x_i, x_{i+1}]$ becomes:

$$w_i = \frac{|f_{i+1} - f_i|}{\sum_{k=1}^{n-1} |f_{k+1} - f_k|}. \quad (16)$$

This is remarkably similar to applying min-max scaling to target encoding. Standard target encoding maps $x_i \mapsto f_i = \mathbb{E}[t|x = x_i]$, and if we scale these values to $[0, 1]$, we get essentially the same transformation for monotonic targets. The key insight is that both methods fundamentally use target variation to guide the transformation, target encoding does it directly, while supervised stretch achieves it via Dirichlet energy minimization. Our framework thus provides a theoretical justification for why target-based transformations are effective: they implicitly smooth the target function in the transformed space.

While target encoding is widely applied to categorical features, its use for numeric features has been limited, partly due to concerns about overfitting or interpretability. The proposed supervised stretch offers a regularized, theoretically grounded approach to incorporating target information for numeric features, with the number of bins T serving as a natural regularization parameter.

Out-of-fold estimation. In practice, to prevent information leakage, we use K -fold cross-validation with adaptive Nadaraya-Watson kernel regression to obtain $\hat{f}(x_i)$ for each sample without using its own target value [25, 32] (details in Appendix B.1). The bin widths are computed using $\Delta \hat{f}_i$ in place of Δf_i .

4 Experiments

We evaluate the proposed stretch transformations against established baselines across comprehensive tabular benchmarks, assessing their impact on diverse neural architectures. The experiments are designed to answer three questions: whether target-aware smoothness optimization improves numeric feature transformation, whether the gains depend on task type, and how robust the conclusions are across models and evaluation criteria. Our results show that supervised stretch has the strongest aggregate performance, with particularly clear advantages in regression. Detailed experimental configurations and additional results are provided in Appendix B.3.

4.1 Experimental Setup

Datasets and Models. We use the TALENT benchmark suite [35, 23], focusing on the *Tiny Benchmark 1* collection of 26 classification and 12 regression datasets. This benchmark covers diverse tabular learning tasks with varying dataset sizes, feature types, and target distributions. Categorical features are encoded as indices by default, with RealMLP internally converting them to one-hot en-

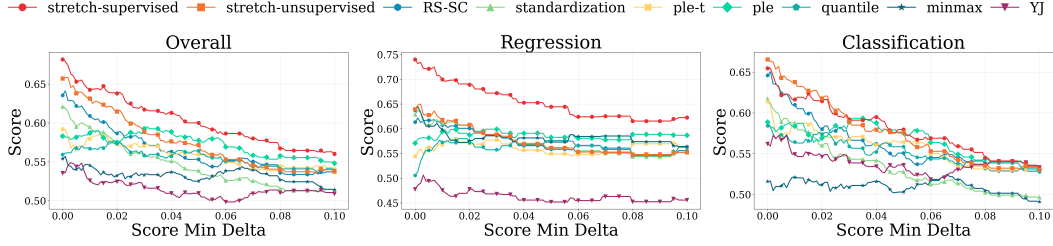


Figure 2: Sensitivity analysis of significance thresholds. Average normalized score is plotted against a fixed minimum separation δ . Larger δ values treat small gaps as ties (score 0.5), leading all methods to converge toward 0.5. Unlike the adaptive threshold used in Table 1, this analysis varies δ across fixed values. Critically, the relative performance ranking remains stable across all reasonable thresholds well before convergence, confirming robustness to threshold choice. Key takeaways: **stretch-supervised** shows the strongest regression performance, and both **stretch-supervised** and **stretch-unsupervised** show top-tier performance in classification. **Abbreviations:** PLE-T (PLE with Tree-based binning), Quantile (Quantile Gaussian), RS-SC (RobustScale+SmoothClip), and YJ (Yeo Johnson).

coding as per its design [17]. Our investigation focuses on numeric feature transformations. Detailed characteristics of these datasets, including the breakdown of numeric versus categorical feature proportions, sample sizes, and feature counts, are provided in Appendix D.

We evaluate five representative neural architectures for tabular data: FT-Transformer (FTT) [9], standard MLP, MLP-PLR [10], RealMLP [17], and ResNet [14]. Experiments span 38 datasets, yielding 190 dataset-model combinations. This is a controlled transformation comparison rather than an end-to-end model leaderboard. For RealMLP, the RS-SC cell retains its original RobustScale+SmoothClip numeric preprocessing. For every alternative transformation, we disable the internal RS-SC stage so that the transformations are not stacked; model and training hyperparameters are nevertheless tuned under the same Optuna protocol in every cell.

Transformation Methods. We compare our proposed supervised and unsupervised stretch against seven established baselines: **standardization** (z-score normalization), **Yeo-Johnson (YJ)** power transformation [36], **quantile transformation** to a Gaussian distribution, **min-max** scaling to $[0, 1]$, **RobustScale+SmoothClip (RS-SC)** as originally used in RealMLP [17], **Piecewise Linear Encoding (PLE)** [10], and **PLE with Tree-based binning (PLE-T)**. PLE is an unsupervised method that typically uses quantile-based binning, whereas PLE-T is a supervised variant that leverages a tree-based model to create bins informed by the target variable. To our knowledge, PLE-T is the only existing supervised transformation for numeric features in the literature and thus serves as our primary supervised baseline. For transformations that only adjust distribution shape (Yeo-Johnson, quantile, PLE, PLE-T, and stretch variants), we apply standardization afterward to ensure consistent scaling. Implementation details regarding computational constraints are provided in Appendix B.2.

Evaluation Protocol. Following the benchmark protocols [35, 23], we use the official TALENT train, validation, and test partitions. We conduct Bayesian optimization using Optuna [1] with 100 trials for each dataset-model-transformation combination, jointly tuning model and transformation hyperparameters (e.g., number of bins for stretch and PLE) using only the training and validation partitions. The test partition is used only after configuration selection. Each selected configuration is evaluated with 15 random seeds on the fixed split; these seeds measure initialization, minibatch-order, and optimization variability, not variability over alternative data splits. We use accuracy for classification and R^2 (clipped to $[0, 1]$) for regression as primary metrics.

Aggregation Protocol. For aggregate comparisons across heterogeneous datasets we follow Grinztajn et al. [12] and apply per-(dataset, model) min-max normalization to obtain a Normalized Score, paired with an adaptive significance filter that maps statistically inconsequential pairs to a tie (0.5); these tie pairs are also excluded from the Avg. Acc and Avg. R^2 columns of Table 1. We also consider head-to-head comparisons of different methods by scrutinizing whether their difference is statistically significant (see Figure 3): t_1 wins over t_2 only when its performance exceeds t_2 ’s by more than the larger seed standard deviation. The exact formulas, threshold definitions, and filtering rules are given in Appendix B.3; the sensitivity analysis in Figure 2 sweeps a range of fixed thresholds to confirm that the rankings are not artefacts of any particular choice.

Table 1: Performance of transformations grouped by model. Normalized Score is computed separately within each (dataset, model) panel and therefore compares transformations within a model; it should not be used to compare model quality across rows. It uses an adaptive significance filter: for each panel we collect the per-method seed standard deviation σ_t (computed over 15 seeds) and use the *median* of $\{\sigma_t\}$ across methods as the threshold; if the gap between the best and worst method is no larger than this median seed std the pair is considered a tie (score 0.5). Tie pairs are excluded from the ‘Avg. Acc’ and ‘Avg. R^2 ’ calculations. The trailing “ $\pm$ ” on Avg. Acc and Avg. R^2 is the standard error of the mean across datasets within the task category, $\sigma_{\text{across datasets}}/\sqrt{n}$, where n is the number of (dataset, model) pairs that contribute to the cell after tie-exclusion. If any single (dataset, model) pair contains a method whose seed std exceeds 0.20 (i.e., per-pair seed SEM $> 0.20/\sqrt{15} \approx .05$), that pair is flagged as “ $> .05$ ” rather than mixed into the per-cell average. For RealMLP, RS-SC denotes its original numeric preprocessing; the other transformation cells disable the internal RS-SC stage while retaining the common hyperparameter-tuning protocol. For performance stratified by dataset feature composition, see Appendix E. **Abbreviations:** Sup. (Stretch Supervised), Unsup. (Stretch Unsupervised).

Metric	Model	Transformations								
		Sup.	Unsup.	Minmax	PLE	PLE-T	Quantile	RS-SC	Standard	YJ
Overall Score	fit	0.6942	0.6543	0.3824	0.6405	0.6461	0.6559	0.5643	0.6214	0.5536
	mlp	0.6343	0.6275	0.5600	0.6835	0.6182	0.5526	0.6653	0.6423	0.5790
	mlp_plr	0.6238	0.7347	0.6037	0.4513	0.5763	0.5673	0.6091	0.6443	0.5802
	realmlp	0.7078	0.6348	0.6321	0.5457	0.5967	0.5566	0.6602	0.5919	0.4482
	resnet	0.7237	0.6610	0.6175	0.5999	0.5294	0.4854	0.6989	0.6205	0.5160
Cls. Score	fit	0.6815	0.6391	0.3434	0.6711	0.6929	0.6838	0.5419	0.6003	0.5598
	mlp	0.5738	0.6322	0.4900	0.6806	0.6324	0.5509	0.6480	0.6525	0.5679
	mlp_plr	0.6134	0.7265	0.5379	0.4578	0.5675	0.6077	0.6473	0.6354	0.6215
	realmlp	0.7074	0.6784	0.6799	0.5051	0.6110	0.5871	0.6961	0.6112	0.4656
	resnet	0.6809	0.6702	0.5473	0.6180	0.5570	0.5033	0.7054	0.5914	0.5763
Reg. Score	fit	0.7217	0.6872	0.4669	0.5743	0.5446	0.5953	0.6129	0.6671	0.5400
	mlp	0.7653	0.6173	0.7117	0.6897	0.5874	0.5561	0.7028	0.6203	0.6030
	mlp_plr	0.6464	0.7525	0.7463	0.4372	0.5954	0.4796	0.5262	0.6635	0.4907
	realmlp	0.7089	0.5403	0.5287	0.6337	0.5656	0.4907	0.5824	0.5501	0.4104
	resnet	0.8165	0.6409	0.7697	0.5606	0.4697	0.4464	0.6847	0.6838	0.3855
Avg. Acc	fit	0.827 $\pm .004$	0.824 $\pm .003$	0.805 $\pm .002$	0.826 $\pm .002$	0.826 $\pm .002$	0.827 $\pm .003$	0.819 $\pm .003$	0.816 $\pm .004$	0.821 $\pm .003$
	mlp	0.821 $\pm .003$	0.825 $\pm .002$	0.815 $\pm .002$	0.828 $\pm .002$	0.822 $\pm .002$	0.816 $\pm .002$	0.823 $\pm .002$	0.819 $\pm .002$	0.822 $\pm .002$
	mlp_plr	0.826 $\pm .001$	0.826 $\pm .002$	0.821 $\pm .002$	0.821 $\pm .002$	0.821 $\pm .003$	0.827 $\pm .002$	0.823 $\pm .002$	0.821 $\pm .002$	0.821 $\pm .003$
	realmlp	0.837 $\pm .002$	0.838 $\pm .001$	0.839 $\pm .002$	0.833 $\pm .002$	0.835 $\pm .002$	0.837 $\pm .002$	0.838 $\pm .002$	0.833 $\pm .001$	0.834 $\pm .002$
	resnet	0.826 $\pm .002$	0.824 $\pm .003$	0.815 $\pm .003$	0.824 $\pm .002$	0.823 $\pm .002$	0.816 $\pm .002$	0.825 $\pm .002$	0.819 $\pm .002$	0.820 $\pm .002$
Avg. R^2	fit	0.691 $\pm .002$	0.681 $\pm .002$	0.666 $\pm .003$	0.680 $\pm .006$	0.667 $\pm .003$	0.681 $\pm .002$	0.673 $\pm .003$	0.682 $\pm .001$	0.667 $\pm .001$
	mlp	0.630 $\pm .005$	0.606 $\pm .007$	0.628 $\pm .008$	0.608 $\pm .039$	0.588 $\pm .011$	0.619 $\pm .005$	0.619 $\pm .005$	0.591 $\pm .042$	0.545 $\pm > .05$
	mlp_plr	0.698 $\pm .004$	0.696 $\pm .004$	0.659 $\pm .004$	0.686 $\pm .002$	0.702 $\pm .002$	0.669 $\pm .005$	0.681 $\pm .004$	0.675 $\pm .003$	0.649 $\pm .005$
	realmlp	0.736 $\pm .004$	0.685 $\pm .004$	0.679 $\pm .003$	0.736 $\pm .003$	0.723 $\pm .003$	0.707 $\pm .005$	0.689 $\pm .004$	0.686 $\pm .002$	0.683 $\pm .008$
	resnet	0.651 $\pm .003$	0.631 $\pm .006$	0.655 $\pm .002$	0.639 $\pm > .05$	0.631 $\pm .005$	0.625 $\pm .011$	0.640 $\pm .002$	0.641 $\pm .003$	0.542 $\pm > .05$

4.2 Results and Analysis

Our analysis draws on three complementary perspectives. First, Figure 2 presents a sensitivity analysis using a range of fixed significance thresholds (δ) to demonstrate the robustness of our findings. Second, Table 1 provides a detailed breakdown for each model using an adaptive threshold based on the median standard error of each specific evaluation. This reveals how different architectures interact with each transformation under a statistically grounded criterion. Finally, the pairwise win rate heatmap in Figure 3 offers direct head-to-head comparisons to identify the most consistently superior methods. Together, these analyses lead to four key findings:

1. Supervised stretch achieves the strongest aggregate performance, especially in regression.

Across all evaluations, supervised stretch stands out as the strongest method, with its advantage most pronounced in regression, where it maintains a substantial margin over all competitors across significance thresholds δ (Figure 2, middle); the competition is tighter in classification but it remains in the top tier (model-specific scores in Table 1). The win-rate analysis (Figure 3) further shows that it statistically outperforms every alternative under the pairwise criterion, with decisive regression win-loss records of **18-7** against PLE, **28-9** against PLE-T, and **14-4** against standardization (Appendix C). These results support our hypothesis that smoothing target functions via Dirichlet energy minimization improves neural network performance, and the method is not fragile under label noise: with label flipping (classification) and additive Gaussian target noise (regression), supervised stretch keeps a competitive mean rank across noise levels and is top-ranked at the highest level tested (Appendix G).

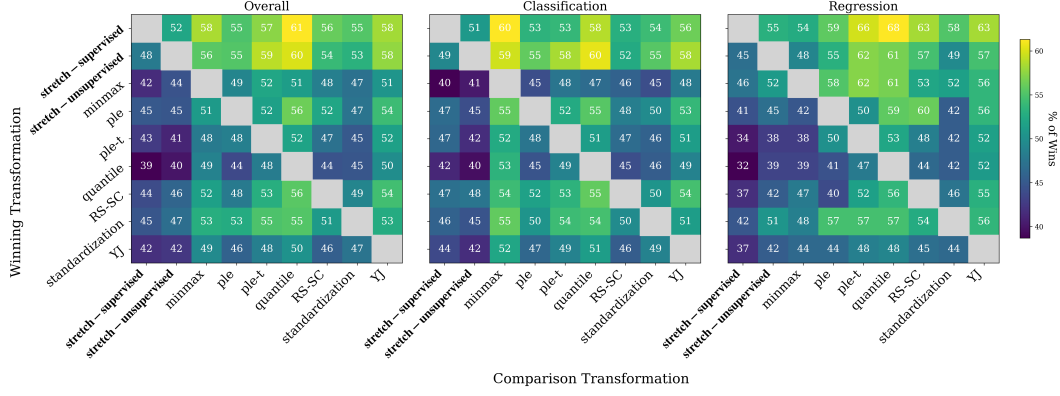


Figure 3: Pairwise win-rates between transformations, grouped by task. Each cell (i, j) shows the percentage of times transformation i (row) statistically outperforms transformation j (column). Supervised stretch has the strongest aggregate pairwise performance across methods and tasks. Unsupervised stretch is the strongest unsupervised baseline under this criterion, second only to its supervised counterpart, with particularly competitive performance in classification. For detailed win-loss-tie analysis against the strongest unsupervised (PLE, Standardization) and supervised (PLE-T) baselines, see Appendix C.

2. Unsupervised stretch is a strong target-agnostic method. Unsupervised stretch consistently ranks as a strong runner-up: it outperforms all methods except its supervised counterpart under the aggregate pairwise comparison (Figure 3) and is especially competitive in classification (Figure 2, right), with classification win-loss records of **38–25** vs. PLE, **38–18** vs. PLE-T, and **26–13** vs. Standardization (Section C). Its per-feature memory overhead is $O(1)$ compared to PLE’s $O(T)$, avoiding computational bottlenecks that can make PLE impractical for large datasets.

3. Transformation preferences vary by architecture and task type. Although supervised stretch is the strongest generalist, architectural preferences remain (e.g., unsupervised stretch is the top choice for MLP-PLR). More importantly, task type drives the largest differences: regression benefits more from advanced transformations than classification, with supervised stretch yielding the largest gains in continuous prediction tasks (Figure 2, middle). This suggests that explicitly engineering smoother feature-to-target mappings is especially valuable for regression, an underexplored direction in a literature largely focused on classification.

4. Non-uniform marginal signals help explain why smoothing is beneficial, especially for regression. Our marginal analysis (Appendix F) shows that real-world tabular features exhibit non-uniform marginal distributions with localized regions of sharp variation (high relative marginal slopes); this structural heterogeneity creates high-frequency components that are difficult for neural networks to learn due to spectral bias. Supervised stretch explicitly identifies and smooths these high-gradient regions, which is particularly relevant for regression with continuous targets and helps explain why our gains are larger there than in classification, where only a decision boundary is needed.

5 Conclusion

This paper introduced the **stretch transformation framework**, a principled approach to numeric feature preprocessing for tabular deep learning that frames transformation as Dirichlet-energy minimization of the target function. The framework yields two methods: **supervised stretch**, which systematically exploits target information to create smoother target functions and achieves the strongest empirical performance, and **unsupervised stretch**, which maximizes worst-case separation through uniform density redistribution. Our analysis also explains existing techniques: unsupervised stretch clarifies why empirical CDF transformation improves learning despite being target-agnostic and reveals its equivalence to PLE via arc-length parameterization, while supervised stretch connects to target encoding through shared use of target variation, with the number of bins providing principled regularization. Empirically, supervised stretch obtains consistent gains across 38 datasets,

supporting the hypothesis that target-aware transformation enhances neural network performance and challenging the prevailing paradigm of unsupervised preprocessing in tabular deep learning.

Limitations and Future Work. Our framework’s design could be enhanced by exploring smoother, spline-based alternatives, and adaptive binning strategies. The current transformation is marginal and therefore does not optimize a joint or conditional smoothness objective; the downstream model may learn feature interactions, but the preprocessing itself is not interaction-aware. Theoretically, the analysis could be generalized beyond the small- σ RBF kernel approximation. Applying Stretch as an option in a TabPFN inference-time preprocessing ensemble is also an interesting direction that is outside the controlled architecture study here.

References

- [1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework, 2019. URL <https://arxiv.org/abs/1907.10902>.
- [2] Sercan Ö Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 6679–6687, 2021.
- [3] Ege Beyazit, Jonathan Kozaczuk, Bo Li, Vanessa Wallace, and Bilal H Fadhallah. An inductive bias for tabular deep learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL <https://openreview.net/forum?id=XEUclJegGt>.
- [4] Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 35(6):7499–7519, June 2024. ISSN 2162-2388. doi: 10.1109/tnnls.2022.3229161. URL <http://dx.doi.org/10.1109/TNNLS.2022.3229161>.
- [5] G. E. P. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, 26(2):211–252, 1964. ISSN 00359246. URL <http://www.jstor.org/stable/2984418>.
- [6] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939785. URL <https://doi.org/10.1145/2939672.2939785>.
- [7] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.
- [8] Nick Erickson, Lennart Purucker, Andrej Tschalzev, David Holzmüller, Prateek Mutalik Desai, David Salinas, and Frank Hutter. Tabarena: A living benchmark for machine learning on tabular data, 2025. URL <https://arxiv.org/abs/2506.16791>.
- [9] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. In *NeurIPS*, 2021.
- [10] Yury Gorishniy, Ivan Rubachev, and Artem Babenko. On embeddings for numerical features in tabular deep learning. In *NeurIPS*, 2022.
- [11] Yury Gorishniy, Akim Kotelnikov, and Artem Babenko. Tabm: Advancing tabular deep learning with parameter-efficient ensembling. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=Sd4wYYOhmY>.
- [12] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? *Advances in neural information processing systems*, 35:507–520, 2022.

- [13] John Hancock and Taghi Khoshgoftaar. Survey on categorical data for neural networks. *Journal of Big Data*, 7, 04 2020. doi: 10.1186/s40537-020-00305-w.
- [14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL <https://arxiv.org/abs/1512.03385>.
- [15] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. Tabpfn: A transformer that solves small tabular classification problems in a second. In *International Conference on Learning Representations 2023*, 2023.
- [16] Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 01 2025. doi: 10.1038/s41586-024-08328-6. URL <https://www.nature.com/articles/s41586-024-08328-6>.
- [17] David Holzmüller, Leo Grinsztajn, and Ingo Steinwart. Better by default: Strong pre-tuned MLPs and boosted trees on tabular data. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL <https://openreview.net/forum?id=3BNPUDvqMt>.
- [18] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.
- [19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR. URL <https://proceedings.mlr.press/v37/ioffe15.html>.
- [20] Myung Jun Kim, Léo Grinsztajn, and Gaël Varoquaux. Carte: pretraining and transfer for tabular learning. *arXiv preprint arXiv:2402.16785*, 2024.
- [21] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.
- [22] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL <http://yann.lecun.com/exdb/mnist/>.
- [23] Si-Yang Liu, Hao-Run Cai, Qi-Le Zhou, and Han-Jia Ye. Talent: A tabular analytics and learning toolbox. *arXiv preprint arXiv:2407.04057*, 2024.
- [24] S. A. Molchanov. Diffusion Processes and Riemannian Geometry. *Russian Mathematical Surveys*, 30(1):1–63, February 1975. doi: 10.1070/RM1975v030n01ABEH001400.
- [25] E. Nadaraya. On estimating regression. *Theory of Probability and Its Applications*, 9:141–142, 1964. URL <https://api.semanticscholar.org/CorpusID:120067924>.
- [26] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.
- [27] Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. TabICL: A tabular foundation model for in-context learning on large data. In *Forty-second International Conference on Machine Learning*, 2025. URL <https://openreview.net/forum?id=0VvD1PmNzM>.
- [28] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International conference on machine learning*, pages 5301–5310. PMLR, 2019.
- [29] Ravid Shwartz-Ziv and Amitai Armon. Tabular data: Deep learning is not all you need, 2021. URL <https://arxiv.org/abs/2106.03253>.

- [30] Boris van Breugel and Mihaela van der Schaar. Why tabular foundation models should be a research priority, 2024. URL <https://arxiv.org/abs/2405.01147>.
- [31] S. R. S. Varadhan. On the behavior of the fundamental solution of the heat equation with variable coefficients. *Communications on Pure and Applied Mathematics*, 20(2):431–455, 1967. doi: <https://doi.org/10.1002/cpa.3160200210>. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpa.3160200210>.
- [32] Geoffrey S. Watson. Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A (1961-2002)*, 26(4):359–372, 1964. ISSN 0581572X. URL <http://www.jstor.org/stable/25049340>.
- [33] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.
- [34] Zhi-Qin John Xu, Yaoyu Zhang, Tao Luo, Yanyang Xiao, and Zheng Ma. Frequency principle: Fourier analysis sheds light on deep neural networks. *arXiv preprint arXiv:1901.06523*, 2019.
- [35] Han-Jia Ye, Si-Yang Liu, Hao-Run Cai, Qi-Le Zhou, and De-Chuan Zhan. A closer look at deep learning on tabular data. *arXiv preprint arXiv:2407.00956*, 2024.
- [36] In-Kwon Yeo and Richard A. Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 12 2000. ISSN 0006-3444. doi: 10.1093/biomet/87.4.954. URL <https://doi.org/10.1093/biomet/87.4.954>.

A Theoretical Foundations and Derivations

A.1 General Objective: From Kernel Smoothness to Dirichlet Energy

Motivation via Heat Kernel Expansion. The primary goal is to find a transformation that makes the target function smoother in the transformed space. We quantify smoothness using kernel correlation with a Gaussian RBF kernel, $k_\sigma(y, z) = \exp(-(y - z)^2/(2\sigma^2))$. The integral operator T_σ associated with this kernel can be expressed via the heat kernel expansion as $T_\sigma = e^{\frac{\sigma^2}{2}\partial_{yy}}$, where ∂_{yy} is the second derivative operator [31, 24]. For a small bandwidth σ , a Taylor expansion gives $T_\sigma = I + \frac{\sigma^2}{2}\partial_{yy} + O(\sigma^4)$.

To isolate the smoothness-dependent term, we consider the centered kernel correlation:

$$\langle g, (T_\sigma - I)g \rangle = \left\langle g, \left(\frac{\sigma^2}{2}\partial_{yy} + O(\sigma^4) \right) g \right\rangle = \frac{\sigma^2}{2} \int_0^1 g(y) \cdot g''(y) dy + O(\sigma^4). \quad (17)$$

Using integration by parts (assuming Neumann boundary conditions, $g'(0) = g'(1) = 0$), this simplifies to:

$$\int_0^1 g(y) \cdot g''(y) dy = [g(y) \cdot g'(y)]_0^1 - \int_0^1 g'(y) \cdot g'(y) dy = - \int_0^1 \|g'(y)\|_2^2 dy. \quad (18)$$

Therefore, maximizing kernel smoothness in this regime is equivalent to minimizing the Dirichlet energy:

$$\langle g, (T_\sigma - I)g \rangle = -\frac{\sigma^2}{2} \int_0^1 \|g'(y)\|_2^2 dy + O(\sigma^4), \quad \text{so we minimize } \mathcal{E}[g] = \int_0^1 \|g'(y)\|_2^2 dy. \quad (19)$$

For vector-valued targets, this identity is applied componentwise and summed over the target dimensions.

Discretization of Dirichlet Energy. For a dataset $\{x_i\}_{i=1}^n$ (assumed sorted) with transformed values $y_i = s(x_i)$, we approximate the continuous target function $g(y)$ by its piecewise linear interpolant, denoted by $\tilde{g}$. The interpolant $\tilde{g}$ is constructed by connecting the points $(y_i, f(x_i))$. By the definition of our transformation, the true function value at the transformed point y_i is precisely $g(y_i) = f(s^{-1}(y_i)) = f(x_i)$.

The derivative of this interpolant, $\tilde{g}'(y)$, is constant over any interval $[y_i, y_{i+1}]$ and is given by:

$$\tilde{g}'(y) = \frac{f(x_{i+1}) - f(x_i)}{y_{i+1} - y_i} = \frac{\Delta f_i}{\Delta y_i}.$$

The total discrete Dirichlet energy is the integral of the squared norm of this derivative, which simplifies to a sum over the intervals:

$$\mathcal{E}_{\text{disc}} = \sum_{i=1}^{n-1} \int_{y_i}^{y_{i+1}} \|\tilde{g}'(y)\|_2^2 dy = \sum_{i=1}^{n-1} \frac{\|\Delta f_i\|_2^2}{\Delta y_i}. \quad (20)$$

This discrete energy serves as the foundation for both our unsupervised and supervised methods.

A.2 Unsupervised Stretch: Derivation of the Minimax Objective

Worst-Case Dirichlet Energy Formulation. Without target information, we formulate a minimax problem to find the spacing allocation that minimizes the worst-case Dirichlet energy. The optimization is performed over all possible target functions f with bounded local variation, i.e., $\|\Delta f_i\|_2^2 \leq C$ for all i . Since the discrete Dirichlet energy $\mathcal{E}_{\text{disc}} = \sum_{i=1}^{n-1} \frac{\|\Delta f_i\|_2^2}{\Delta y_i}$ depends only on the sequence of difference vectors $\{\Delta f_i\}_{i=1}^{n-1}$, this is equivalent to solving the following problem:

$$\max_{\{\Delta f_i\}} \sum_{i=1}^{n-1} \frac{\|\Delta f_i\|_2^2}{\Delta y_i} \quad \text{s.t.} \quad \|\Delta f_i\|_2^2 \leq C, \quad \forall i. \quad (21)$$

This bounded local variation constraint assumes that for an unknown function, the magnitude of its change over any adjacent interval is bounded, which is a practical assumption for unsupervised scenarios where we want to be robust against arbitrary local fluctuations.

Solving the Minimax Problem. The inner maximization is straightforward. Since each term $\frac{\|\Delta f_i\|_2^2}{\Delta y_i}$ is nonnegative and increasing in $\|\Delta f_i\|_2^2$, the worst case is attained at the upper bound:

$$\|\Delta f_i\|_2^2 = C, \quad \forall i. \quad (22)$$

Substituting this into the objective yields:

$$\max_{\{\Delta f_i\}} \sum_{i=1}^{n-1} \frac{\|\Delta f_i\|_2^2}{\Delta y_i} = C \sum_{i=1}^{n-1} \frac{1}{\Delta y_i}. \quad (23)$$

Since C is a positive constant, the outer minimization over spacings becomes:

$$\min_{\{\Delta y_i > 0\}} \sum_{i=1}^{n-1} \frac{1}{\Delta y_i} \quad \text{s.t.} \quad \sum_{i=1}^{n-1} \Delta y_i = 1. \quad (24)$$

Uniform Spacing Solution. This is a convex optimization problem. The Lagrangian is:

$$\mathcal{L}(\{\Delta y_i\}, \lambda) = \sum_{i=1}^{n-1} \frac{1}{\Delta y_i} + \lambda \left(\sum_{i=1}^{n-1} \Delta y_i - 1 \right). \quad (25)$$

Taking derivative with respect to Δy_i and setting it to zero gives:

$$-\frac{1}{(\Delta y_i)^2} + \lambda = 0, \quad (26)$$

which implies that all Δy_i are equal. Using the constraint $\sum_{i=1}^{n-1} \Delta y_i = 1$, we obtain:

$$\Delta y_i^* = \frac{1}{n-1}, \quad \forall i. \quad (27)$$

Thus, the minimax-optimal spacing allocation is uniform.

A.3 Derivation of Supervised Objective

Exact Piecewise-Linear Objective. When target information is available, the discrete Dirichlet energy from Appendix A.1 can be written explicitly under the piecewise-linear stretch map. Consider bin $B_t = [b_{t-1}, b_t]$ with allocated width w_t . For adjacent samples $x_i, x_{i+1} \in B_t$, the transformation is linear within the bin, so

$$\Delta y_i = y_{i+1} - y_i = \frac{w_t}{b_t - b_{t-1}} \Delta x_i, \quad (28)$$

where $\Delta x_i = x_{i+1} - x_i$. Substituting this into the discrete Dirichlet energy gives the exact within-bin contribution

$$E_t = \frac{b_t - b_{t-1}}{w_t} \sum_{i \in I_t} \frac{\|\Delta f_i\|_2^2}{\Delta x_i}, \quad (29)$$

where $I_t = \{i : x_i, x_{i+1} \in B_t\}$. This exact objective is sensitive to very small input spacings: finite-sample noise in estimated target differences can make this term unstable when Δx_i is very small.

Robust Lower-Bound Surrogate. To obtain a more stable bin-level objective, we use a lower bound based on the total target variation within each bin. Define

$$S_t = \sum_{i \in I_t} \|\Delta f_i\|_2. \quad (30)$$

Since the allocated width of bin t satisfies $\sum_{i \in I_t} \Delta y_i = w_t$, Cauchy-Schwarz gives

$$\left(\sum_{i \in I_t} \|\Delta f_i\|_2 \right)^2 \leq \left(\sum_{i \in I_t} \frac{\|\Delta f_i\|_2^2}{\Delta y_i} \right) \left(\sum_{i \in I_t} \Delta y_i \right). \quad (31)$$

Therefore,

$$\sum_{i \in I_t} \frac{\|\Delta f_i\|_2^2}{\Delta y_i} \geq \frac{S_t^2}{w_t}. \quad (32)$$

We adopt this robust lower bound as our objective:

$$\min_{\{w_t > 0\}} \sum_{t=1}^T \frac{S_t^2}{w_t} \quad \text{s.t.} \quad \sum_{t=1}^T w_t = 1. \quad (33)$$

This objective is a robust bin-level surrogate rather than the exact unrestricted sample-spacing optimum; equality would additionally require the within-bin transformed spacings to align with target variation. The bin count T controls the corresponding trade-off: smaller T preserves more local structure, whereas larger T allows finer, potentially noisier, width reallocation. Solving the Lagrangian yields

$$w_t^* = \frac{S_t}{\sum_{u=1}^T S_u}. \quad (34)$$

B Implementation and Experimental Details

B.1 Out-of-Fold Kernel Regression for Supervised Stretch

A naive implementation of supervised stretch would use the entire training set to estimate the target function $f(x)$ needed to compute the bin widths $\{w_t\}$. This, however, would introduce significant information leakage. The transformation s would be designed using the same target values t_i that the downstream model g_θ is later trained to predict. In essence, the target information would be used twice, once to shape the feature space, and again to train the model within that space. This can lead to overly optimistic performance estimates and poor generalization.

To prevent this, we employ a standard K -fold cross-validation scheme to obtain out-of-fold (OOF) estimates of the marginal regression function, $\hat{f}(x_i)$, for each sample. This ensures that t_i is not used to construct its own out-of-fold estimate $\hat{f}(x_i)$. The final bin widths are then computed from OOF predictions across the training set, reducing direct self-label leakage while still using target information at the transformation-design stage.

For classification, we use stratified K -fold ($K = 10$, or fewer if a class is too small) on the one-hot encoded targets. For regression, we use standard K -fold. Within each fold, we fit an adaptive Nadaraya-Watson kernel regressor on the training portion of the data to generate predictions for the validation portion [25, 32]. Let $\mathcal{D}_{\text{tr}}$ denote the set of indices of the samples in the training portion for a given fold. The conditional expectation at a point x in the validation set is then estimated as:

$$\hat{f}(x) = \frac{\sum_{j \in \mathcal{D}_{\text{tr}}} K_h(x, x_j) \cdot Y_j}{\sum_{j \in \mathcal{D}_{\text{tr}}} K_h(x, x_j) + \epsilon}, \quad \text{where} \quad K_h(x, x_j) = \exp\left(-\frac{(x - x_j)^2}{2h(x)^2}\right). \quad (35)$$

Here, Y_j is the target value for the j -th sample, and the adaptive bandwidth $h(x)$ is determined by the distance to the k -th nearest neighbor of x within the training data, allowing the smoothness of the estimate to vary with local sample density.

After obtaining OOF predictions for all samples, we sort them by the feature value and compute the total variation S_t of these predictions within each quantile bin. Several fallback mechanisms ensure robustness: if a feature has too few unique values, or if the total target variation is negligible, we revert to simpler transformations (identity or unsupervised stretch).

B.2 Computational Fallback Scheme for PLE

PLE expands feature dimensionality from d to $d \times T$, which can become computationally prohibitive. In our experiments, if the resulting tensor size exceeds a memory limit, we use a fallback scheme that substitutes standardization for PLE on that specific dataset-model combination. This affects fewer than 5% of the 190 combinations and does not impact our proposed methods.

B.3 Experimental Setup Details

Dataset Selection. From the original TALENT Benchmark 1 collection of 42 datasets, we exclude four datasets that contain only categorical features (Amazon_employee_access, BNG(tic-tac-toe), led24, splice), yielding our final experimental suite of 38 datasets.

Categorical Feature Handling. We use indices encoding as the default. FT-Transformer learns embeddings, RealMLP internally converts to one-hot, and other models use the indices directly. For feature-expanding transformations like PLE, FT-Transformer applies a linear projection to each expanded dimension to create tokens.

Hyperparameter Search Spaces. We conduct Bayesian optimization using Optuna with 100 trials per configuration. The search spaces for models and transformations are detailed in Table 3 and Table 2. We follow the hyperparameter search space design from the original TALENT benchmark [35, 23], including their special syntax for sampling methods, which we define here for clarity:

- `?distribution[default, ...args]`: An “optional” parameter. With 50% probability, the `default` value is used (e.g., 0.0 for dropout, disabling it). Otherwise, a value is sampled from the specified `distribution` using the remaining arguments.
- `$name[...]`: A special, complex sampling function defined by the benchmark’s codebase. For instance, `$mlp_d_layers` samples the number of layers and then the width of each layer within given bounds, with different sampling strategies for the first, middle, and last layers.

Table 2: Transformation hyperparameter search spaces.

Transformation	Parameter Search Space
Supervised/Unsupervised Stretch	<code>n_bins: ?categorical[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 100000][†]</code>
PLE	<code>n_bins: int[2, 256]</code>
Other transformations	No hyperparameters

B.4 Aggregation and Pairwise Comparison Protocol

This subsection makes the protocol summarised in Section 4 fully explicit. Let d , m , and t index datasets, models, and transformations.

Per-(dataset, model) min-max normalisation. Following Grinsztajn et al. [12], we map raw metrics to a normalised score

$$\text{score}_{d,m,t} = \frac{\text{metric}_{d,m,t} - \min_t \text{metric}_{d,m,t}}{\max_t \text{metric}_{d,m,t} - \min_t \text{metric}_{d,m,t}}, \quad (36)$$

so that the best transformation on each (d, m) pair scores 1 and the worst scores 0.

Adaptive significance filter. For each (d, m) we compute the per-method seed standard deviation $\sigma_{d,m,t}$ over the 15 evaluation seeds and set the threshold $\delta_{d,m} = \text{median}_t \sigma_{d,m,t}$. If the gap $\max_t \text{metric}_{d,m,t} - \min_t \text{metric}_{d,m,t}$ does not exceed $\delta_{d,m}$, the choice of transformation is deemed inconsequential, every $\text{score}_{d,m,t}$ is set to 0.5, and the pair is excluded from the Avg. Acc / Avg. R^2 averages of Table 1. The Normalized Score columns of Table 1 use this adaptive threshold; the sensitivity analysis in Figure 2 replaces $\delta_{d,m}$ with a range of fixed values to confirm that the ranking is robust.

Pairwise win criterion. For the head-to-head heatmap (Figure 3), a transformation t_1 is declared to win against t_2 on a given (d, m) only if

$$\text{metric}_{d,m,t_1} > \text{metric}_{d,m,t_2} + \max(\sigma_{d,m,t_1}, \sigma_{d,m,t_2}). \quad (37)$$

This requires the gap to exceed the larger of the two seed standard deviations, so that an isolated lucky seed cannot trigger a win. Pairs that do not satisfy equation 37 in either direction are recorded as ties and contribute 0.5 to each side’s count. The win-rate reported in the heatmap is the win count divided by the 190 dataset-model combinations (pairs where both methods are missing are excluded).

C Detailed Pairwise Performance Analysis

To provide a more granular analysis of the stretch transformation’s performance, this section extends the summary presented in the win-rate heatmap Section 4 (Figure 3). The pairwise comparison uses the strict win-loss-tie criterion stated in Eq. equation 37; the win-rate percentages shown in the

Table 3: Model hyperparameter search spaces.

Model	Parameter	Search Space
MLP	d.layers	\$mlp_d.layers[1, 8, 64, 2048]*
	dropout	?uniform[0.0, 0.0, 0.5] [†]
	lr	loguniform[1e-5, 0.01]
	weight_decay	?loguniform[0.0, 1e-6, 0.001] [†]
MLP-PLR	d.layers	\$mlp_d.layers[1, 8, 64, 1024]*
	dropout	?uniform[0.0, 0.0, 0.5] [†]
	n_frequencies	int[16, 96]
	frequency_scale	loguniform[0.01, 100.0]
	d.embedding	int[16, 64]
	lr	loguniform[1e-5, 0.01]
	weight_decay	?loguniform[0.0, 1e-6, 0.001] [†]
FT-Transformer	n.layers	int[1, 4]
	d.token	categorical[8, 16, 32, 64, 128]
	residual_dropout	?uniform[0.0, 0.0, 0.2] [†]
	attention_dropout	uniform[0.0, 0.5]
	ffn_dropout	uniform[0.0, 0.5]
	d.ffn_factor	uniform[0.67, 2.67]
	lr	loguniform[1e-5, 0.001]
RealMLP	weight_decay	loguniform[1e-6, 0.001]
	num_emb_type	categorical[none, pbl, pl, plr]
	add_front_scale	categorical[true, false]
	hidden_sizes	categorical[[256, 256, 256], [64, 64, 64, 64, 64], [512]]
	p_drop	categorical[0.0, 0.15, 0.30]
	act	categorical[selu, relu, mish]
	lr	loguniform[0.02, 0.3]
	wd	categorical[0.0, 0.02]
	ls_eps	categorical[0.0, 0.1]
	plr_sigma	loguniform[0.05, 0.5]
ResNet	n.layers	int[1, 8]
	d	int[64, 512]
	d_hidden_factor	uniform[1.0, 4.0]
	hidden_dropout	uniform[0.0, 0.5]
	residual_dropout	?uniform[0.0, 0.0, 0.5] [†]
	lr	loguniform[1e-5, 0.01]
	weight_decay	?loguniform[0.0, 1e-6, 0.001] [†]

heatmap are summary statistics derived directly from the raw W-L-T counts across all 190 dataset-model combinations.

Here, we look closer at these raw counts. Figures 4 through 9 present the detailed scatter plots comparing our supervised and unsupervised methods against three key baselines: PLE, PLE-T, and standardization. In each figure, every point represents a single dataset-model combination. Points located above the diagonal indicate a win for the stretch transformation, points below indicate a loss, and points on the diagonal represent a statistical tie. The total win-loss-tie counts are summarized and displayed directly on each plot, providing the quantitative confirmation for the performance advantages discussed in Section 4.2.

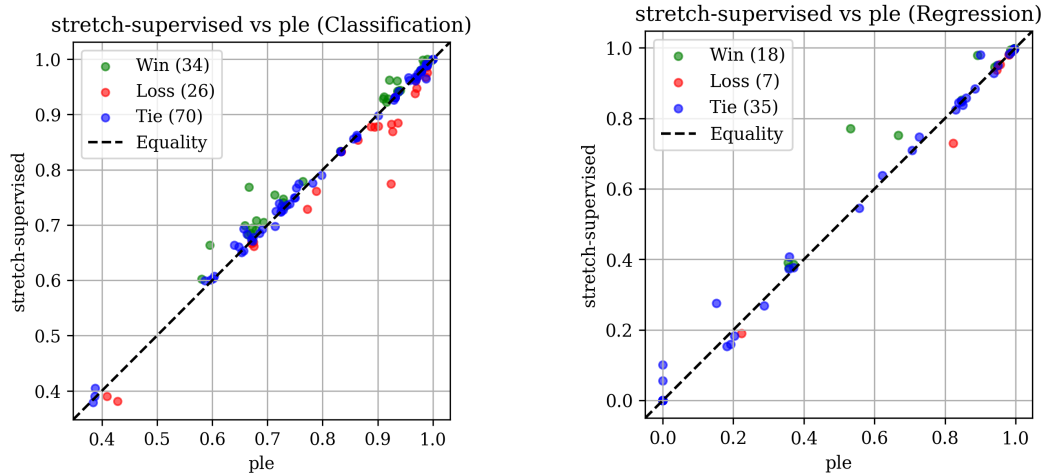


Figure 4: Stretch Supervised vs PLE

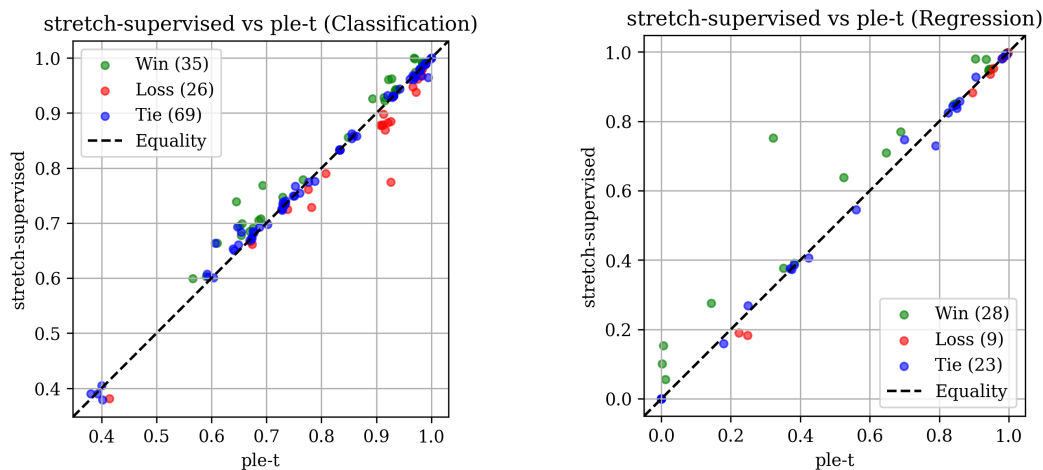


Figure 5: Stretch Supervised vs PLE-T

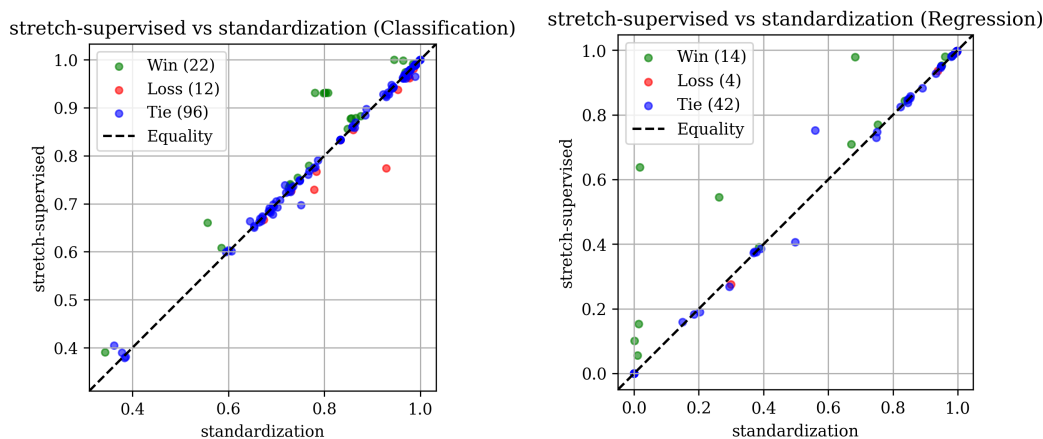


Figure 6: Stretch Supervised vs Standardization

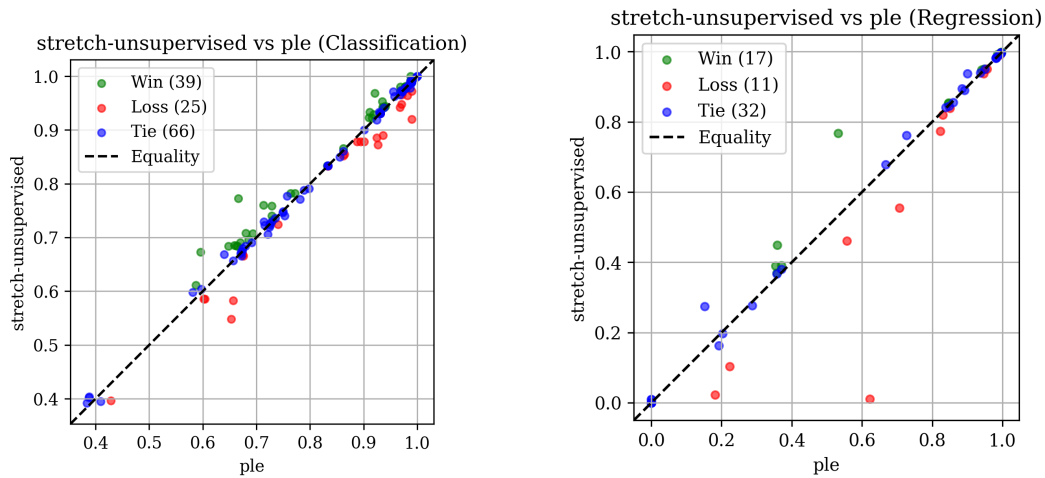


Figure 7: Stretch Unsupervised vs PLE

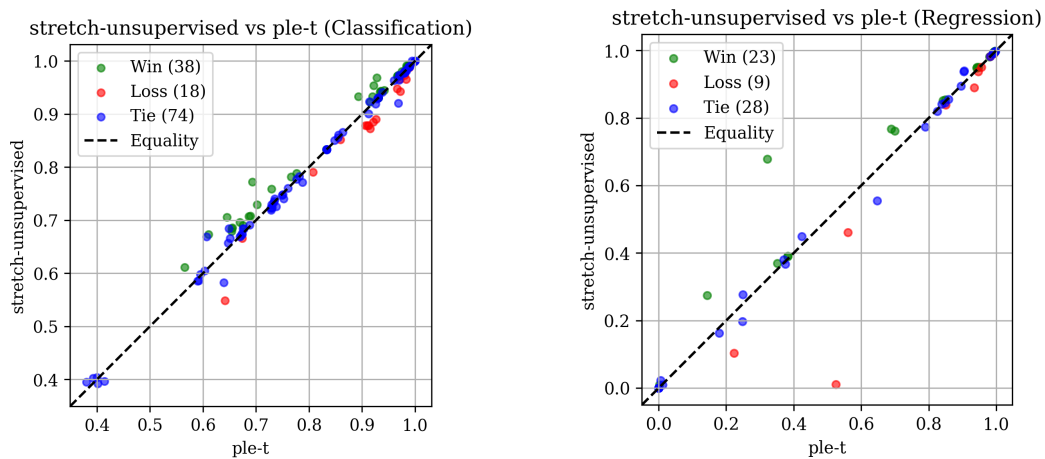


Figure 8: Stretch Unsupervised vs PLE-T

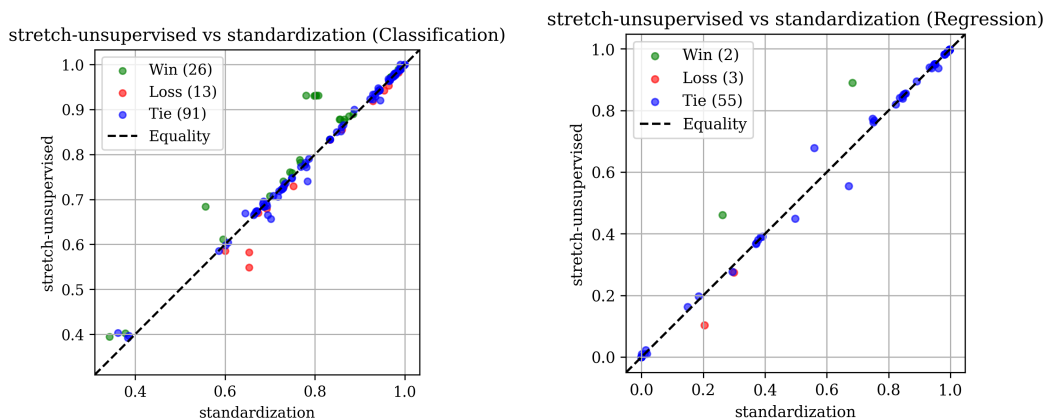


Figure 9: Stretch Unsupervised vs Standardization

Table 4: Dataset characteristics grouped by feature type dominance.

Dataset	Num Features	Cat Features	Total Samples	Imbalance Ratio	N Classes
Balanced Datasets					
Diamonds	6	3	53940	—	—
E-CommereShippingData	6	4	10999	—	—
Fitness_Club_c	3	3	1500	—	—
Kaggle_bike_sharing_demand	3	6	10886	—	1
NHANES_age_prediction	4	3	2277	—	—
Shop_Customer_Data	4	2	2000	—	—
compass	8	9	16644	1.000	—
estimation_of_obesity_levels	8	8	2111	—	—
Category Dominant Datasets					
archive_r56_Portuguese	1	29	651	—	—
okcupid_stem	2	11	26677	6.826	3
socmob	1	4	1156	—	—
thyroid-dis	6	20	2800	52.645	5
Numeric Dominant Datasets					
ASP-POTASSCO-classification	140	1	1294	12.286	11
Ailerons	40	0	13750	—	—
Bias_correction_r_2	21	0	7725	—	—
Click_prediction_small	3	0	39948	—	—
Contaminant-detection-11.0GHz	30	0	2400	—	—
FOREX_audjpy-day-High	10	0	1832	1.056	2
FOREX_audsgd-hour-High	10	0	43825	1.061	2
IEEE80211aa-GATS	27	0	4046	—	—
Intersectional-Bias-Assessment	14	5	11000	—	2
Job_Profitability	27	1	14480	—	—
Mobile_Price_Classification	14	6	2000	—	—
VulNoneVul	16	0	5692	—	2
Waterstress	22	0	1188	—	2
electricity	7	1	45312	1.355	2
htru	8	0	17898	—	—
ibm-employee-performance	23	7	1470	5.504	2
internet_firewall	7	0	65532	—	—
jungle_chess_2pcs_endgame	6	0	44819	5.320	3
kc1	21	0	2109	5.469	2
optdigits	64	0	5620	1.032	10
page-blocks	10	0	5473	175.464	5
pol	26	0	10082	1.000	2
pol_reg	48	0	15000	—	—
pole	26	0	14998	—	—
wine	4	0	2554	1.000	2
yeast	8	0	1484	92.600	10

D Dataset Composition and Feature Types

The 38 datasets in our benchmark span a wide range of feature compositions. To make this explicit, we group them by the relative proportion of numeric and categorical features:

- **Numeric Dominant** (26 datasets): numeric features clearly outnumber categorical ones, including datasets that are purely numeric.
- **Balanced** (8 datasets): the two feature types are comparable in number (within a 2:1 ratio in either direction).
- **Category Dominant** (4 datasets): categorical features dominate, with at least a 3:1 ratio over numeric features.

Table 4 lists the per-dataset statistics. Sample sizes range from about 650 to over 65,000, and feature counts from a handful up to 140. The imbalance ratio column reports class imbalance for classification tasks where applicable.

Since stretch transformations only act on numeric features, the relative amount of numeric information differs substantially across these three groups. We use this grouping in Appendix E (Table 5) to check whether the gains from stretch persist as the share of numeric features decreases.

E Performance Breakdown by Feature Composition

Table 5 reports per-model average scores within each of the three dataset groups defined in Appendix D. The main observations are:

- **Numeric Dominant:** Supervised Stretch is the top method on most models, which is the regime where our objective is most directly applicable.
- **Balanced:** results are more mixed across methods, with Supervised Stretch and PLE-T trading wins on different model-task pairs, and no single method dominating.
- **Category Dominant:** Despite numeric features being a minority, the stretch variants remain competitive. Supervised Stretch attains the highest regression Overall Score, and Unsupervised Stretch the highest classification Overall Score in this group.

Table 5: Performance by dataset category and task type (average score).

Category	Task	Model	Sup.	Unsup.	Minmax	Ple	Ple T	Quantile	RS-SC	Std	YJ
Balanced	Classification	fit	0.7987	0.8000	0.7911	0.7967	0.7994	0.8023	0.8005	0.7987	0.8014
		mlp	0.7673	0.7827	0.7738	0.7860	0.7907	0.7809	0.7855	0.7813	0.7849
		mlp-plr	0.7843	0.7994	0.7850	0.7891	0.7880	0.7875	0.7926	0.7916	0.7932
		realmlp	0.8008	0.8020	0.7984	0.7990	0.8101	0.8021	0.8001	0.8005	0.8009
		resnet	0.7844	0.7793	0.7735	0.7777	0.7871	0.7795	0.7817	0.7834	0.7868
		Overall	0.7871	0.7927	0.7844	0.7897	0.7950	0.7905	0.7921	0.7911	0.7934
	Regression	fit	0.5493	0.5478	0.5422	0.5418	0.5477	0.5482	0.5503	0.5479	0.5503
		mlp	0.4404	0.4300	0.4740	0.4051	0.4321	0.4747	0.4736	0.3896	0.4961
		mlp-plr	0.5340	0.5346	0.5375	0.4708	0.5128	0.5313	0.5346	0.5311	0.5327
		realmlp	0.5501	0.5475	0.5357	0.5445	0.5491	0.5419	0.5478	0.5471	0.5435
		resnet	0.5259	0.5172	0.5225	0.4962	0.4889	0.5274	0.5326	0.5207	0.5254
		Overall	0.5199	0.5154	0.5224	0.4917	0.5061	0.5247	0.5278	0.5073	0.5296
Category Dominant	Classification	fit	0.7207	0.7196	0.7185	0.7206	0.7191	0.7161	0.7199	0.7205	0.7119
		mlp	0.7063	0.7125	0.7054	0.7072	0.6993	0.7030	0.7102	0.7085	0.7077
		mlp-plr	0.7076	0.7086	0.7093	0.6978	0.7046	0.7082	0.7128	0.7098	0.7093
		realmlp	0.7202	0.7154	0.7168	0.7141	0.7122	0.7160	0.7173	0.7169	0.7160
		resnet	0.7107	0.7133	0.7171	0.7025	0.7049	0.7091	0.7077	0.7094	0.7116
		Overall	0.7131	0.7139	0.7134	0.7084	0.7080	0.7105	0.7136	0.7130	0.7113
	Regression	fit	0.5797	0.5845	0.5241	0.5178	0.5191	0.5790	0.5374	0.5951	0.5857
		mlp	0.4713	0.3908	0.4261	0.4453	0.2726	0.4377	0.3772	0.3813	0.4507
		mlp-plr	0.4564	0.4850	0.4760	0.5130	0.5190	0.4576	0.4366	0.4666	0.4646
		realmlp	0.5984	0.6088	0.5979	0.6122	0.5774	0.6251	0.5889	0.6134	0.5854
		resnet	0.4342	0.3587	0.4895	0.4493	0.4134	0.3550	0.3818	0.4104	0.3532
		Overall	0.5080	0.4856	0.5027	0.5076	0.4603	0.4909	0.4644	0.4934	0.4879
Numeric Dominant	Classification	fit	0.8340	0.8287	0.8048	0.8315	0.8305	0.8331	0.8216	0.8179	0.8252
		mlp	0.8313	0.8326	0.8234	0.8362	0.8283	0.8270	0.8306	0.8254	0.8294
		mlp-plr	0.8331	0.8309	0.8269	0.8279	0.8273	0.8348	0.8278	0.8250	0.8255
		realmlp	0.8444	0.8451	0.8477	0.8401	0.8392	0.8450	0.8459	0.8379	0.8401
		resnet	0.8394	0.8374	0.8248	0.8366	0.8339	0.8296	0.8387	0.8301	0.8307
		Overall	0.8364	0.8349	0.8255	0.8345	0.8318	0.8339	0.8329	0.8273	0.8302
	Regression	fit	0.8228	0.8017	0.7964	0.8260	0.7961	0.8041	0.7991	0.7993	0.7716
		mlp	0.8100	0.7951	0.7977	0.7969	0.7964	0.7757	0.7969	0.7950	0.6096
		mlp-plr	0.8884	0.8746	0.8011	0.8868	0.8892	0.8310	0.8600	0.8413	0.7879
		realmlp	0.9063	0.8013	0.8016	0.9043	0.8884	0.8435	0.8162	0.8023	0.8089
		resnet	0.8060	0.7980	0.7979	0.7966	0.7977	0.7808	0.7974	0.7979	0.6162
		Overall	0.8467	0.8141	0.7990	0.8421	0.8335	0.8070	0.8139	0.8072	0.7188

F Marginal Signal Non-Uniformity Analysis

Supervised stretch is most useful when a numeric feature’s marginal effect on the target is non-uniform, i.e., when there are local regions where the target changes much faster than elsewhere. A natural concern is whether this is actually a common pattern in tabular data, or whether most features have roughly flat marginal effects on which our method would have little to do. This appendix measures the degree of non-uniformity directly on the benchmark datasets.

F.1 Methodology

For each numeric feature in each benchmark dataset, we proceed as follows:

1. **Preprocessing.** Mean-impute missing values and standardize to zero mean and unit variance.
2. **Marginal function.** For each unique feature value x , compute $f(x) = \mathbb{E}[t \mid x]$. For regression, this is the mean of the standardized target; for classification, the empirical class-probability vector $f(x) \in \mathbb{R}^K$ (see footnote 1).
3. **Local slopes.** For consecutive unique values x_i, x_{i+1} , compute $|\Delta f_i / \Delta x_i|$, where $|\cdot|$ denotes absolute value (regression) or ℓ_2 norm (classification).
4. **Normalization.** Divide each local slope by the per-feature mean so that the rescaled slope is

$$r_i = \frac{|\Delta f_i / \Delta x_i| \cdot n}{\sum_{j=1}^n |\Delta f_j / \Delta x_j|}, \quad (38)$$

where n is the number of local slopes for that feature. This makes each feature’s slopes have mean 1, so we are looking at the *shape* of the slope distribution, not its overall magnitude. A perfectly uniform marginal effect gives $r_i \equiv 1$; non-uniformity shows up as values both above and below 1.

5. **Aggregation.** Pool the r_i values from all numeric features in a dataset into one distribution.
6. **Summary statistics.** Compute the standard deviation and skewness of this pooled distribution per dataset. Standard deviation captures how spread out the slopes are; positive skewness flags the presence of a few regions with much larger slopes than the rest.

F.2 Results

Table 6 reports the standard deviation and skewness for every dataset in the benchmark, together with four image datasets included as a reference point.

Tabular features are clearly non-uniform. Standard deviations on the tabular datasets range from 0.47 to 20.15 with a median around 1.1. If marginal effects were close to flat, the rescaled slopes would all be near 1 and the standard deviations would be small. Instead, the values are large enough that, on most features, some regions have local slopes several times larger than others. Skewness is positive on most datasets, which means the distribution has a heavy right tail: a small number of regions are much steeper than the rest of the feature.

Image data behaves differently. For comparison, MNIST [22], Fashion-MNIST [33], CIFAR-10 and CIFAR-100 [21] all sit between 0.28 and 0.39 in standard deviation, and have small to moderate skewness. Image features are pixel intensities on a fixed grid, so neighbouring pixels are equally spaced and bounded by the same range; the marginal slope distribution is correspondingly more uniform. Numeric tabular features have no such grid: samples can be arbitrarily close in feature space and targets in regression are unbounded, which is consistent with the much larger spread we see.

Non-uniformity is not specific to numeric-dominant datasets. The same pattern shows up across all three groups of Appendix D. Even category-dominant datasets contain numeric features with substantial non-uniformity, e.g., `socmob` (std = 2.03, skewness = 6.26) and `thyroid-dis` (std = 0.94, skewness = 1.20). What matters is the shape of the slope distribution per numeric feature, not how many numeric features the dataset has overall.

Connection to method behaviour. Supervised stretch reacts to exactly the kind of structure measured here. In bins where the target varies rapidly the bin widths grow; in bins where it varies little they shrink. When a feature actually is close to uniform, the bin-wise total variations S_t are similar across bins and the allocation collapses back towards uniform widths, so the transformation has little effect. The fact that most features in the benchmark have non-trivial dispersion and skewness

Table 6: Marginal distribution metrics (Std and Skewness)

Dataset	Task Type	Std	Skewness
Balanced Datasets			
Diamonds	Regression	2.1275	5.52
E-CommereShippingData	Binary	1.3061	10.28
Fitness_Club.c	Binary	1.2097	0.60
Kaggle_bike_sharing_demand_challenge	Regression	1.0102	2.47
NHANES_age_prediction	Regression	0.9773	1.57
Shop_Customer_Data	Regression	2.6387	6.50
compass	Binary	0.9258	1.02
estimation_of_obesity_levels	Multiclass	0.7353	-0.05
Category Dominant Datasets			
archive_r56_Portuguese	Regression	1.1955	1.33
okcupid_stem	Multiclass	1.1132	1.44
socmob	Regression	2.0314	6.26
thyroid-dis	Multiclass	0.9361	1.20
Numeric Dominant Datasets			
ASP-POTASSCO-classification	Multiclass	0.4705	-1.27
Ailerons	Regression	1.3136	3.48
Bias_correction_r_2	Regression	12.5727	53.98
Click_prediction_small	Binary	1.0574	0.72
Contaminant-detection-11.0GHz	Binary	1.0746	0.66
FOREX_audjpy-day-High	Binary	1.0024	0.03
FOREX_audsgd-hour-High	Binary	0.8862	0.09
IEEE80211aa-GATS	Regression	13.4710	61.56
Intersectional-Bias-Assessment	Binary	1.1410	0.33
Job_Profitability	Regression	10.1227	73.20
Mobile_Price_Classification	Multiclass	0.8191	0.74
VulNoneVul	Binary	2.3204	3.53
Waterstress	Binary	1.0665	0.14
electricity	Binary	1.1502	1.65
htru	Binary	3.3842	4.15
ibm-employee-performance	Binary	1.5927	1.21
internet_firewall	Multiclass	20.1475	53.98
jungle_chess_2pcs_endgame	Multiclass	0.6012	0.50
kc1	Binary	1.1874	0.64
optdigits	Multiclass	0.5452	2.06
page-blocks	Multiclass	2.0516	2.73
pol	Binary	1.6962	3.22
pol_reg	Regression	1.6065	2.54
pole	Regression	1.5757	2.30
wine	Binary	1.0016	1.02
yeast	Multiclass	0.8551	1.20
Image Datasets			
MNIST	Multiclass	0.353687	0.12
Fashion-MNIST	Multiclass	0.391164	0.94
CIFAR-10	Multiclass	0.380173	1.47
CIFAR-100	Multiclass	0.276569	1.75

suggests there is in practice plenty of structure for the method to act on, which is consistent with the empirical gains in Section 4.

G Noise Robustness Analysis

Since supervised stretch uses targets to design the transformation, a natural question is how it behaves when those targets are noisy. We test this by adding controlled label noise to four datasets and tracking how the relative ranking of methods changes with the noise level.

G.1 Experimental Setup

Datasets and methods. We use four datasets from our benchmark: `Contaminant-detection`, `yeast` and `ibm-employee-performance` (classification) and `NHANES-age-prediction` (regression). We compare Supervised Stretch and Unsupervised Stretch against Standardization, PLE, and PLE-T, on the same five architectures used in the main experiments.

Noise injection. For classification, we use symmetric label flipping: at noise level η , a fraction η of training labels is replaced by a uniformly random label. For regression, we add Gaussian noise to training targets, $\epsilon \sim \mathcal{N}(0, \sigma_{\text{noise}}^2)$ with $\sigma_{\text{noise}} = \eta \cdot \sigma_{\text{target}}$. Validation and test sets are kept clean, so the metric still measures recovery of the true signal.

Evaluation. Because accuracy and R^2 are not directly comparable across datasets, we report the *mean rank* of each method. For every (dataset, model) pair and every noise level, the five methods are ranked from 1 (best) to 5 (worst). We then average these ranks over the $4 \times 5 = 20$ (dataset, model) combinations. Error bars in the figures show the standard deviation of the rank across these combinations.

G.2 Results

Figure 10 shows the aggregated mean rank as a function of noise level; Figures 11a–12b show per-dataset ranks.

Supervised Stretch does not collapse under noise. The mean rank of Supervised Stretch (red) stays in a competitive range over the full noise sweep and does not show the gradual degradation seen for some baselines. At the highest noise level we test ($\eta = 0.5$), it actually has the best mean rank among the five methods. This is consistent with the role of target information in the transformation: bin widths are computed from out-of-fold smoothed estimates of $f(x)$, which average out a substantial fraction of label noise before it ever affects the transformation.

Unsupervised Stretch is a stable target-free option. Standardization (blue) is a flat reference, as expected. Unsupervised Stretch (purple) tracks the same trend across all noise levels and at $\eta = 0.5$ ranks third, slightly behind PLE. Since it uses only $O(1)$ memory per feature instead of PLE’s $O(T)$, it remains an efficient choice when one cannot trust the targets enough to use Supervised Stretch.

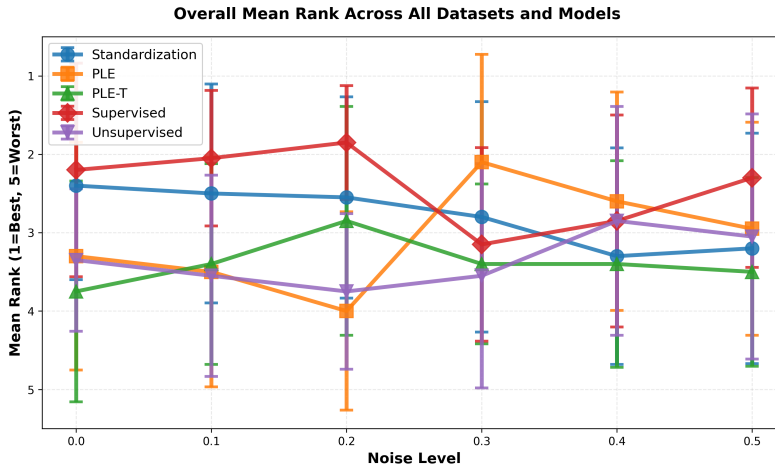


Figure 10: **Mean rank under label noise, aggregated over all (dataset, model) pairs.** Each method is ranked 1–5 on every (dataset, model) pair at every noise level, then averaged over the 20 combinations ($4 \text{ datasets} \times 5 \text{ models}$). Lower is better. Supervised Stretch (red) attains the best mean rank at $\eta = 0.5$. Error bars show the standard deviation of ranks across (dataset, model) pairs.

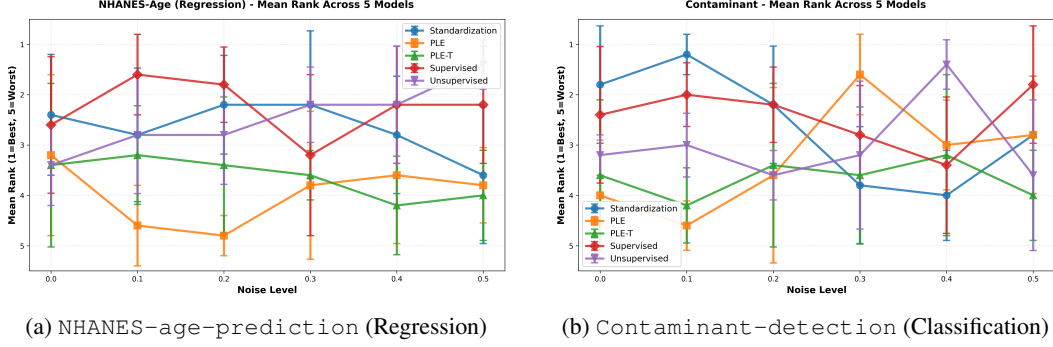


Figure 11: **Detailed Mean Rank Trajectories (1/2)**. Mean rank across 5 models for NHANES-age and Contaminant datasets under varying noise levels.

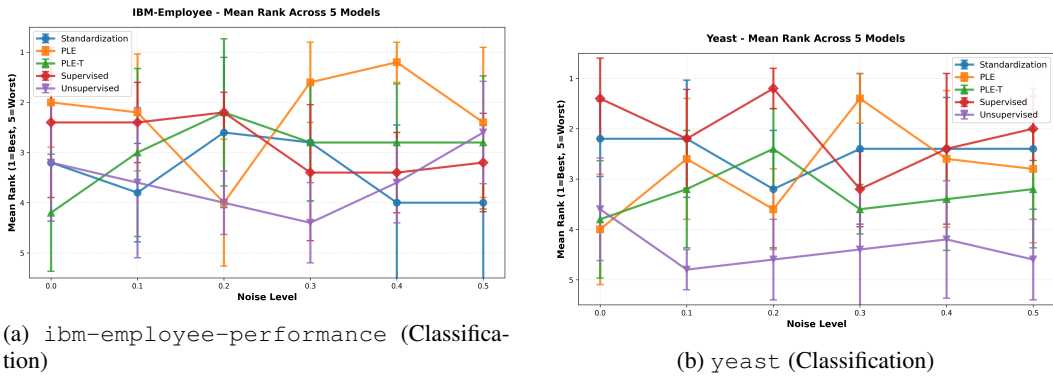


Figure 12: **Detailed Mean Rank Trajectories (2/2)**. Mean rank across 5 models for IBM-Employee and Yeast datasets under varying noise levels.

H Additional Metrics

For completeness, we report aggregated results across seven evaluation metrics: classification (Accuracy, Average Recall, F1, AUC) and regression (R^2 , MAE, RMSE). Each table summarises one metric across the five models and nine transformations, using the same tie-exclusion protocol as Table 1: per (dataset, model) pair, if the gap between the best and worst method is at most the median of the per-method seed standard deviations, that pair is treated as a tie and excluded from the per-cell average. The best displayed mean in each model row is shown in bold. The trailing “ $\pm$ ” is the standard error of the mean across the remaining datasets within the task category. For MAE and RMSE we caution that cross-dataset aggregation is not particularly meaningful, since these losses depend on each dataset’s target scale; we include the aggregated numbers as a coarse summary only.

Table 7: Aggregate Accuracy per model and transformation (higher is better); a counterpart to the ‘Avg. Acc’/‘Avg. R^2 ’ rows of Table 1. Each cell is the mean of the per-(dataset, model) values over the regression or classification benchmarks (with R^2 clamped at 0, matching the main paper). The trailing “ $\pm$ ” is the average per-pair standard error of the mean across 15 seeds, $\sigma_{\text{seed}}/\sqrt{15}$, averaged across the contributing datasets; cells whose underlying per-pair seed std exceeds 0.20 are flagged as “ $>.05$ ”. **Abbreviations:** Sup. (Stretch Supervised), Unsup. (Stretch Unsupervised), RS-SC (Robust Scale + Smooth Clip), YJ (Yeo–Johnson).

Model	Transformations								
	Sup.	Unsup.	Minmax	PLE	PLE-T	Quantile	RS-SC	Standard	YJ
fit	0.827 $\pm$.004	0.824 $\pm$.003	0.805 $\pm$.002	0.826 $\pm$.002	0.826 $\pm$.002	0.828 $\pm$.003	0.819 $\pm$.003	0.816 $\pm$.004	0.821 $\pm$.003
mlp	0.821 $\pm$.003	0.824 $\pm$.002	0.815 $\pm$.002	0.828 $\pm$.002	0.822 $\pm$.002	0.816 $\pm$.002	0.823 $\pm$.002	0.819 $\pm$.002	0.822 $\pm$.002
mlp_plr	0.826 $\pm$.001	0.826 $\pm$.002	0.821 $\pm$.002	0.821 $\pm$.002	0.821 $\pm$.003	0.827 $\pm$.002	0.823 $\pm$.002	0.821 $\pm$.002	0.821 $\pm$.003
realmlp	0.837 $\pm$.002	0.838 $\pm$.001	0.839 $\pm$.002	0.833 $\pm$.002	0.834 $\pm$.002	0.837 $\pm$.002	0.838 $\pm$.002	0.833 $\pm$.001	0.834 $\pm$.002
resnet	0.826 $\pm$.002	0.824 $\pm$.003	0.814 $\pm$.003	0.824 $\pm$.002	0.823 $\pm$.002	0.816 $\pm$.002	0.825 $\pm$.002	0.819 $\pm$.002	0.820 $\pm$.002

Table 8: Aggregate Avg. Recall per model and transformation (higher is better); a counterpart to the ‘Avg. Acc’/‘Avg. R^2 ’ rows of Table 1. Each cell is the mean of the per-(dataset, model) values over the regression or classification benchmarks (with R^2 clamped at 0, matching the main paper). The trailing “ $\pm$ ” is the average per-pair standard error of the mean across 15 seeds, $\sigma_{\text{seed}}/\sqrt{15}$, averaged across the contributing datasets; cells whose underlying per-pair seed std exceeds 0.20 are flagged as “ $>.05$ ”. **Abbreviations:** Sup. (Stretch Supervised), Unsup. (Stretch Unsupervised), RS-SC (Robust Scale + Smooth Clip), YJ (Yeo–Johnson).

Model	Transformations								
	Sup.	Unsup.	Minmax	PLE	PLE-T	Quantile	RS-SC	Standard	YJ
fit	0.737 $\pm$.007	0.740 $\pm$.004	0.711 $\pm$.004	0.744 $\pm$.003	0.744 $\pm$.004	0.746 $\pm$.005	0.728 $\pm$.005	0.728 $\pm$.006	0.737 $\pm$.005
mlp	0.733 $\pm$.005	0.738 $\pm$.004	0.723 $\pm$.003	0.734 $\pm$ $>.05$	0.739 $\pm$.003	0.728 $\pm$.004	0.740 $\pm$.004	0.728 $\pm$.004	0.733 $\pm$.003
mlp_plr	0.744 $\pm$.002	0.742 $\pm$.003	0.734 $\pm$.003	0.733 $\pm$.003	0.734 $\pm$.004	0.742 $\pm$.005	0.740 $\pm$.003	0.731 $\pm$.003	0.736 $\pm$.004
realmlp	0.762 $\pm$.003	0.760 $\pm$.003	0.759 $\pm$.003	0.754 $\pm$.003	0.757 $\pm$.003	0.759 $\pm$.003	0.759 $\pm$.003	0.747 $\pm$.003	0.751 $\pm$.003
resnet	0.746 $\pm$.003	0.740 $\pm$.005	0.728 $\pm$.005	0.738 $\pm$.003	0.740 $\pm$.003	0.729 $\pm$.004	0.744 $\pm$.004	0.732 $\pm$.003	0.734 $\pm$.004

Table 9: Aggregate F1 per model and transformation (higher is better); a counterpart to the ‘Avg. Acc’/‘Avg. R^2 ’ rows of Table 1. Each cell is the mean of the per-(dataset, model) values over the regression or classification benchmarks (with R^2 clamped at 0, matching the main paper). The trailing “ $\pm$ ” is the average per-pair standard error of the mean across 15 seeds, $\sigma_{\text{seed}}/\sqrt{15}$, averaged across the contributing datasets; cells whose underlying per-pair seed std exceeds 0.20 are flagged as “ $>.05$ ”. **Abbreviations:** Sup. (Stretch Supervised), Unsup. (Stretch Unsupervised), RS-SC (Robust Scale + Smooth Clip), YJ (Yeo–Johnson).

Model	Transformations								
	Sup.	Unsup.	Minmax	PLE	PLE-T	Quantile	RS-SC	Standard	YJ
fit	0.680 $\pm$ $>.05$	0.681 $\pm$ $>.05$	0.656 $\pm$.005	0.691 $\pm$.005	0.689 $\pm$.006	0.693 $\pm$.006	0.672 $\pm$.006	0.665 $\pm$ $>.05$	0.683 $\pm$ $>.05$
mlp	0.670 $\pm$ $>.05$	0.682 $\pm$.005	0.662 $\pm$.003	0.682 $\pm$ $>.05$	0.686 $\pm$.005	0.673 $\pm$.005	0.686 $\pm$.005	0.670 $\pm$.005	0.674 $\pm$.004
mlp_plr	0.693 $\pm$.003	0.692 $\pm$.004	0.676 $\pm$.004	0.684 $\pm$.004	0.680 $\pm$.005	0.688 $\pm$.005	0.690 $\pm$.004	0.675 $\pm$.004	0.683 $\pm$.005
realmlp	0.713 $\pm$.004	0.709 $\pm$.004	0.709 $\pm$.004	0.704 $\pm$.003	0.707 $\pm$.003	0.709 $\pm$.004	0.709 $\pm$.003	0.691 $\pm$.004	0.699 $\pm$.003
resnet	0.692 $\pm$.004	0.679 $\pm$ $>.05$	0.670 $\pm$.006	0.686 $\pm$.004	0.687 $\pm$.004	0.670 $\pm$.006	0.693 $\pm$.006	0.670 $\pm$.004	0.677 $\pm$.006

Table 10: Aggregate AUC per model and transformation (higher is better); a counterpart to the ‘Avg. Acc’/‘Avg. R^2 ’ rows of Table 1. Each cell is the mean of the per-(dataset, model) values over the regression or classification benchmarks (with R^2 clamped at 0, matching the main paper). The trailing “ $\pm$ ” is the average per-pair standard error of the mean across 15 seeds, $\sigma_{\text{seed}}/\sqrt{15}$, averaged across the contributing datasets; cells whose underlying per-pair seed std exceeds 0.20 are flagged as “ $>.05$ ”. **Abbreviations:** Sup. (Stretch Supervised), Unsup. (Stretch Unsupervised), RS-SC (Robust Scale + Smooth Clip), YJ (Yeo–Johnson).

Model	Transformations								
	Sup.	Unsup.	Minmax	PLE	PLE-T	Quantile	RS-SC	Standard	YJ
fit	0.868 $\pm$.007	0.866 $\pm$.004	0.834 $\pm$.005	0.872 $\pm$.002	0.869 $\pm$.004	0.872 $\pm$.004	0.862 $\pm$.004	0.859 $\pm$.005	0.862 $\pm$.005
mlp	0.861 $\pm$.004	0.871 $\pm$.002	0.855 $\pm$.002	0.866 $\pm$.003	0.861 $\pm$.004	0.868 $\pm$.002	0.866 $\pm$.003	0.865 $\pm$.002	0.867 $\pm$.002
mlp_plr	0.863 $\pm$.002	0.863 $\pm$.003	0.866 $\pm$.002	0.856 $\pm$.002	0.860 $\pm$.003	0.867 $\pm$.003	0.863 $\pm$.003	0.859 $\pm$.003	0.862 $\pm$.003
realmlp	0.867 $\pm$.003	0.871 $\pm$.003	0.875 $\pm$.002	0.863 $\pm$.003	0.872 $\pm$.003	0.873 $\pm$.003	0.871 $\pm$.003	0.873 $\pm$.003	0.872 $\pm$.003
resnet	0.872 $\pm$.003	0.863 $\pm$ $>.05$	0.858 $\pm$.005	0.869 $\pm$.002	0.864 $\pm$.003	0.872 $\pm$.003	0.872 $\pm$.005	0.869 $\pm$.003	0.872 $\pm$.003

Table 11: Aggregate R^2 per model and transformation (higher is better); a counterpart to the ‘Avg. Acc’/‘Avg. R^2 ’ rows of Table 1. Each cell is the mean of the per-(dataset, model) values over the regression or classification benchmarks (with R^2 clamped at 0, matching the main paper). The trailing “ $\pm$ ” is the average per-pair standard error of the mean across 15 seeds, $\sigma_{\text{seed}}/\sqrt{15}$, averaged across the contributing datasets; cells whose underlying per-pair seed std exceeds 0.20 are flagged as “ $> .05$ ”. **Abbreviations:** Sup. (Stretch Supervised), Unsup. (Stretch Unsupervised), RS-SC (Robust Scale + Smooth Clip), YJ (Yeo–Johnson).

Model	Transformations								
	Sup.	Unsup.	Minmax	PLE	PLE-T	Quantile	RS-SC	Standard	YJ
ftt	0.691 $\pm$.002	0.681 $\pm$.002	0.666 $\pm$.003	0.680 $\pm$.006	0.667 $\pm$.003	0.681 $\pm$.002	0.673 $\pm$.003	0.681 $\pm$.001	0.667 $\pm$.001
mlp	0.630 $\pm$.005	0.606 $\pm$.007	0.628 $\pm$.008	0.608 $\pm$ $> .05$	0.588 $\pm$ $> .05$	0.619 $\pm$.005	0.619 $\pm$.005	0.591 $\pm$ $> .05$	0.545 $\pm$ $> .05$
mlp_plr	0.698 $\pm$.004	0.696 $\pm$.004	0.659 $\pm$.004	0.686 $\pm$.002	0.702 $\pm$.002	0.669 $\pm$.005	0.681 $\pm$.004	0.675 $\pm$.003	0.649 $\pm$.005
realmlp	0.736 $\pm$.004	0.685 $\pm$.004	0.679 $\pm$.003	0.736 $\pm$.003	0.723 $\pm$.003	0.707 $\pm$.005	0.689 $\pm$.004	0.686 $\pm$.002	0.683 $\pm$ $> .05$
resnet	0.651 $\pm$.003	0.631 $\pm$.006	0.655 $\pm$.002	0.639 $\pm$ $> .05$	0.631 $\pm$.005	0.625 $\pm$ $> .05$	0.640 $\pm$.002	0.641 $\pm$.003	0.542 $\pm$ $> .05$

Table 12: Aggregate MAE per model and transformation (lower is better); a counterpart to the ‘Avg. Acc’/‘Avg. R^2 ’ rows of Table 1. Each cell is the mean of the per-(dataset, model) values over the regression benchmarks. Note: MAE and RMSE are tied to each dataset’s target scale, so the cross-dataset average is only a coarse summary and is not directly comparable across rows; we include it for completeness only and omit a $\pm$ uncertainty. **Abbreviations:** Sup. (Stretch Supervised), Unsup. (Stretch Unsupervised), RS-SC (Robust Scale + Smooth Clip), YJ (Yeo–Johnson).

Model	Transformations								
	Sup.	Unsup.	Minmax	PLE	PLE-T	Quantile	RS-SC	Standard	YJ
ftt	30.84	30.80	31.27	30.41	30.27	31.10	30.90	30.73	31.62
mlp	36.42	37.19	36.91	38.50	37.25	36.97	36.03	37.88	36.29
mlp_plr	33.06	32.96	32.58	34.91	33.96	34.07	33.16	33.59	33.47
realmlp	30.10	30.31	30.79	30.09	30.33	30.28	30.25	30.33	30.79
resnet	35.16	35.32	35.67	34.71	34.56	36.34	34.26	35.47	37.42

Table 13: Aggregate RMSE per model and transformation (lower is better); a counterpart to the ‘Avg. Acc’/‘Avg. R^2 ’ rows of Table 1. Each cell is the mean of the per-(dataset, model) values over the regression benchmarks. Note: MAE and RMSE are tied to each dataset’s target scale, so the cross-dataset average is only a coarse summary and is not directly comparable across rows; we include it for completeness only and omit a $\pm$ uncertainty. **Abbreviations:** Sup. (Stretch Supervised), Unsup. (Stretch Unsupervised), RS-SC (Robust Scale + Smooth Clip), YJ (Yeo–Johnson).

Model	Transformations								
	Sup.	Unsup.	Minmax	PLE	PLE-T	Quantile	RS-SC	Standard	YJ
ftt	57.11	57.28	57.97	56.97	56.45	57.80	57.40	57.13	59.47
mlp	67.4	128.2	74.7	123.9	100.6	69.5	66.1	204.7	114.6
mlp_plr	61.25	60.00	59.69	63.26	62.58	62.27	61.70	61.19	62.85
realmlp	55.34	56.84	56.41	56.63	56.93	56.43	55.87	56.85	57.58
resnet	62.1	94.9	74.8	113.0	113.2	65.5	62.1	79.2	177.2