Weighted Pairwise Difference Learning

Mohamed Karim Belaid[1,2,3], Tim Wibiral[2,4], Maximilian Rabus[2] and Eyke Hüllermeier[3]

[1]IDIADA Fahrzeugtechnik GmbH, Munich, Germany
[2]Dr. Ing. h.c. F. Porsche AG Stuttgart, Germany
[3]Ludwig-Maximilians-Universität Munich, Germany
[4]Universität Ulm, Germany

Abstract

Pairwise Difference Learning (PDL) has recently been proposed as a new meta-learning technique for regression and classification tasks. Instead of learning the sought mapping from instances to outcomes directly, PDL learns a function that predicts the difference between the outcomes of any pair of instances given as input. To generate predictions for a new query, the method averages over predicted differences from the outcomes of so-called anchor points. This paper enhances the PDL regressor by incorporating techniques for anchor weighting, i.e., modulating the influence of anchor points according to the reliability and precision of their predictions. Our extensive empirical analysis demonstrates highly competitive performance of the weighted PDL regressor. Furthermore, we provide an accessible and publicly available implementation of weighted PDL in a Python package.

Keywords

Regression, pairwise difference learning, instance weighting

1. Introduction

Pairwise Difference Learning (PDL) was independently proposed by Tynes et al. [1] and Wetzel et al. [2] as a meta-learning framework for regression. Instead of learning a regression function f directly, i.e., a mapping from instances to outcomes, PDL learns a function that predicts the *difference* between the outcomes of any pair of instances given as input. The principle behind PDL is to rewrite f at a point x relative to another point (the anchor) x': $f(x) = f(x') + \Delta(x, x')$, where $\Delta(x, x') = f(x) - f(x')$. PDL aims to train an approximation $\tilde{\Delta}$ of this difference function Δ, allowing for new predictions $y = f(x)$ by averaging the predicted differences with respect to the training data $(x_1, y_1), \ldots, (x_N, y_N)$:

$$y \approx \frac{1}{N} \sum_{i=1}^{N} y_i + \tilde{\Delta}(x, x_i) \tag{1}$$

An important motivation of PDL is the quadratic increase in training data, which makes the method appealing in the small data regime, even if the data loses the property of independence. Moreover, as PDL combines model-based learning (of the difference function Δ) with instance-based learning (accumulating evidence from different anchors), the prediction (1) benefits from a statistical averaging effect.

In (1), each training point x_i contributes equally to the prediction. To enhance performance, one may think of applying a weighting scheme where each anchor x_i is assigned a weight w_i (such that the weights are non-negative and sum to one):

$$y \approx \sum_{i=1}^{N} w_i \cdot (y_i + \tilde{\Delta}(x, x_i)) \tag{2}$$

This approach is in line with the general idea of *instance weighting* in machine learning [3], which is most commonly used to increase the (relative) influence of training examples that are representative and

LWDA'24: Lernen, Wissen, Daten, Analysen. September 23–25, 2024, Würzburg, Germany
karim.belaid@idiada.com (M. K. Belaid); tim.wibiral@uni-ulm.de (T. Wibiral); maximilian.rabus2@porsche.de (M. Rabus); eyke@if.lmu.de (E. Hüllermeier)
0000-0001-9038-9045 (M. K. Belaid); 0000-0001-5742-1795 (T. Wibiral); 0000-0003-0755-1772 (M. Rabus); 0000-0002-9944-4108 (E. Hüllermeier)

reliable. In contrast, examples that are not representative or reliable — such as noise and outliers — may bias the learner in the wrong direction and are given less weight. In other words, the weighting is supposed to reflect the "competence" of an anchor as a local predictor, thereby enhancing the overall prediction performance. However, as will be seen later on, the additional flexibility through weighting can also have other advantages, such as speeding up predictions when some weights are zero. Moreover, although this paper focuses on regression, the weighting strategy can be extended to other prediction tasks as well.

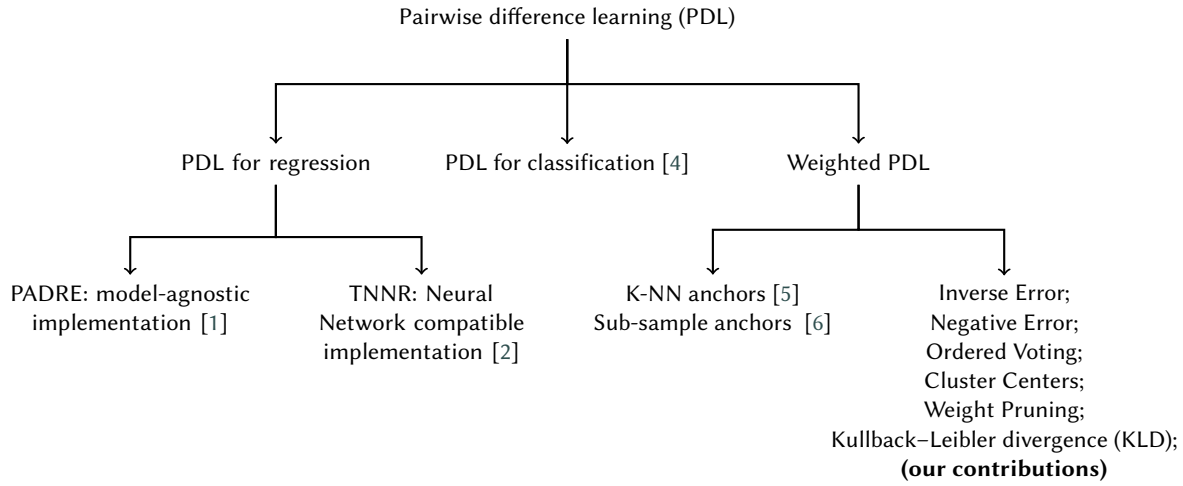

Figure 1: Contributions to PDL

Expanding on the PDL framework, our contributions are as follows (cf. Fig. 1):

- We introduce several innovative weighting algorithms to enhance the weighted PDL framework, improving prediction accuracy by refining anchor contributions (Section 3).
- We expand the "pairwise difference learning library" (pdll) on PyPI to include our weighted PDL implementation, ensuring compatibility with any scikit-learn model (Section 3.4).
- We perform a comprehensive experimental evaluation, comparing the performance of the PDL Regressor to leading ML methods (Section 4).

2. Related Work

Tynes et al. [1] pioneered the Pairwise Difference Regressor, an innovative meta-learner designed for chemical tasks. This method not only outperforms traditional random forest models in prediction accuracy but also delivers robust uncertainty quantification, proving particularly beneficial in computational chemistry where estimating differences between data points can help correct systematic errors.

Simultaneously, Wetzel et al. [2] developed a similar approach using Twin Neural Network architectures tailored for supervised and semi-supervised regression tasks. Their method predicts the differences between target values of distinct data points and can leverage unlabeled data by pairing it with labeled anchor data points. By ensembling the predicted differences, Wetzel et al.'s technique achieved high performance in regression problems, albeit specialized for neural network models [2].

The literature on PDL has since diversified into various methodologies, as depicted in Fig. 1. Spiers et al. [7] extended PDL to measure sample similarity in chemistry, emphasizing differences in spectral shapes using Euclidean and Mahalanobis distances, and introduced a Z-score metric for outlier detection and model adaptation. Jiménez-Luna et al. [8] applied twin neural networks, termed DeltaDelta, to rank molecular potency.

PDL's applicability extends beyond standard regression tasks. It can be adapted for targets that are known, bounded, or defined by inequalities (e.g., $y = 5.3$, $y < 2.1$, or $y > 6.5$) by predicting

relative changes between pairs [9]. Belaid et al. [4] adapted PDL for binary and multiclass classification, demonstrating significant accuracy improvements across various benchmark datasets and highlighting the theoretical underpinnings of PDL's performance enhancements.

The PDL regressor and its variants have shown efficacy in diverse applications, including image-based regression [10], chemical property prediction [11], quantum mechanical reaction modeling [12], drug activity ranking [13], and approximating prudent response surface simulations [14].

In its typical implementation, PDL treats all training samples as equally important anchors. Recent literature suggests methods to reduce complexity, such as random subsampling of k anchors or selecting the k nearest anchors [5, 6]. Nevertheless, the original PDL method, which benefits from the statistical advantages of averaging, remains superior [15].

In the field of data valuation, various techniques have been developed to assess the significance of training samples. One approach, hard negative mining [16], prioritizes samples with higher losses, thus enhancing model training efficiency. Kullback–Leibler Divergence (KLD) promotes higher entropy in weight distributions, effectively penalizing peak values and improving performance across diverse areas such as image classification and language modeling [17]. Another notable technique is stacked generalization [18], an ensembling method where a learner makes final predictions based on the outputs of multiple weak learners. This method can utilize a linear optimizer, leveraging the L_1 and L_2 losses on a validation set, to select a subset of weak learners, thereby enhancing predictive performance [19].

3. Weighted PDL

Consider a standard setting of supervised (regression) learning: Given a set of training data

$$\mathscr{D} = \{(x_i, y_i)\}_{i=1}^{N} \subset \mathbb{R}^d \times \mathbb{R}\,,$$

comprised of training instances in the form of feature vectors $x_i \in \mathbb{R}^d$ together with observed outcomes $y \in \mathbb{R}$, and assumed to be generated i.i.d. according to an underlying (unknown) joint probability measure P, the task is to learn a predictor $\mathrm{PDR} : \mathbb{R}^d \to \mathbb{R}$ with low risk (expected loss) $R(h) = \mathbf{E}_{(X,Y)\sim P} L(\mathrm{PDR}(X), Y)$ according to a loss function $L : \mathbb{R}^2 \to \mathbb{R}$.

PDL Regressor transforms the original training data $\mathscr{D}$ into the new data

$$\mathscr{D}_{pair} = \left\{(z_{i,j}, y_{i,j}) \,|\, 1 \le i, j \le N\right\}, \tag{3}$$

where $y_{i,j}$ denotes the difference $y_i - y_j$ and $z_{i,j} = \phi(x_i, x_j)$ is a joint feature representation of x_i and x_j. The simplest representation is a concatenation of x_i and x_j. Subsequent research on PDL revealed that incorporating the difference vector $x_i - x_j$ into this representation significantly boosts performance [1, 4]. Consequently, our work also employs this enriched feature representation.

A regressor h is trained on $\mathscr{D}_{pair}$ in a conventional manner. For any pair of input vectors, x and x', the role of this function is to predict the difference $y - y'$ between their respective outcomes y and y'. We thus obtain the predicted difference function as $\tilde{\Delta}(x_i, x_j) = h(\phi(x_i, x_j))$.

The pairwise difference predictor $\tilde{\Delta}$ must adhere to the property of antisymmetry:

$$\tilde{\Delta}(x, x') = -\tilde{\Delta}(x', x)\,, \tag{4}$$

for all $x, x' \in \mathbb{R}^d$. However, this property is not guaranteed when learning $\tilde{\Delta}$ from $\mathscr{D}_{pair}$, even if it holds for the training data itself. We demonstrate the presence of this limitation in Annex B by quantifying the error of learning the antisymmetry property using empirical experiments. To address this issue, one can either restrict the class of regression functions to antisymmetric ones or incorporate antisymmetry as a constraint in the learning process. To circumvent any restriction of the model class or complication of the learning task, we adopt a straightforward solution: we "antisymmetrize" a predictor as follows:

$$\tilde{\Delta}_{sym}(x, x') = \frac{\tilde{\Delta}(x, x') - \tilde{\Delta}(x', x)}{2}\,. \tag{5}$$

This solution was previously used with twin neural networks [2] and PDL Classifier [4].

At prediction time, given a query instance x_q, the outcome y_q can be predicted using each data point (x_i, y_i) from the original training data $\mathscr{D}$. Given that $y_q = y_i + (y_q - y_i)$, and the difference $y_q - y_i$ is estimated by $\tilde{\Delta}_{sym}(x_q, x_i)$, the predicted value $\hat{y}_q$ is formulated as $\hat{y}_q = y_i + \tilde{\Delta}_{sym}(x_q, x_i)$. Instead of relying on a single prediction of this nature, PDL Regressor leverages all available anchors

$$\hat{y}_q = \mathrm{PDR}(x_q) = \sum_{i=1}^{N} w_i \cdot (y_i + \tilde{\Delta}_{sym}(x_q, x_i)) \,, \tag{6}$$

such that $\sum_{i=1}^{N} w_i = 1$. In its basic version, PDL assigns the same importance to all training examples ($w_i = \frac{1}{N}$) [1, 4], although some of them might be outliers, redundant, or mislabeled. As a more general approach, we propose to optionally weight the anchor points. In our approach, these weights are learned using a validation set $\mathscr{D}_{val}$. We present the method in the general case where we are given k anchors $(x_1^a, y_1^a), \ldots, (x_k^a, y_k^a)$ that may differ from the N training points. We propose several approaches to weighting the anchors. For each, we will discuss the primary intuition, the calculation method, and the complexity. It is important to note that certain methods do not generate weights that sum up to one, requiring an adjustment of the vector $w = (w_1, \ldots, w_k)$.

$$w_i \leftarrow \frac{w_i}{\sum_j w_j} \,. \tag{7}$$

We categorize the proposed approaches into three main types: optimization-based methods, heuristic methods, and unsupervised learning methods.

3.1. Optimization Methods

We learn the weight vector $w = (w_1, \ldots, w_k)$ for the k anchors by solving a constrained optimization problem. This involves minimizing the loss of the weighted predictions on the validation data, subject to the constraints that all weights are non-negative and sum to 1. For regression, the weighted prediction is defined by (6).

The loss function, representing the mean absolute error (MAE), is given by:

$$L(w) = \frac{1}{|\mathscr{D}_{val}|} \sum_{(x_q, y_q) \in \mathscr{D}_{val}} \left| y_q - \hat{y}_q(w) \right| \,, \tag{8}$$

Without Regularization In this approach, we focus on a straightforward objective function that minimizes the validation loss:

$$w^{\mathrm{opt}} = \arg \min_{w \in [0,1]^k} L(w) \quad \textbf{s.t.} \quad \|w\|_1 = 1 \,. \tag{9}$$

Using Kullback–Leibler Divergence (KLD) To mitigate overfitting, we minimize a regularized loss that promotes a uniform weight distribution. This prevents the assignment of excessive weight to a single or few anchors. The objective function is defined as follows:

$$w^{\mathrm{opt}} = \arg \min_{w \in [0,1]^k} L(w) + \lambda \cdot L(u) \cdot \mathrm{KL}(w \parallel u) \quad \textbf{s.t.} \quad \|w\|_1 = 1 \,, \tag{10}$$

where $\mathrm{KL}(w \parallel u)$ represents the Kullback–Leibler divergence (KLD) between the weight vector $w = (w_1, \ldots, w_k)$ and the uniform distribution[1] $u = (1/k, \ldots, 1/k)$, with λ as the regularization parameter relative to the initial validation loss $L(u)$.[2] The KLD term penalizes highly concentrated weight distributions, thus encouraging the optimizer to maintain a higher entropy in its weight distribution.

[1] Which can be simplified to $\log(k)$ minus the entropy of w.

[2] For our experiments, we set λ to 0.05.

Using Weight Pruning Unlike the previous method that promotes a uniform weight distribution, this method focuses on enhancing sparsity and reducing the magnitudes of weights. The L_1 norm, denoted as $\|w\|_1$, is constrained to 1 in the optimization. Thus, the aim is to minimize the sum of the $k-1$ smallest weights. The L_1 norm to minimize can be written as

$$\begin{aligned}\arg\min_{w\in[0,1]^k} L_1(w) &= \arg\min_{w\in[0,1]^k} \|w\|_1 - w_{\max} \\ &= \arg\min_{w\in[0,1]^k} 1 - w_{\max} \\ &= \arg\min_{w\in[0,1]^k} -w_{\max}\end{aligned}$$

where $w_{\max}$ is the largest weight in w. The objective function is defined as follows:

$$w^{\text{opt}} = \arg\min_{w\in[0,1]^k} L(w) - \lambda_1 \cdot L(u) \cdot w_{\max} + \lambda_2 \cdot L(u) \cdot \|w\|_2^2 \quad \textbf{s.t.} \quad \|w\|_1 = 1\,. \tag{11}$$

where λ_1 and λ_2 parameterize the trade-off between the MAE loss $L(w)$ and the minimization of the weights. $L(u)$ allow having the same scale of magnitude as $L(w)$. We experiment with three parameter settings: pure L_1 loss $(\lambda_1, \lambda_2) = (0.1, 0)$, pure L_2 loss $(\lambda_1, \lambda_2) = (0, 0.1)$, and a combination of L_1 and L_2 loss $(\lambda_1, \lambda_2) = (0.05, 0.025)$. When anchors are highly correlated, such as those from the same cluster, L_1 loss tends to select only one anchor. In contrast, a mix of L_1 and L_2 regularizations encourages the selection of grouped anchors, distributing the weights among them.

Using Extreme Weight Pruning To further investigate the impact of significantly reducing the anchor set, we conduct experiments with $(\lambda_1, \lambda_2) = (0.8, 0)$ to evaluate the performance when a large subset of weights is set to zero. If this approach results in more than 90% of null weights, we recursively adjust $\lambda_1 \leftarrow \lambda_1 \cdot 0.5$ and repeat the experiment. The process continues until λ_1 falls below 0.0001.

To solve the objective functions (9, 10, 11), we employ the Sequential Least Squares Programming (SLSQP) method, which efficiently handles the constraints, such as $\|w\|_1 = 1$. However, the SLSQP optimizer introduces an additional computational cost of $\mathcal{O}(k^3)$ [20].

3.2. Heuristic Methods

Inverse Error The Inverse Error method is based on the intuition that anchors yielding lower validation errors should be assigned higher weights, as they are likely more reliable predictors. We use the mean absolute error (MAE) to quantify the validation error associated with the anchor (xi, yi):

$$e_i = \frac{1}{|\mathcal{D}_{val}|} \sum_{(x_q, y_q)\in\mathcal{D}_{val}} \left|(y_i + \tilde{\Delta}_{sym}(x_q, x_i)) - y_q\right|\,. \tag{12}$$

The anchor weights are computed as the inverse of these errors:

$$w_i = \frac{1}{e_i + \epsilon}\,.$$

Here, ϵ is a small constant (e.g., $\epsilon = 0.0001$) added to prevent division by zero and ensure numerical stability. These weights are then normalized using (7). This method directly links anchor weights to their individual performance, assigning greater influence to better-performing anchors. Its primary advantage lies in the independent estimation of error e_i from the validation set, resulting in a straightforward and computationally efficient approach. However, it does not account for the potential mutual error cancellation effects that might arise from ensembling multiple anchors. This method operates with a time complexity of $\mathcal{O}(k \cdot |\mathcal{D}_{val}|)$.

Negative Error The Negative Error approach assigns weights to anchors inversely related to their anchor-wise error, as calculated in (12). Specifically, the weight is determined by subtracting the error from the maximum error across all anchors: $w_i = \max(e_1, \ldots, e_k) - e_i$. These weights are then normalized according to (7). This method operates with a time complexity of $\mathcal{O}(k \cdot |\mathcal{D}_{val}|)$.

Ordered Voting The Ordered Voting method ranks anchors based on their errors e_i from (12) in ascending order. Each anchor is assigned a rank $r_i \in \{1, \dots, k\}$, where the anchor with the lowest error receives rank 1 and the one with the highest error receives rank k. Weights are then assigned based on these ranks:

$$w_i = \frac{k - r_i + 1}{k(k+1)/2}.$$

The computational cost of this method equals the cost of the negative error method plus the complexity of sorting the anchors, yielding a total time complexity of $\mathcal{O}(k \cdot |\mathcal{D}_{val}| + k \cdot \log(k))$.

3.3. Unsupervised Method: K-means Cluster Centers

To significantly reduce the number of anchors and accelerate the prediction process, we leverage K-means clustering to identify prototypical cluster centers within the training set. The method searches for a number of cluster centers equal to 10% of the training samples or at least 3. Then, it retains only the anchors closest to these centers. This approach ensures that the retained anchors are highly representative of the dataset, thereby reducing the anchor set size without sacrificing representativeness. K-means operates with a time complexity of $\mathcal{O}(k^2 \cdot d \cdot t)$ given d features and t iterations.

In a similar vein, Wetzel proposed selecting the k-nearest neighboring anchors to each unseen test point [5]. Although this method dynamically selects anchors and requires the entire anchor set in memory, our K-means clustering approach permanently reduces the anchor set, leading to memory savings.

3.4. PDL Library

Our library[3] offers a Python implementation of the weighted PDL, conforming to Scikit-learn standards. This compatibility ensures seamless integration into existing codebases with minimal adjustments. The following example demonstrates how to incorporate the weighted PDL with just one additional line of code:

```
!pip install pdll
from pdll import PairwiseDifferenceRegressor
X_train, X_val, y_train, y_val = load_data()
model = RandomForestRegressor()
model = PairwiseDifferenceRegressor(model)
model.fit(X, y)
model.learn_anchor_weights(X_val, y_val, method='L2')  # Added line
model.predict(...)
```

4. Evaluation

To benchmark the proposed weighted PDL methods, we use a variety of public datasets from OpenML [21] and state-of-the-art Scikit-learn baseline learners [22].

4.1. Data

OpenML offers a diverse range of datasets, with 61% containing fewer than 2000 data points. This study focuses on these smaller datasets, where the pairwise learning approach is presumably most effective. Adhering to dataset selection constraints similar to the OpenML-CC18 benchmark [23], we randomly

[3]Link: https://github.com/Karim-53/pdll

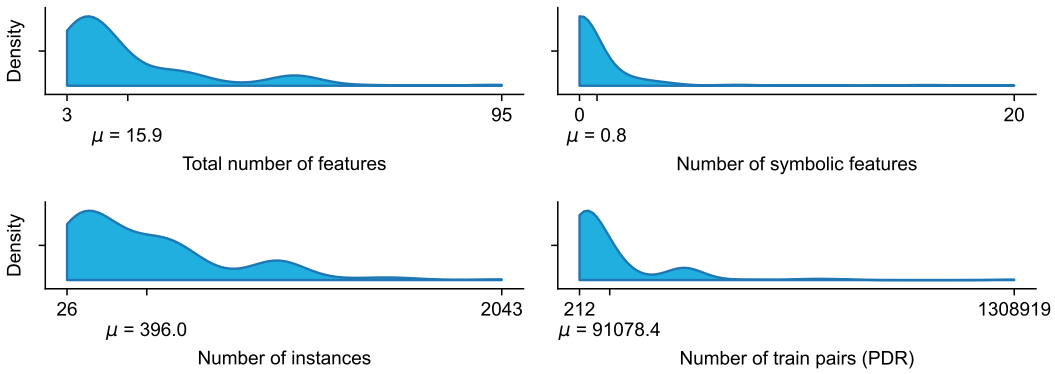

Figure 2: Distribution of key characteristics of the 231 OpenML regression datasets used for evaluation, specifying the minimum, mean, and maximum. On average, the datasets contain around 400 data points with around 16 features. Most features are numeric.

selected 231 datasets (summary statistics in Fig. 2). Despite their small size, the effective data volume for PDL increases quadratically due to pairing, resulting in over $1,000,000$ pairs of training data points.

We evaluated the methods using 7 baseline models, conducting 5 iterations of 5-fold cross-validation on each of the 231 datasets, following a similar approach to a previous benchmark study on PDL for classification [4]. This resulted in a total of $80,850$ train-test runs, consuming one week of wall-time on a high-performance computing cluster. The validation set comprised 30% of the training data.

4.2. Data Processing Pipeline

We implemented a standardized data processing pipeline for all runs using scikit-learn [22]. This pipeline includes standardization for numeric features, one-hot encoding for nominal features, and ordinal encoding for ordinal features. Since PDL requires the pair difference $x_i - x_j$ as additional inputs, all processed features are treated as numeric when computing these differences.

4.3. Performance Measures

We evaluate performance using the mean absolute error (MAE) and the normalized MAE (NMAE) for regression tasks:

$$\text{NMAE} = \frac{\text{MAE}}{\max(y_{\text{truth}}) - \min(y_{\text{truth}})}.$$

To compare models, we count the number of wins and losses based on the average performance over 25 runs (5 times 5-fold cross-validation) per dataset. A win is counted when PDR's average performance exceeds that of the baseline model, and a loss is counted otherwise. Similarly, when comparing PDR to weighted PDR using L2, a win is counted if the weighted PDR using L2 outperforms the PDR.

To determine significant wins and losses, we perform a Student's t-test for each dataset to assess the statistical significance of performance differences. A win or loss is considered significant if the p-value is below a predetermined threshold ($\alpha = 0.05$). In cases where average scores tie, the total number of wins and losses may not sum to 231, which is the total number of datasets benchmarked.

As an alternative to counting wins and losses, we also consider the average performance across all datasets $\pm$ the standard error. While this statistic is meant to provide a first impression, it should be interpreted with caution, because averaging over different datasets is theoretically questionable.

4.4. Results

This section compares PDL with baseline ML methods, evaluates the proposed weighted PDL methods, and finally contrasts the best weighted PDL with standard PDL.

Table 1
Comparing 7 baseline regressors and the PDR meta-algorithm, averaging over 231 datasets.

	Significant wins		Wins		MAE		Normed MAE	
Estimator	Base	PDR	Base	PDR	Base ±sem	PDR ±sem	Base ±sem	PDR ±sem
Bagging	21	**135**	60	**171**	0.4376 ±0.0033	**0.4115** ±0.0036	0.0881 ±0.0008	**0.0827** ±0.0008
DecisionTree	0	**203**	6	**225**	0.5413 ±0.0042	**0.4438** ±0.0040	0.1086 ±0.0010	**0.0891** ±0.0009
ExtraTree	1	**204**	7	**224**	0.5700 ±0.0043	**0.4120** ±0.0036	0.1144 ±0.0010	**0.0829** ±0.0008
ExtraTrees	8	**150**	33	**198**	0.4066 ±0.0034	**0.3872** ±0.0034	0.0818 ±0.0008	**0.0779** ±0.0008
GradientBoosting	38	**131**	61	**170**	0.4159 ±0.0035	**0.3994** ±0.0034	0.0840 ±0.0008	**0.0806** ±0.0008
HistGradBoosting	19	**170**	38	**193**	0.4779 ±0.0036	**0.3940** ±0.0037	0.0985 ±0.0009	**0.0794** ±0.0008
RandomForest	49	**108**	95	**136**	0.4192 ±0.0032	**0.4102** ±0.0036	0.0845 ±0.0008	**0.0825** ±0.0008

Does PDL systematically enhance regression task performance? Previous studies utilizing the PDL regressor have been limited to datasets primarily within the chemistry field. Our comprehensive evaluation attempts to confirm the added value of this meta-algorithm across a broader range of datasets. Analyzing 231 datasets, our results indicate variable improvements across different base learners, as detailed in Tab. 1. On average, PDL consistently outperforms the base learners. In 89% of the cross-validation runs, the PDL regressor achieves a strictly better test MAE. Among the baseline models, the Extra Trees Regressor recorded the best average performance, with PDR(Extra Trees Regressor) achieving the highest performance overall among all baseline and PDR models. These findings collectively highlight PDL's effectiveness in enhancing performance independently of the base learner, with the weighted variant offering a valuable alternative through its unique characteristics. Given the excellent performance of PDL, a natural question arises: why does PDL outperform all the studied baselines? Previous work has already proposed some hypotheses [1], conducted empirical tests [2], and provided theoretical explanations for how each component of PDL contributes to its superior performance [4]. Therefore, we will limit this discussion to a general overview, allowing us to better focus on the weighted methods.

Which weighted PDL method performs best? The critical difference (CD) diagram in Fig. 3 visually compares the performance of ten proposed weighting methods based on their MAE across 231 datasets. The models are ranked from left to right, with lower ranks indicating better performance. The diagram employs the Wilcoxon signed-rank test to statistically compare each pair of algorithms and applies Holm's method to control for the family-wise error rate. The extensive dataset and cross-validation runs confirm the robustness of the results, although not all methods show significant differences. While the pure L2 weighting method achieves the highest average ranking (4.5), it only secures the first rank in 58% of the run. This indicates that other methods may outperform L2, depending on the dataset.

Other methods find the optimal weights less frequently but offer distinct advantages. For instance, heuristic methods can accelerate the weighting process thanks to their lower time complexity and suitability for online algorithms, as they would require the execution of the algorithm only on the added or removed anchors. The normalization of the weights can also be done in an online manner. On the

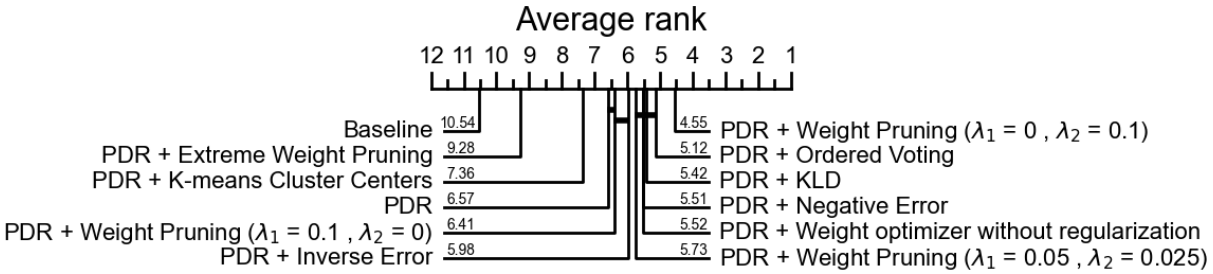

Figure 3: Critical difference diagram comparing the rank of ten Weighted PDL methods using the average ranking of the test MAE across 231 datasets. Lower numbers indicate a better average rank.

other hand, L_1 and L_2-based methods, along with the K-means method, enhance the prediction speed by reducing the number of anchors used, albeit with a potential increase in overfitting compared to the KLD method.

Table 2
Comparing PDR and weighted PDR using L2. Averaging over 231 datasets.

	Significant wins		Wins		MAE		Normed MAE	
Estimator	PDR	L2	PDR	L2	PDR ±sem	L2 ±sem	PDR ±sem	L2 ±sem
Bagging	1	**108**	45	**185**	0.4115 ±0.0036	**0.4026** ±0.0035	0.0827 ±0.0008	**0.0811** ±0.0008
DecisionTree	27	**64**	88	**142**	0.4438 ±0.0040	**0.4367** ±0.0039	0.0891 ±0.0009	**0.0879** ±0.0009
ExtraTree	**36**	35	**123**	108	0.4120 ±0.0036	0.4131 ±0.0037	0.0829 ±0.0008	0.0833 ±0.0009
ExtraTrees	9	**96**	55	**176**	0.3872 ±0.0034	**0.3825** ±0.0035	0.0779 ±0.0008	**0.0770** ±0.0008
GradientBoosting	13	**88**	66	**164**	0.3994 ±0.0034	**0.3946** ±0.0034	0.0806 ±0.0008	**0.0797** ±0.0008
HistGradBoosting	2	**115**	35	**196**	0.3940 ±0.0037	**0.3861** ±0.0036	0.0794 ±0.0008	**0.0780** ±0.0008
RandomForest	4	**115**	43	**188**	0.4102 ±0.0036	**0.4003** ±0.0035	0.0825 ±0.0008	**0.0807** ±0.0008

Does the extension to weighted PDL Regressor using L2 enhance performance? In our comprehensive empirical study comparing original PDR and weighted PDR using L2 regularization across 231 datasets, the results indicate a clear performance advantage of the weighted version, see Tab. 2. The only exception is PDR based on Extra Tree. However, the most significant finding is the ability to improve on top of Extra Trees, which has the best performance among PDRs. Although the improvement over the MAE seems small, it is statistically significant (as determined by the t-test for significant wins), and it allows us to reach a new top performance not reached by any baseline or PDR model.

5. Conclusion

Building on the basic idea of pairwise difference learning, we proposed instance (anchor) weighting as an extension of this meta-learning method for regression. To determine suitable weights, we proposed approaches based on simple heuristics, linear optimizers, as well as unsupervised learning algorithms. In a large-scale empirical evaluation of weighted PDL regression, we confirmed its potential and showed that it consistently outperforms the baseline models and PDR models. Each weighting method offers specific advantages: faster computation complexity, better performance, or faster predictions after computing the weights.

In summary, given the availability of a validation set, the use of weighted PDL is highly recommended. Leveraging our Python package enables straightforward and efficient implementation of this approach.

References

[1] M. Tynes, W. Gao, D. J. Burrill, E. R. Batista, D. Perez, P. Yang, N. Lubbers, Pairwise difference regression: A ML meta-algorithm for improved prediction and uncertainty quantification in chemical search, Journal of Chemical Information and Modeling 61 (2021) 3846–3857.

[2] S. J. Wetzel, R. G. Melko, I. Tamblyn, Twin neural network regression is a semi-supervised regression algorithm, ML: Science and Technology 3 (2022) 045007.

[3] H. Shimodaira, Improving predictive inference under covariate shift by weighting the log-likelihood function, Journal of Statistical Planning and Inference 90 (2000) 227–244.

[4] M. K. Belaid, M. Rabus, E. Hüllermeier, Pairwise Difference Learning for Classification, Discovery Science (2024).

[5] S. J. Wetzel, Twin Neural Network Improved k-Nearest Neighbor Regression, arXiv preprint arXiv:2310.00664 (2023).

[6] S. J. Wetzel, How to get the most out of twinned regression methods, arXiv preprint arXiv:2301.01383 (2023).
[7] R. C. Spiers, C. Norby, J. H. Kalivas, Physicochemical Responsive Integrated Similarity Measure (PRISM) for a Comprehensive Quantitative Perspective of Sample Similarity Dynamically Assessed with NIR Spectra, Analytical Chemistry (2023).
[8] J. Jiménez-Luna, L. Pérez-Benito, G. Martinez-Rosell, S. Sciabola, R. Torella, G. Tresadern, G. De Fabritiis, DeltaDelta neural networks for lead optimization of small molecule potency, Chemical science 10 (2019) 10911–10918.
[9] Z. Fralish, P. Skaluba, D. Reker, Leveraging bounded datapoints to classify molecular potency improvements, RSC Medicinal Chemistry (2024).
[10] J. Hu, S. Yang, J. Mao, C. Shi, G. Wang, Y. Liu, X. Pu, Exploring a general convolutional neural network-based prediction model for critical casting diameter of metallic glasses, Journal of Alloys and Compounds 947 (2023) 169479.
[11] Z. Fralish, A. Chen, P. Skaluba, D. Reker, DeepDelta: predicting ADMET improvements of molecular derivatives with deep learning, Journal of Cheminformatics 15 (2023) 101.
[12] Y. Chen, Y. Ou, P. Zheng, Y. Huang, F. Ge, P. O. Dral, Benchmark of general-purpose ML-based quantum mechanical method AIQM1 on reaction barrier heights, The Journal of Chemical Physics 158 (2023).
[13] Y. Wang, R. D. King, Extrapolation is Not the Same as Interpolation, in: International Conference on Discovery Science, Springer, 2023, pp. 277–292.
[14] J. Tyree, P. Clusius, Prudent Response Surface Models, Master's thesis, University of Helsinki, 2023.
[15] J. Surowiecki, The wisdom of crowds, Anchor, 2005.
[16] T. Malisiewicz, A. Gupta, A. A. Efros, Ensemble of exemplar-svms for object detection and beyond, in: 2011 International Conference on Computer Vision, IEEE, 2011, pp. 89–96.
[17] G. Pereyra, G. Tucker, J. Chorowski, Ł. Kaiser, G. Hinton, Regularizing neural networks by penalizing confident output distributions, arXiv preprint arXiv:1701.06548 (2017).
[18] D. H. Wolpert, Stacked generalization, Neural networks 5 (1992) 241–259.
[19] S. Reid, G. Grudic, Regularized Linear Models in Stacked Generalization, in: J. A. Benediktsson, J. Kittler, F. Roli (Eds.), Multiple Classifier Systems, volume 5519, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 112–121. doi:10.1007/978-3-642-02326-2_12.
[20] D. Kraft, A software package for sequential quadratic programming, Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt (1988).
[21] J. Vanschoren, J. N. van Rijn, B. Bischl, L. Torgo, OpenML: networked science in ML, SIGKDD Explorations 15 (2013) 49–60. URL: http://doi.acm.org/10.1145/2641190.264119. doi:10.1145/2641190.2641198.
[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: ML in Python, the Journal of ML research 12 (2011) 2825–2830.
[23] B. Bischl, G. Casalicchio, M. Feurer, P. Gijsbers, F. Hutter, M. Lang, R. G. Mantovani, J. N. van Rijn, J. Vanschoren, OpenML benchmarking suites, arXiv preprint arXiv:1708.03731 (2017).

A. Reproducibility

We provide a script in the pdll GitHub repository[4] to reproduce our results. Simply start the script, and it will run the script on all datasets and machine learning baseline models we used in this paper with 5 repetitions for all 5 cross-validation folds.

We recommend you to use python 3.9 or python 3.10, because we tested it extensively with those versions. The runtime of the whole script is over a week. Feel free to select a subset of the datasets or

[4] https://github.com/Karim-53/pdll/blob/main/run_benchmark_weighted_pdr.py

baseline models to get preliminary results. Or let the script run for an hour and stop it, the script saves checkpoints that you can use.

B. Can PDL Regressor learn the antisymmetry property?

Taking a popular example for this empirical experiment, we consider the Boston house-price dataset and a minimalist training set containing only $N = 100$ data points. The base learner (histogram gradient boosting regressor) is trained with $N^2 = 10,000$ pairs. The target output ranges from 5 to 50 in the training set. We calculate the absolute difference $\Delta_{ij} = |\tilde{\Delta}(x_i, x_j) + \tilde{\Delta}(x_j, x_i)|$ for training pairs with $x_i \neq x_j$. If the pairwise difference predictor $\tilde{\Delta}$ was antisymmetric, then all Δ_{ij} would be equal to 0. However, these values may become as large as 2.5, i.e., 5.5% of the target space. The absolute difference can reach 25% on other datasets used in the evaluation section. Additionally, certain symmetric pairs have predictions of the same sign. Despite training on all pair combinations, the PDL regressor is not able to learn the antisymmetry property even on the training set.

C. Extended results

Fig. 4 presents a scatter plot that contrasts the performance of baseline machine learning models with their equivalent Pairwise Difference Regression (PDR) implementations (without any weighting). Each point on the plot signifies the average Test Mean Absolute Error (MAE) over 25 runs, categorizing the results into significant wins for PDR, significant wins for the baseline, or non-significant differences. The plot illustrates that while PDR consistently surpasses simpler ML models, achieving significant improvements over robust ensemble methods like Extra Trees becomes more challenging.

Similarly, Fig. 5 depicts a comparison between PDR and the most promising anchor weighting method based on the L2 regularization. Although the performance gains recorded with the L2 method are smaller, they remain noteworthy and significant.

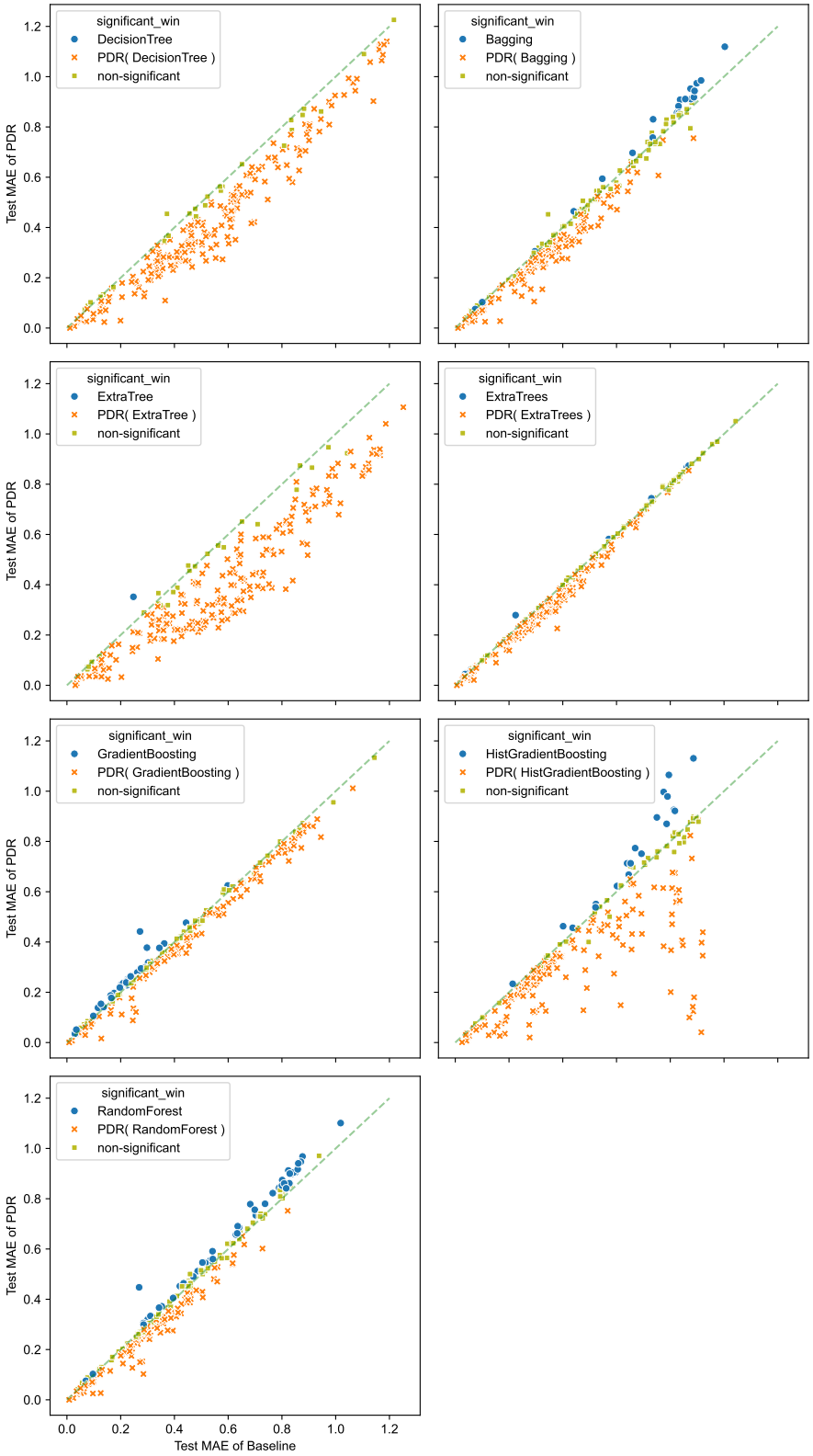

Figure 4: Scatter plot of the baseline and PDR model, showing the test MAE per dataset, averaged over 25 cross-validation runs.

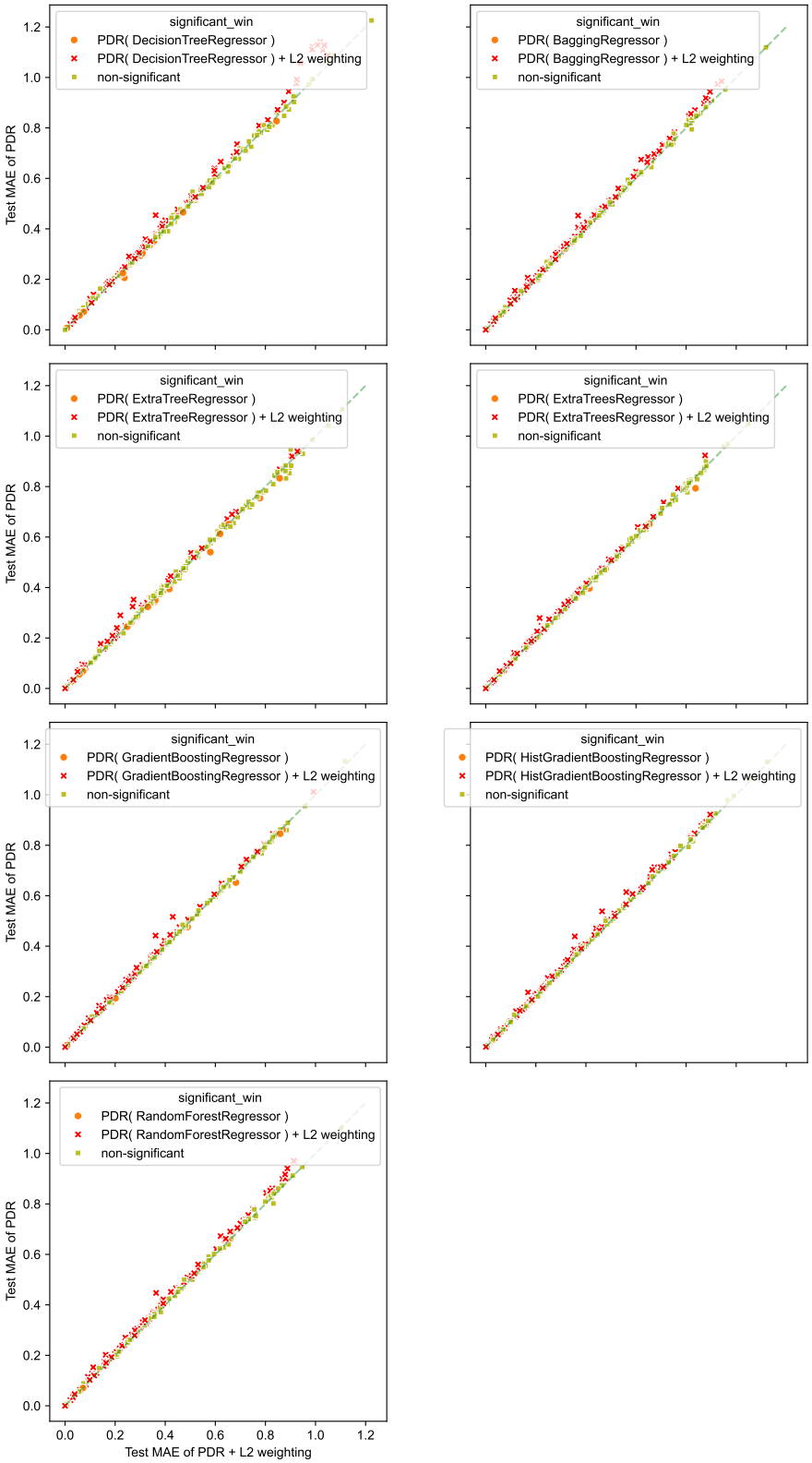

Figure 5: Scatter plot of the PDR model and weighted PDR using L2 weighting, showing the test MAE per dataset, averaged over 25 cross-validation runs.