Regularized Gradient Descent for Hyper-Parameter Tuning in the Kappa-Eta Algorithm

Deepak Ramsubhag
Department of Compututing and Information Technology
The University of the West Indies
St. Augustine, Trinidad
deepak.ramsubhag@uwi.edu

Patrick Hosein
Department of Electrical and Computer Engineering
The University of the West Indies
St. Augustine, Trinidad
patrick.hosein@uwi.edu

***Abstract*—The recently introduced Kappa-Eta algorithm has demonstrated exceptional performance in regression, classification, and other machine learning tasks. In this algorithm, each feature is associated with its own hyperparameter, which has traditionally been optimized in isolation. We show that jointly optimizing all parameters over the feature space leads to better performance due to the existence of multiple local optima. Furthermore, while independent tuning is more susceptible to the influence of outliers, joint optimization reduces these effects. We also introduce regularization and compare three approaches: independent optimization, joint optimization, and joint optimization with regularization. This combined method not only mitigates overfitting and enhances robustness against noise but also reduces computation by significantly narrowing the search space.**

***Index Terms*—Regression, Machine Learning, Parameter Tuning, Gradient Descent, Regularization, Kappa-Eta Algorithm**

I. Introduction

In machine learning, the performance of an algorithm heavily depends on proper hyperparameter tuning. Hyperparameters govern the learning process and, if set incorrectly, can lead to issues such as underfitting or overfitting, where the model fails to capture the underlying data patterns or becomes too tailored to the training data, respectively. Efficient tuning of these hyperparameters is critical, as it directly influences both the speed of convergence during training and the model's final generalization performance.

One effective approach to address this challenge is gradient descent, an iterative optimization algorithm that updates parameters by moving in the opposite direction of the gradient. At each step, the algorithm computes the gradient, which represents the slope of the loss function with respect to the parameters, and adjusts them to reduce the error. Enhancements such as momentum, which accumulates previous gradients to smooth updates and speed up convergence, and adaptive learning rate techniques, which dynamically adjust the step size to better navigate complex error landscapes, have been shown to further improve optimization performance [1]. A widely used variant that incorporates both momentum and adaptive learning rates is the ADAM optimizer, which has demonstrated strong performance across a variety of machine learning applications [2].

Regularization is a powerful tool in machine learning because it improves a model's ability to generalize to new data by controlling its complexity. Without regularization, models risk overfitting the training data, capturing noise rather than the underlying patterns. Techniques like L1 regularization (lasso) encourage sparsity by shrinking some coefficients to zero, while L2 regularization (ridge) penalizes large coefficients, reducing the model's variance. [3] introduced the Elastic Net approach, which combines these two methods to effectively balance the benefits of both penalties.

The $\kappa\eta$ regression algorithm, introduced in [4], addresses the bias-variance trade-off by optimizing the balance between personalization and robustness. Its application in determining insurance policy premiums revealed that increased personalization (i.e., incorporating more features) can reduce robustness, particularly in smaller datasets where fewer samples are available per feature. Its overall effectiveness was further validated in [5], where the algorithm outperformed other popular regression models.

As shown in [6], the $\kappa\eta$ algorithm efficiently handles both continuous and categorical data, making it well-suited for datasets with high feature dimensionality or limited sample sizes. The algorithm first encodes features into the target space using a weighted average of the target value. This enables meaningful comparisons between distinct attributes, such as age and gender.

The algorithm relies on a distance metric and hyperparameters, κ and η, which serve as exponents in the weighting function, as defined in Equation (1). These parameters regulate the influence of distant data points on the weighted average: a larger κ or η significantly reduces their impact, whereas a smaller value leads to a more gradual decrease.

$$\hat{y}_i(\kappa) = \frac{\sum_{j \in X^{tr}} \frac{y_j^{tr}}{(1+d[i,j])^{\kappa}}}{\sum_{j \in X^{tr}} \frac{1}{(1+d[i,j])^{\kappa}}} \tag{1}$$

For each feature there is an associated κ value that must be optimized to obtain an encoded feature value represented in target value units. Using feature distances we can then compute sample distances and perform a similar calculation for the predicted target of the test sample using a parameter η.

Traditionally, these parameters have been tuned independently, ignoring the interactions among features, which can lead to suboptimal performance. Moreover, conventional methods lack regularization, often yielding parameter values that are higher than necessary resulting in significant overfitting. To address these issues, we employ a regularized gradient descent method to identify the optimal parameter values, ensuring computational efficiency while enhancing robustness to noise and outliers, which ultimately improves model performance.

II. Related Work and Contributions

The following studies present efficient tuning methods for determining the optimal hyperparameters in the $\kappa\eta$ algorithm. However, they tune each feature independently, overlooking potential interactions among features. [7] proposed using Successive Quadratic Approximation (SQA) to efficiently determine the optimal κ value. SQA iteratively refines a quadratic approximation of the objective function, guiding the search toward its minimum. The study demonstrated that optimizing κ using SQA, based on the function's shape properties, was significantly more efficient than traditional methods. [8] proposed a method for approximating optimal κ values using statistical properties of data including density, noise and seasonality. They also employed a modified Golden Section Search algorithm to efficiently minimize the error function curve. While these approaches improved the algorithm's computational efficiency significantly, they remained susceptible to overfitting and did not improve performance.

The following studies have improved the algorithm's resilience to noise and outliers by refining its tuning process and adjusting core components, including loss functions and distance metrics. [9] increased robustness to noise and outliers by incorporating Huber loss in the encoding stage of the algorithm to improve performance. [10] highlighted that noise and outliers can cause poor encoding of values in the model. They showed that using alternative distance metrics, such as Manhattan distance or Minkowski distance can significantly enhance the model's resilience to these disturbances.

Our contributions include a regularized, multi-parameter gradient descent approach for hyperparameter tuning in the $\kappa\eta$ algorithm. This approach significantly improves overall performance while maintaining high computational efficiency.

III. Overview of the $\kappa\eta$ Algorithm

The $\kappa\eta$ algorithm is divided into two distinct stages: an encoding stage followed by a predictive stage. However, in the case of a univariate dataset, the algorithm reduces to a single stage, where x_i directly serves as the prediction for any test sample i, as illustrated in Equation. (2).

$$x_i = \frac{\sum_{j \in X^{tr}} \frac{y_j^{tr}}{(1+d_f[i,j])^{\kappa}}}{\sum_{j \in X^{tr}} \frac{1}{(1+d_f[i,j])^{\kappa}}} \tag{2}$$

A. Encoding

In [6], the authors present a method for encoding categorical features first using mean target encoding. Then any sample point x_i can then be encoded using Equation (2), which incorporates a weighting function, a tuning parameter, and a distance metric. In this formulation, the target values of training features are weighted using Equation (3).

$$\sum_{j \in X^{tr}} \frac{y_j^{tr}}{(1+d_f[i,j])^{\kappa}} \tag{3}$$

Here, y^{tr} represents the training target, X^{tr} denotes the feature training data, and i corresponds to the i^{th} sample being encoded. The distance $d[i,j]$ used in the weighting function measures the distance between the sample point to be encoded, x_i, and a training point for a single feature.

B. Prediction

Once each sample point x_i has been encoded across all features $f \in F$, the algorithm proceeds to the predictive stage, where the focus shifts from individual feature distances to the overall distance between samples. The final prediction is then computed using a similar approach to Equation (2).

$$\hat{y}_i(\eta) = \frac{\sum_{j \in X^{tr}} \frac{y_j^{tr}}{(1+D[i,j])^{\eta}}}{\sum_{j \in X^{tr}} \frac{1}{(1+D[i,j])^{\eta}}} \tag{4}$$

The weighting formula maintains the same structure as Equation (3), but in this context, the distance $D[i,j]$ represents the total distance derived from the individual feature distances between samples i and j, where $\hat{y}_i$ is the predicted value for a test sample i. The exponent used in this stage is η, and the Minkowski distance serves as the distance metric, enabling the use of Euclidean or Manhattan distance depending on the value of p, as defined in Equation (5). This flexibility has been shown to enhance the algorithm's adaptability across different dataset types, as demonstrated in [10].

$$d_{\text{Minkowski}}[i,j] = \left(\sum_{f=1}^{n} |x_{i,f} - x_{j,f}|^p \right)^{1/p} \tag{5}$$

In summary, this algorithm imputes an encoded feature value by computing a weighted average of the target variable, where weights are determined based on feature distances. It then generates predictions by applying a similar weighted averaging approach, this time considering the distance between entire samples using the previously imputed values. The hyperparameters κ and η control the bias-variance trade-off during encoding and prediction, respectively. When $\kappa = 0$ or $\eta = 0$, x_i or $\hat{y}_i$ is assigned the mean of the training target values. Increasing κ and η places greater emphasis on samples closer to x_i or $\hat{y}_i$. To prevent data leakage, target test values y^t are never used in feature encoding or final predictions. Additionally, an optimal κ exists for each feature in the encoding process, and an optimal η for the prediction stage. Algorithm 1 presents the pseudo-code for the $\kappa\eta$ regression algorithm, while a visual representation of its mechanics is available at [11].

Algorithm 1 Pseudocode of the $\kappa\eta$ algorithm

$C \equiv$ set of categorical features
$N \equiv$ set of numerical features
$X_{tr} \equiv$ set of training samples
$X_t \equiv$ set of testing samples
$F \equiv$ set of features
$\kappa_f \geq 0$ for each $f \in F \equiv$ hyperparameters
$\eta \geq 0 \equiv$ hyperparameter
$p \equiv$ hyperparameter
$v_f \equiv$ set of categories for feature f
for each $f \in C$ **do**
  for each $v \in v_f$ **do**
    $z \equiv \{x \in X_{tr} | x_f = v\}$
    $\mu_{f,v} \leftarrow \frac{1}{|z|} \sum_{x \in z} y_x$ (mean target value over training samples where feature f has value v)
  end for
  Replace category values with their mean μ_v in both X_{tr} and X_t
end for
for each $f \in N \cup C$ **do**
  for each sample j in X_{tr} **do**
    $x^{tr}_{j,f} \leftarrow \frac{x^{tr}_{j,f}}{\max(x^{tr}_f) - \min(x^{tr}_f)}$ (normalize feature values in the train set)
  end for
  for each sample j in X_t **do**
    $x^{t}_{j,f} \leftarrow \frac{x^{t}_{j,f}}{\max(x^{tr}_f) - \min(x^{tr}_f)}$ (normalize feature values in the test set)
  end for
end for
for each $f \in N \cup C$ **do**
  for each sample i with unique feature value v in X_{tr} and X_t **do**
    $\mu_{f,v} \leftarrow \frac{\sum_{j \in X_{tr}} \frac{y^{tr}_j}{(1+d_f[i,j])^{\kappa_f}}}{\sum_{j \in X_{tr}} \frac{1}{(1+d_f[i,j])^{\kappa_f}}}$ (imputed mean target value over training samples when feature f has value v)
  end for
  Replace feature values with the imputed mean values $\mu_{f,v}$ in both X_{tr} and X_t
end for
for each test sample i in X_t **do**
  for each training sample j in X_{tr} **do**
    $D[i,j] \leftarrow \left(\sum_{f \in F} |x_{i,f} - x_{j,f}|^p \right)^{1/p}$ (calculate Minkowski Distance)
  end for
end for
for each test sample i in X_t **do**
  $\hat{y}_i(\eta) \leftarrow \frac{\sum_{j \in X_{tr}} \frac{y^{tr}_j}{(1+D[i,j])^{\eta}}}{\sum_{j \in X_{tr}} \frac{1}{(1+D[i,j])^{\eta}}}$
end for
Return output values $\hat{y}$

IV. Problem and Solution Illustration

A. Noise and Outliers

When optimizing κ for a given feature or η, significant noise or outliers can lead to poor tuning. Consider a one-dimensional dataset split into a training and test set, where both contain a major outlier with a similar feature value, as illustrated in Figure 1. In such cases, severe overfitting occurs because the presence of outliers during tuning pushes the optimal κ to an extremely high value, as indicated by the yellow line. This happens because increasing κ significantly amplifies the influence of the nearest training sample, which is the other outlier with a similar target value. This reduces the impact of further training samples thereby reducing the overall error. In contrast, when the outliers were removed, the optimal κ is found to be much lower, as shown by the brown line. This is the typical shape of the error(κ) function.

Cases like this can arise from different types of outliers, including global outliers, where a feature value is extreme relative to the dataset or a target value deviates significantly from what the features would typically predict. Contextual outliers occur when a target value is only anomalous under specific feature conditions, while collective outliers emerge when a group of feature-target pairs forms an unusual pattern that deviates from the expected trend. Depending on how the data is split during tuning, the effects of these outliers can persist, and even with cross-validation across multiple folds, large errors may still be present across evaluations.

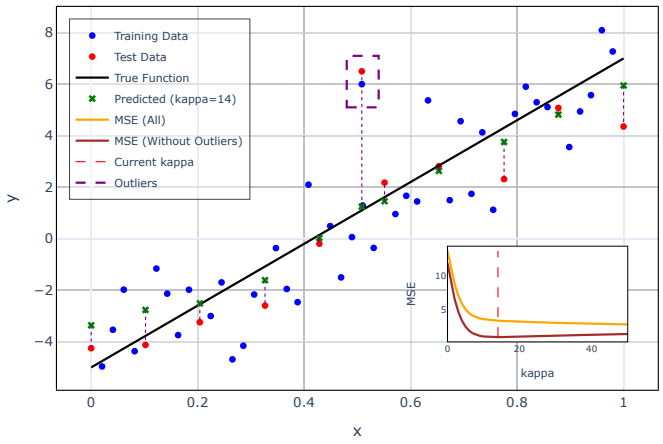

Fig. 1. Effects of Outliers on Optimal κ

To alleviate this problem, we implement elastic net regularization by applying a penalty to all parameters, including all κ values and η, to prevent them from becoming excessively high. Elastic net regularization combines the benefits of L1 and L2 penalties, encouraging both sparsity and small parameter values respectively. Sparsity is particularly beneficial in this algorithm because when the κ values are reduced to zero, the associated features no longer contribute to the final predictions enabling automatic feature selection. The overall regularization term in our approach is given by

$$\text{Reg} = \lambda\Big[\alpha\,\mathcal{L}_1 + (1-\alpha)\,\mathcal{L}_2\Big] \tag{6}$$

$$\mathcal{L}_1 = \sum_f |\kappa_f| + s_\eta\,|\eta|, \qquad \mathcal{L}_2 = \sum_f \kappa_f^2 + s_\eta\,\eta^2.$$

where:

- λ is the regularization strength,
- α controls the balance between the L1 (sparsity-promoting) and L2 (shrinkage-promoting) components,
- s_η is a scaling factor applied to η to impose a smaller penalty on it.

We have found experimentally that when η is significantly larger than κ values, performance improves. To account for this, we apply a scaling factor to η so that it incurs a smaller penalty, while maintaining a single λ for overall simplicity.

B. Multiple Minima in Parameter Tuning

Consider the Iris dataset specified in Table I, where features indexed by 2 and 3 were used. Figure 2 presents a contour plot of the error of predictions relative to κ_1 and κ_2, which correspond to the respective features. A fixed value of $\eta = 20$ was used and the plot reveals two minima, with κ_1 values around 35 and 55 yielding similar results for a single κ_2 value. As the number of features increases, the complexity of the error landscape grows and experimental observations confirm the existence of multiple minima.

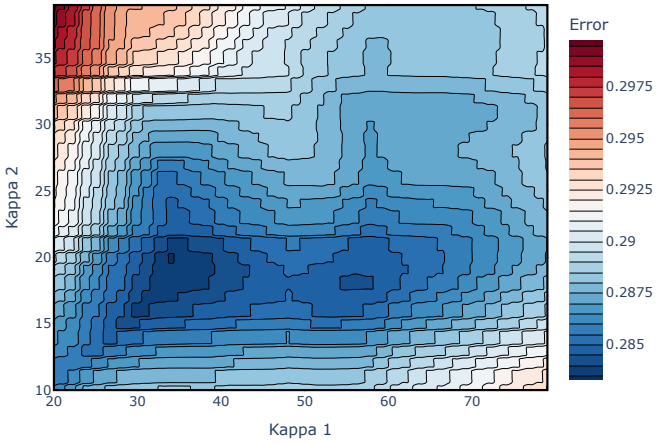

Fig. 2. Contour Visualization of Error vs. κ_1 and κ_2

To address this, we employ a multi-parameter gradient descent method, using an ADAM optimizer, which adapts the learning rate for each parameter and smooths out noise in the gradients. By combining momentum (to accelerate through plateaus) and adaptive scaling (to handle different gradient magnitudes), ADAM is well-suited for this application as it performs particularly well in noisy or nonconvex settings, where it more effectively avoids suboptimal local minima. It also requires less tuning compared to standard gradient descent and converges rapidly.

TABLE I
SUMMARY OF DATASETS

Dataset	Samples, N	Features, f	Target Value, Y	Citation
Student Math	394	32	G3	[12]
Auto	392	8	mpg	[13]
Energy Y2	768	8	Y2	[14]
Energy Y1	768	8	Y1	[14]
Iris	150	4	Sepal Length	[15]
Concrete	1030	8	Comp Str	[16]
Wine Quality	1599	11	Res. Sugar	[17]

V. Proposed Approach

First, consider the traditional approach. The tuning process involves K-fold cross-validation at both the encoding and predictive stages. For encoding, an optimal κ value is determined separately for each feature. To evaluate a single κ, the data is split into training and test subsets across the folds, encoded values are computed using the training subset, and the mean error is calculated over all samples and folds. A similar procedure is used to identify the optimal η value for the final prediction stage. Traditionally, this tuning is performed using a linear search, incrementing κ from zero by a fixed step size, with each candidate κ or η value requiring evaluation.

In the proposed method, the combination of all parameters are evaluated by splitting the training data into multiple folds using K-fold cross-validation. Within each fold, the data is divided into a training subset and a corresponding test subset. Feature values in the test set are encoded using Equation (2), and distances between samples are computed. These distances are then used to generate predictions via Equation (4). After obtaining predictions for all test samples in a fold, the mean error is calculated and then averaged across all folds to obtain the overall cross-validation error. This error, combined with the elastic net regularization term in Equation (6), forms the objective function used in the gradient descent optimization to simultaneously tune the κ values and η. Algorithm 2 outlines the gradient descent process for tuning parameters.

VI. Numerical Results

We evaluated the performance of the $\kappa\eta$ algorithm using different tuning strategies across all datasets summarized in Table I using all features. All experiments employed 5-fold cross-validation, with Mean Squared Error (MSE) as the loss function and Euclidean distance as the distance metric. A regularization parameter of $\alpha = 0.5$ and a scaling factor $s_\eta = 0.1$ (Equation 6) were consistently used. For gradient descent, the following parameters were set: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1 \times 10^{-8}$, and a maximum of 1000 iterations. A gradient norm tolerance of 1×10^{-6} and an error tolerance of 1×10^{-8} were applied across all datasets. A single starting point was used, with all κ values initialized to 1 and η set to 25. The only parameters varied for each dataset were the learning rate

Algorithm 2 Gradient Descent Method for the $\kappa\eta$ Algorithm

Input: Training data (X, y), initial parameters $\kappa^{(0)}$, $\eta^{(0)}$, learning rate γ, Adam parameters β_1, β_2, gradient norm tolerance *tol*, error tolerance ϵ, maximum iterations T, and regularization parameters λ, α, scaling factor s_η.
Initialize:
  Set moment estimates $m \leftarrow \mathbf{0}$ and $v \leftarrow \mathbf{0}$.
  Set iteration counter $t \leftarrow 1$.
while $t \leq T$ **do**
  Compute the gradients (partial derivative with respect to each parameter):

$$g_\kappa,\, g_\eta \leftarrow \nabla_{\kappa,\,\eta}\, \mathcal{L}(X, y, \kappa, \eta)$$

  Form the gradient vector:

$$g \leftarrow \begin{bmatrix} g_\kappa \\ g_\eta \end{bmatrix}$$

  if $\|g\| <$ *tol* **then**
    break
  end if
  Update the Adam moment estimates:

$$m \leftarrow \beta_1\, m + (1 - \beta_1)\, g$$

$$v \leftarrow \beta_2\, v + (1 - \beta_2)\, g^2$$

  Compute bias-corrected estimates:

$$\hat{m} \leftarrow \frac{m}{1 - \beta_1^t}, \quad \hat{v} \leftarrow \frac{v}{1 - \beta_2^t}$$

  Compute the parameter update:

$$\Delta \leftarrow \gamma\, \frac{\hat{m}}{\sqrt{\hat{v}} + \epsilon}$$

  Update the parameters:

$$\begin{bmatrix} \kappa \\ \eta \end{bmatrix} \leftarrow \begin{bmatrix} \kappa \\ \eta \end{bmatrix} - \Delta$$

  Project the parameters onto the nonnegative domain (1×10^{-10} used instead of 0 for numerical stability):

$$\kappa \leftarrow \max(\kappa,\, 1 \times 10^{-10}), \quad \eta \leftarrow \max(\eta,\, 10^{-10})$$

  Evaluate the current objective function $\mathcal{L}(X, y, \kappa, \eta)$.
  if improvement in $\mathcal{L}$ is less than ϵ **then**
    break
  end if
  $t \leftarrow t + 1$
end while
Output: Optimized parameters κ, η.

TABLE II
MAE AND MSE OF EACH APPROACH ACROSS DATASETS

	$\kappa\eta$ Independent		$\kappa\eta$ Joint		$\kappa\eta$ Reg_Joint		KNN		Random Forest		XGBoost	
Dataset	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE
Student Math	1.158	3.786	1.040	3.233	1.023	2.823	2.931	15.343	**0.992**	**2.656**	1.057	2.712
Auto	2.077	8.736	2.006	8.363	2.000	8.191	2.068	8.309	**1.908**	7.525	1.910	**7.260**
Energy Y2	0.841	1.790	0.701	1.568	**0.701**	**1.566**	1.472	3.986	0.790	1.859	0.765	1.576
Energy Y1	0.292	0.209	0.278	0.183	**0.279**	**0.181**	1.157	2.824	0.385	0.505	0.397	0.461
Iris	0.276	0.127	0.270	0.118	**0.265**	**0.118**	0.277	0.122	0.277	0.12	0.281	0.128
Concrete	4.589	46.34	4.353	41.49	4.327	41.60	6.028	71.856	3.399	24.578	**3.152**	**19.968**
Wine Quality	0.478	1.090	0.453	1.090	0.453	1.090	**0.432**	0.987	0.472	0.878	0.464	**0.806**

and the regularization strength λ, both adjusted in powers of 10. Table II presents a comparison of three tuning methods: Independent, where each parameter is tuned individually, Joint, which uses gradient descent without regularization, and Regularized_Joint, which incorporates regularization into the joint tuning approach. We also compare these results with optimally tuned models including K-Nearest Neighbors (due to its similarity to $\kappa\eta$), Random Forest, and XGBoost.

VII. DISCUSSION

We observe that the joint-parameter gradient descent approach outperforms traditional individual tuning, and adding regularization further improves performance by mitigating overfitting and controlling excessive parameter values. The proposed tuning method for the $\kappa\eta$ algorithm consistently yields better results than KNN and outperforms XGBoost and Random Forest, leading regression models, on many datasets.

Notably, optimal κ values varied with the tuning method. In the Concrete dataset, the traditional approach produced values near the upper bound for the search (100), while gradient descent yielded more conservative values around 1.0.

In experiments, the initial values of κ and η were found to significantly influence the results. Starting κ values too close to 0 or setting η too low often led to poor minima. Initializing κ values at 1 ensures that all features contribute to the predictions, allowing the bias-variance trade-off to be fine-tuned effectively thereafter. Moreover, higher initial η values relative to κ yielded better outcomes. Traditional tuning methods typically produce much higher κ values than η because parameters are tuned sequentially. In contrast, our approach tunes both simultaneously, allowing η to have a greater impact. It is crucial to carefully tune η, as it governs the ultimate balance between bias and variance.

VIII. CONCLUSION AND FUTURE WORK

We conclude that optimizing the $\kappa\eta$ algorithm using a multi-parameter approach, such as gradient descent, significantly outperforms methods that tune each feature independently. Incorporating regularization further prevents overfitting and enhances performance. Combining these strategies improves robustness to noise and outliers while efficiently determining optimal parameters despite a complex error landscape. Future work may explore global optimization methods for tuning hyper parameters and further examine regularization's impact on the error landscape.

REFERENCES

[1] S. Ruder, "An overview of gradient descent optimization algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1609.04747

[2] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017. [Online]. Available: https://arxiv.org/abs/1412.6980

[3] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 67, no. 2, pp. 301–320, 2005.

[4] P. Hosein, "A data science approach to risk assessment for automobile insurance policies," *International Journal of Data Science and Analytics*, pp. 1–12, 2023.

[5] S. Gooljar, K. Manohar, and P. Hosein, "Performance evaluation and comparison of a new regression algorithm," *ArXiv*, vol. abs/2306.09105, 2023.

[6] K. Baboolal, S. Gooljar, and P. Hosein, "A novel approach to feature encoding," in *2023 IEEE International Conference on Technology Management, Operations and Decisions (ICTMOD)*, 2023, pp. 1–6.

[7] P. Hosein, K. Manohar, and K. Manohar, "A successive quadratic approximation approach for tuning parameters in a previously proposed regression algorithm," in *Proceedings of the 12th International Conference on Data Science, Technology and Applications - Volume 1: DATA*, INSTICC. SciTePress, 2023, pp. 629–633.

[8] D. Ramsubhag, K. Baboolal, and P. Hosein, "Efficient hyper-parameter tuning for the kappa regression algorithm," in *2024 International Conference on Decision Aid Sciences and Applications (DASA)*, 2024, pp. 1–6.

[9] D. Ramsubhag and P. Hosein, "Kappa encoding using the huber loss function," in *2024 IEEE International Conference on Technology Management, Operations and Decisions (ICTMOD)*, 2024, pp. 1–6.

[10] ——, "On the determination of a suitable distance metric for the kappa regression algorithm," in *2024 IEEE International Conference on Technology Management, Operations and Decisions (ICTMOD)*, 2024, pp. 1–6.

[11] ——, "Kappa visualization," Available online at https://deepak-ramsubhag.github.io/Kappa-Algorithm-Visualization/.

[12] P. Cortez, "Student performance," UCI Machine Learning Repository, 2014.

[13] R. Quinlan, "Auto mpg," UCI Machine Learning Repository, 1993.

[14] A. Tsanas and A. Xifara, "Energy efficiency," UCI Machine Learning Repository, 2012.

[15] R. A. Fisher, "Iris," UCI Machine Learning Repository, 1988.

[16] I.-C. Yeh, "Concrete compressive strength," UCI Machine Learning Repository, 2007.

[17] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Wine quality," UCI Machine Learning Repository, 2009.