k-Most-Influential Neighbours: Noise-Resistant kNN

Matt Mallen, Derek Bridge

School of Computer Science & Information Technology, University College Cork, Ireland

Abstract

The k-Nearest Neighbours (kNN) algorithm is a fundamental machine learning method known for its simplicity and effectiveness. However, it is notably sensitive to noisy data, which can significantly degrade its performance. We introduce k-Most-Influential Neighbours (kMIN), a novel variant designed to enhance noise resistance while adhering to the lazy learning paradigm of kNN. kMIN integrates reliability and similarity measures to obtain what we call influence. kMIN uses influence to identify neighbours and to weight their contributions when making predictions, mitigating the impact of unreliable training examples. We evaluate kMIN against traditional kNN, Weighted-kNN, and dataset editing algorithms for noise-filtering across multiple classification and regression datasets with varying noise levels. The results indicate that kMIN shows promise in noisy datasets, outperforming existing methods in several cases, but with further testing required to solidify these findings.

Keywords

k-nearest Neighbours, edited nearest neighbours, instance selection, noisy data

1. Introduction

The k-Nearest Neighbours (kNN) algorithm, first developed by Fix and Hodges [1], is a staple in machine learning for both classification and regression tasks due to its simplicity and intuitive approach. It operates on the premise that similar problems have similar solutions; that is to say, it assumes that the target value or class of a query example can be predicted by aggregating the values/classes of the training examples that are closest in distance to it. Despite its advantages, kNN is notably sensitive to noisy (erroneous) or unrepresentative examples (outliers or atypical cases) in the training set [2]. Noisy data can significantly skew the distance calculations and, consequently, the predictions. This sensitivity hampers kNN's performance in real-world applications where data imperfections are common. Methods that remove noise or that enhance kNN's resistance to noise are crucial for improving its applicability to practical scenarios, especially in safety-critical domains such as disease prediction [3].

There exist several dataset editing methods that attempt to identify and delete noisy examples from training sets, in preparation for use of kNN (see Section 2.1). However, they have limitations (Section 2.2), notably a focus on classification to the neglect of regression.

Instead, we seek to develop a variant of kNN that enhances kNN's resistance to noise. Robustness to noise ensures more reliable predictions, increases the algorithm's utility in diverse domains, and reduces the need for extensive dataset editing, which can be resource-intensive.

In this paper, we propose the k-Most-Influential Neighbours (kMIN) algorithm, a novel variant of kNN that incorporates measures of reliability and similarity into the neighbour selection and aggregation processes. kMIN aims to mitigate the influence of noisy training examples while departing as little as possible from kNN's lazy learning paradigm. Our key contributions include:

- Introducing a reliability metric based on an example's contribution to accurate predictions.
- Integrating reliability with similarity to adjust neighbour influence dynamically.
- Evaluating kMIN's performance against traditional kNN, Weighted kNN, and kNN on datasets that have been cleaned using dataset editing methods — on multiple datasets with varying noise levels.

The rest of this paper proceeds as follows:

AICS 2024: 32nd Irish Conference on Artificial Intelligence and Cognitive Science, December 9–10, 2024, University College Dublin
mattmallen@protonmail.com (M. Mallen); d.bridge@cs.ucc.ie (D. Bridge)
0000-0002-8720-3876 (D. Bridge)

- **Section 2: Related Work** – We review prior research on kNN and its variants, including Weighted-kNN; we present dataset editing algorithms for classification (in particular, RENN and BBNR); and, we show how to extend dataset editing algorithms to regression tasks.
- **Section 3: kMIN Algorithm** – We introduce the k-Most-Influential Neighbours (kMIN) algorithm, defining reliability, similarity, and influence, and explain their roles in fetching and aggregating neighbours.
- **Section 4: Evaluation** – We describe the datasets, experimental setup, and statistical methods used to compare kMIN with kNN, Weighted-kNN, and dataset editing algorithms on datasets with and without injection of artificial noise.
- **Section 5: Results** – We present the experimental results, highlighting kMIN's performance in handling noise and compare it to existing methods in both classification and regression tasks.
- **Section 6: Conclusions, Limitations, and Future Work** – We conclude with key findings, discuss limitations, and outline future research directions.

2. Related Work

The standard kNN algorithm predicts the target value or class of a query example by identifying the k closest examples in the training set based on a distance metric, commonly Euclidean distance [1]. For regression tasks, the prediction will be the mean of the neighbours' target values; for classification tasks, the predicted class will be determined by a vote among the neighbours. In Weighted-kNN, which is a variant of the standard kNN algorithm, neighbours have weights, which are inversely proportional to their distance from the query example, giving closer neighbours more influence [4]. The prediction is then either a weighted mean or weighted vote of the neighbours' target values/classes.

kNN and its variants are lazy learners: generalization takes place at inference time, when given a query example for which a prediction is required. There is no training step, prior to inference. In eager learners, by contrast, a model that generalizes the training data is learned in a step that precedes use of the model for inference. Lazy learners have the disadvantage of tending to be slower at inference time, since most, if not all, of the computation has been deferred. But they have the advantage of being easily incremental. When new training examples arrive, they can be inserted into the training set and will have immediate effect on subsequent inference. For eager learners to incorporate new training examples requires that models be re-built, which, because it is expensive, is usually only done periodically.

Nevertheless, lazy learners may include pre-processing steps that come prior to inference. For kNN, for example, pre-processing might involve the following: construction from the training data of a data structure, such as a kd-tree [5], to speed-up inference-time retrieval of the k-nearest neighbours; or computation of all-pairs similarities between training examples for faster inference-time identification of the k nearest-neighbours by an algorithm such as Fish-and-Shrink [6]; or feature-selection, to identify irrelevant or noisy features to exclude them from computation of distances, or feature-weighting to ascribe importance weights to features for use in the distance function [7]. Pre-processing might also involve deleting examples from the training set, and we discuss methods for doing this in more detail in the remainder of this section.

2.1. Dataset Editing Algorithms for Classification

To address kNN's vulnerability to noise [2], various noise-reduction techniques have been developed. We refer to them as *dataset editing algorithms*. They clean datasets in a pre-processing step, prior to using kNN for inference. It is important to distinguish between two main goals of dataset editing algorithms: removing noise and reducing redundancy. Redundancy reduction is about discarding surplus data to improve efficiency. Our focus in this paper is with noise reduction. We will describe two such algorithms for classification tasks.

2.1.1. Repeated Edited Nearest Neighbours (RENN)

One editing algorithm is Repeated Edited Nearest Neighbours (RENN), as first proposed by Tomek [8]. RENN operates by repeatedly applying Wilson's Edited Nearest Neighbour (ENN) rule [9]. The ENN rule involves examining each example in the training set and removing it if its class label does not agree with the majority class of its k-nearest neighbours. Unlike the basic ENN, which is applied once, RENN iteratively uses the ENN rule until the algorithm cannot remove any more examples. Repeated application of the ENN rule allows for deeper cleaning, ensuring that only the most representative and reliable examples remain.

2.1.2. Blame-Based Noise Reduction (BBNR)

Another editing algorithm is Blame-Based Noise Reduction (BBNR) as first proposed by Delany and Cunningham [10]. We describe and use version 2 of the algorithm [11]. BBNR operates on the principle of 'blaming' examples that are frequently involved in incorrect kNN predictions. While RENN focuses on which examples kNN has misclassified, BBNR instead highlights which examples are helping to misclassify their neighbours.

BBNR assesses the impact of each training example on the overall prediction accuracy by finding its *Liability Set*. The Liability Set of example $\mathbf{x}$ is the set of other training examples that are misclassified by their k-nearest neighbours and where $\mathbf{x}$ was one of the incorrect majority neighbours. BBNR then removes the most problematic examples (those with largest Liability Sets) one by one. However, removal is only tentative. After removing an example from the training set, it carries out a check to see whether removal of the example has resulted in previously correctly classified examples now having erroneous predictions. If removal of the example has resulted in incorrect predictions, then the example is reinserted into the training set. To assist with this, BBNR uses the concept of a *Coverage Set*. The *Coverage Set* of an example $\mathbf{x}$ is the set of other training examples that are correctly classified by their k-nearest neighbours and where $\mathbf{x}$ was one of the correct majority neighbours. When BBNR tentatively removes an example $\mathbf{x}$, then the check that was mentioned above needs to be applied to members of $\mathbf{x}$'s *Coverage Set*. We will also use the notion of a Coverage Set in defining kMIN (Section 3).

2.2. Limitations of Existing Methods

2.2.1. Deviation from Lazy Learning

Pre-processing steps, like the ones introduced by editing algorithms, such as BBNR and RENN, move away from the classic kNN paradigm. This may reduce some of the unique appeal of kNN that may make a designer opt for it in a given application context. In particular, the algorithm becomes less straightforwardly incremental. The editing algorithm must be re-run periodically, possibly even re-instituting examples that were previously deleted. Our kMIN algorithm also departs from strict laziness but it confines its departure to the minimal pre-calculation of reliability values without altering the training data.

2.2.2. Over-elimination Risk

Aggressive noise filtering can remove valuable training examples, leading to information loss and potential degradation in predictive performance, particularly since filtering is a binary decision to keep or delete an example. Indeed, the original version of BBNR was found to be too aggressive on many datasets, leading to the more conservative version 2 [11]. The decision of whether to keep or delete examples is typically based on reliability, but estimating reliability well in neighbourhoods of mixed quality remains challenging [12].

Unlike dataset editing algorithms, kMIN does not make binary decisions. Instead, it integrates reliability into the influence calculation, adjusting the contribution of examples dynamically without removing them. This helps avoid the risk of over-elimination and retains potentially useful examples.

2.2.3. Applicability Constraints

There has been a focus on editing algorithms for classification tasks, leaving editing algorithms for regression tasks relatively neglected. kMIN applies just as well for regression as it does for classification. To enable us to compare kMIN with RENN and BBNR on both classification tasks and regression tasks, we review the work on adapting RENN and BBNR to regression and propose our own variants in the next subsection.

2.3. Dataset Editing Algorithms for Regression

Key to the operation of RENN and BBNR in classification tasks is knowing what it means for one training example to *agree* with others. In classification, agreement can be straightforwardly determined by whether the class labels of two examples match. Then, in RENN and BBNR, we are looking for examples whose class does not agree with the majority class of its k-nearest neighbours, with RENN focusing on the misclassified example and BBNR focusing on the neighbours that cause the misclassification.

For regression tasks, it is more challenging to define agreement. It would be much too strict to define it in terms of exactly equal target values. Instead, agreement will be when target values are close enough. But it is not clear how to draw the 'dividing line'.

Kordos & Blachnik adapt the ENN rule to regression tasks: two target values agree if their absolute difference is less than or equal to some threshold [13]. More specifically, they consider the absolute difference between an example's target value and the value predicted by its neighbours, which is the mean of the neighbours' target values. For the threshold θ, they use the standard deviation of the neighbours' target values, multiplied by some hyperparameter α. Martin *et al.* apply this definition of agreement to give versions of RENN, BBNR and a number of other dataset editing algorithms suitable for regression tasks [14].

In our experiments (Section 4), we wanted to include dataset editing algorithms for the regression datasets as well as the classification datasets. However, we use our own modified definition of agreement for regression tasks, which differs slightly from the one in [13]. Where Kordos & Blachnik compare an example's target value with the value predicted by its neighbours (the mean of their target values), we carry out an element-wise check: an example agrees with a neighbour if the absolute difference between their target values is within a certain threshold. For overall agreement, we retain the need for majority agreement, as specified in the original RENN and BBNR algorithms.

In addition, we also have an alternative way of defining the threshold. Where Kordos & Blachnik use a threshold that is proportional (by an amount α) to the standard deviation of the neighbours' target values, we also consider a threshold that is proportional to the standard deviation of the target values of the whole training set. The choice between these two ways of calculating the threshold is a matter for model selection, using the validation sets (Section 4).

Our element-wise definition of agreement adheres more closely to the original ENN and dataset editing algorithms, such as RENN and BBNR, which rely on individual comparisons rather than aggregate statistics. We hypothesise that this fine-grained approach allows for a more precise identification of noisy instances in regression tasks. It might be interesting to conduct an empirical comparison between the two ways of computing agreement in regression tasks. We leave this to future work, since differences are likely to be small and the focus of this paper is on comparing dataset editing with our kMIN approach. In the experiments reported in Section 4, results for dataset editing on regression datasets use our element-wise approach.

3. k-Most-Influential Neighbours (kMIN)

The k-Most-Influential Neighbours (kMIN) algorithm's guiding principle is that an example's neighbours should not contribute equally when predicting the example's target value or class. Specifically, it posits that neighbours that are more similar to the example and that have higher reliability should significantly influence the outcome — with the balance between reliability and similarity being a tunable hyperparameter.

3.1. Reliability, Similarity and Influence

When defining *reliability*, we take inspiration from the concept of a *Coverage Set*, which we explained in the context of BBNR earlier (Section 2.1.2). For kMIN, the reliability r of an example $\mathbf{x}$ is the size of its *Coverage Set*. So, if, using standard kNN, a training example helps to predict ten other training examples correctly, it has a reliability of 10. Here, when we say "predict correctly", we are referring to *agreement* (Section 2.3): where the examples share the same class for classification tasks, and where the absolute difference between the example's target values is less than or equal to threshold θ for regression.

When defining *similarity*, we use the inverse of Euclidean Distance (d): $s(\mathbf{x}, \mathbf{x}') = 1/(d(\mathbf{x}, \mathbf{x}') + \epsilon)$. We add a small constant, ϵ, to prevent division-by-zero.

The *influence* of an example $\mathbf{x}'$ on an example $\mathbf{x}$ is a combination of the similarity between the two and the reliability of $\mathbf{x}'$:

$$i(\mathbf{x}, \mathbf{x}') = (\lambda \times s(\mathbf{x}, \mathbf{x}')) + ((1 - \lambda) \times r(\mathbf{x}') \tag{1}$$

Here it becomes clear why we need to use similarity rather than distance. Distance would be 'at odds' with reliability: high influence comes with low distance (high similarity) and high reliability.

In fact, there remains a problem. To effectively tune hyperparameter λ requires that similarity s and reliability r have values that are comparable with each other. After some preliminary experimentation with several methods, we chose to standardize their values: we compute the mean and standard deviation of each from the training set, and then subtract this mean and divide by the standard deviation before using values in Equation 1.

Having defined *influence*, there are now three ways that kMIN might use it: it can use it to fetch neighbours; it can use it when aggregating target values/classes of neighbours; or it can use it for both.

3.1.1. Using Influence to Fetch

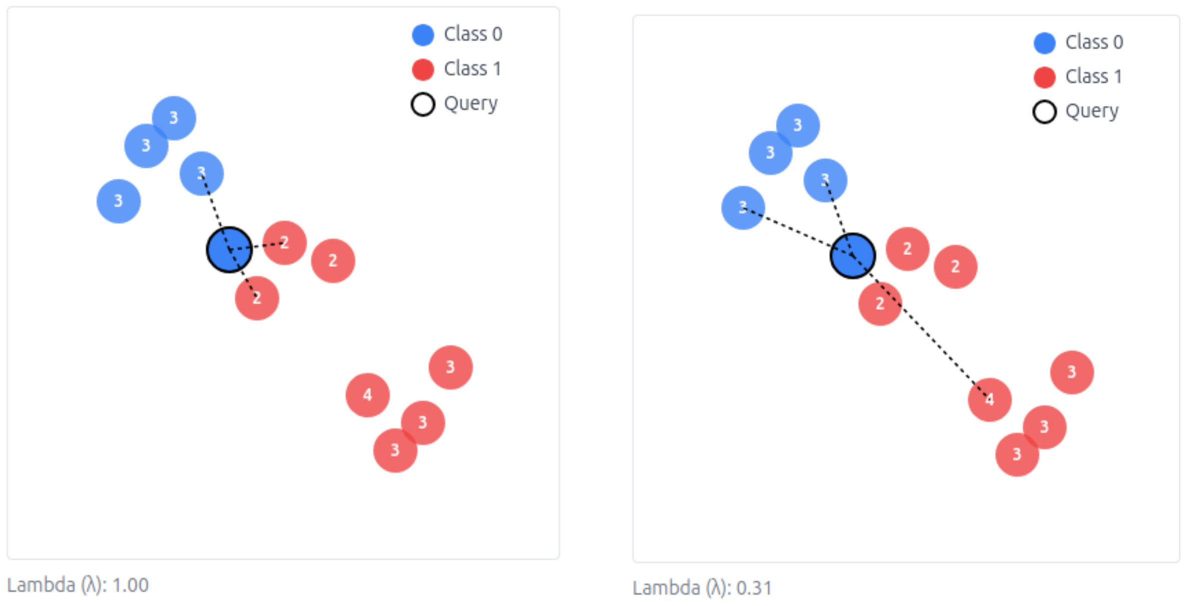
For a given query example $\mathbf{x}$, kMIN can retrieve the k training examples with highest influence over $\mathbf{x}$. If λ is 0, then only reliability is considered; if λ is 1, then, from the point of view of fetching neighbours, kMIN behaves like normal kNN. Figure 1 helps visualise kMIN neighbour selection with a toy example (for $k = 3$). In Figure 1a, with λ = 1, the algorithm selects neighbours purely based on distance, choosing nearby points, which, in the example, mostly happen to have low reliability scores and incorrectly classifying the query example as Class 1. In Figure 1b, with λ = 0.31, the algorithm balances distance with reliability, selecting more distant but highly reliable points (reliability scores of 3 or more) and correctly classifying the query example as Class 0. This demonstrates how kMIN can overcome local noise by incorporating reliability into neighbour selection.

3.1.2. Using Influence to Aggregate

Alternatively, kMIN can use *influence* when aggregating the target values/classes of the k-nearest neighbours. This is like Weighted-kNN. However, instead of calculating weights using the reciprocal of the distance, the weight of each neighbour is that neighbour's *influence*, as calculated with Equation 1.

3.1.3. Using Influence to Fetch and Aggregate

We can use *influence* in both steps: kMIN can fetch the k most influential training examples and then also weight each neighbour's contribution to the final prediction by its *influence*. In this case, there is no reason to believe that the optimal value for hyperparameter λ should be the same in the two steps. Hence, for influence-based fetching we use Equation 1; and for weighted aggregation, we use the same formula but with λ replaced by a new hyperparameter λ'.



(a) λ = 1.00: Using only similarity (b) λ = 0.31: Favouring reliability over similarity

Figure 1: Visualisation of how kMIN selects neighbours with different λ values (for $k = 3$). The query example has a black outline. Numbers inside points indicate reliability scores. Dashed lines show the selected neighbours.

For a particular dataset, the decision about whether kMIN should use *influence* only for fetching neighbours (with hyperparameter λ), or only for aggregation (with hyperparameter λ'), or for both (with hyperparameters λ and λ') is a matter for model selection, using the validation sets (Section 4).

4. Evaluation

In this section, we present details of an empirical comparison that we have conducted, with the results being in the next section.[1]

4.1. Predictors and Datasets

On classification datasets, we compare standard kNN, Weighted-kNN, RENN+kNN, BBNR+kNN, and our own kMIN. For regression datasets, we do the same, but using our modified versions of RENN and BBNR (Section 2.3).

We use six UCI Machine Learning Repository datasets[2] for classification, and six for regression. We delete examples that have missing values; we standardize numeric-values features using means and standard deviations computed only from training data; and we one-hot encode categorical-valued features.

4.2. Experimental Methodology

We evaluated all algorithms using nested 5-fold cross-validation. In nested cross-validation, each fold is used in turn as the test set, the remainder being the development set. But the development set is itself split into 5 folds, each of these being used in turn as the validation set, the remainder being the training set.

We use the validation sets for model selection and hyperparameter tuning. For classification, we must choose the value of k. We start with the square root of the size of the training set, and then repeatedly try values a small amount both sides of the current value, and we repeat until we home in on

[1]For reproducibility, all source code is available on a public GitHub repository: https://github.com/mattmallencode/NcKNN.
[2]https://archive.ics.uci.edu/

the best value. For kMIN, we decide whether to use *influence* for fetching neighbours, for aggregating neighbours' classes, or for both, and we must set λ and λ' as appropriate (Section 3). We try values of λ and λ' between 0 and 1 in increments of 0.05.

For regression, additionally, our modified versions of RENN and BBNR need values for θ, the agreement threshold. As explained in Section 2.3, θ is defined as α multiplied by either the standard deviation of the neighbours' target values or the training set target values. For α, we try integer values between 1 and 10 inclusive, the same values that were found to be effective in [13]. Regression experiments are therefore 20 times more expensive than classification ones. To make them more feasible, we added stochasticity to the selection of k and, in kMIN, we try values of λ and λ' between 0 and 1 in increments of 0.1 instead of 0.05.

4.3. Statistical Significance Testing

To evaluate algorithm performance, we use accuracy for classification and mean absolute error (MAE) for regression. We apply statistical significance testing at the fold level, comparing performance across 5 cross-validation folds. P-values in Tables 1, 2, 6 and 7 are derived using the Friedman test, which checks for significant differences between algorithms across folds. A P-value below 0.05 indicates at least one algorithm differs significantly, warranting post hoc analysis using the Nemenyi test to determine specific pairwise differences. The Friedman test, a non-parametric alternative to repeated-measures ANOVA, is ideal for comparing multiple algorithms on heterogeneous datasets without assuming normality [15]. If significant differences ($P < 0.05$) are found, the Nemenyi test further identifies which algorithms differ. In Tables 3, 4 and 5, p-values indicate pairwise significance, with lower values ($p < 0.05$) marking significant differences.

4.4. Determining the Noise Level

To test noise resistance, we separately report results, first, where we do not inject artificial noise into the training sets, and, second, where we do inject artificial noise. For classification, we randomly alter the class label of a specified fraction of the training set to another class; in regression, we add Gaussian noise to the target value of a similarly specified subset of the training set.

The levels of noise were progressively increased. We consider noise levels of 10%, 20%, 30%, 40%, and 50%, commonly used in the literature [16]. Evaluating each algorithm at every noise level would multiply the computational effort, given our large hyperparameter search space. Instead, for each dataset, we select the lowest noise level that results in a statistically significant performance degradation, according to a t-test and comparing against the baseline of no noise.

5. Results

5.1. Classification Results

Table 1 shows the accuracy of the algorithms on the datasets without injection of artificial noise. kMIN is competitive throughout, and it is the best performing algorithm on the Car dataset.

If the P-value in Table 1 is below 0.05, then at least one of the algorithms is significantly different from the others. This is only true for the Car dataset. To know which difference or differences are significant, we consult Table 3. In this table, each cell represents the p-value from the comparison of a pair of algorithms (hence symmetric) using Nemenyi tests. A p-value less than 0.05 indicates a statistically significant performance difference. From Table 1, we know that kNN and kMIN outperform RENN; and, from Table 3, we see that these differences are significant.

Table 2 shows the accuracy of the algorithms on the datasets when there is injection of artificial noise. kMIN is now more than competitive: it is the top performer in four of the six datasets, namely: Car, Heart, Votes and Zoo. However, of these four, there is no statistical significance ($P \geq 0.05$) in Heart and Zoo.

Table 1
Accuracy on Classification Datasets without Artificial Noise
(The best performing algorithm for each dataset is highlighted in bold)
(Datasets where at least one algorithm differs significantly from another ($P < 0.05$) have underlined P-values)

Dataset	kNN	W.kNN	RENN	BBNR	kMIN	P-value
Car	0.9109	0.9057	0.8721	0.8947	**0.9178**	<u>0.0015</u>
Heart	**0.8214**	**0.8214**	0.8114	0.7742	0.8012	0.0551
Iris	0.9400	**0.9667**	0.9533	0.9600	0.9600	0.1024
Votes	0.9139	0.9139	0.9098	**0.9315**	0.9226	0.5878
Wine	0.9776	**0.9832**	0.9606	0.9778	0.9492	0.0708
Zoo	0.9105	0.9600	0.9105	**0.9700**	0.9405	0.1104

Table 2
Accuracy on Classification Datasets with Artificial Noise
(The level of noise that we injected is shown in parentheses)
(The best performing algorithm for each dataset is highlighted in bold)
(Datasets where at least one algorithm differs significantly from another ($P < 0.05$) have underlined P-values)

Dataset	kNN	W.kNN	RENN	BBNR	kMIN	P-value
Car (10%)	0.8941	0.8981	0.8646	0.8356	**0.8999**	<u>0.0032</u>
Heart (40%)	0.6673	0.6403	0.6866	0.6266	**0.7473**	0.4702
Iris (50%)	0.7467	0.6667	**0.7733**	0.5533	0.7667	<u>0.0361</u>
Votes (50%)	0.4735	0.5003	0.6463	0.5089	**0.6812**	0.1571
Wine (30%)	0.9270	**0.9381**	0.8937	0.7359	0.9379	<u>0.0116</u>
Zoo (50%)	0.7224	0.6638	0.6929	0.4838	**0.7433**	0.3240

Table 3
Pairwise p-values for the Nemenyi Post Hoc Test Results for the Car Dataset without Artificial Noise
(Significant differences have underlined p-values)

Algorithm	kNN	W.kNN	RENN	BBNR	kMIN
kNN					
Weighted kNN	0.7811				
RENN	<u>0.0120</u>	0.2200			
BBNR	0.2659	0.8946	0.7243		
kMIN	0.9000	0.6108	<u>0.0042</u>	0.1447	

In fact, according to the Friedman test, there are statistically significant differences for Car, Iris and Wine. We investigate these further using pairwise Nemenyi tests. Table 4 shows these for the Car dataset with artificial noise. From Table 2, we know that kNN and kMIN outperform BBNR; and, from Table 4, we see that these differences are statistically significant. We find exactly the same on the noisy Wine dataset: kNN and kMIN outperform BBNR (Table 2) and the differences are significant (Table 5). When it comes to the Iris dataset, despite the Friedman test indicating statistical significance ($P = 0.0361$), the Nemenyi test (which is more sensitive than Friedman) did not reveal any pairwise statistically significant differences; for this reason, we do not include a table of Nemenyi test p-values for the noisy Iris dataset.

5.2. Regression Results

In Table 6, we see how the algorithms compare on the regression datasets without artificial noise. Again kMIN is competitive, bearing in mind that low values for MAE are better. It is never the best performing method. The Friedman test P-values in Table 6 suggest that there is at least one significant difference for the Automobile, Auto MPG and Real Estate datasets ($P < 0.05$). Pairwise Nemenyi tests reveal that: kNN is significantly better than RENN and BBNR for the Auto-MPG dataset; kNN and Weighted-kNN are both significantly better than RENN and BBNR for the Automobile dataset; and on Real Estate,

Table 4
Pairwise p-values for the Nemenyi Post Hoc Test Results for Car Dataset with Artificial Noise
(Significant differences have underlined p-values)

Algorithm	kNN	W.kNN	RENN	BBNR	kMIN
kNN					
W.kNN	0.9000				
RENN	0.3172	0.0904			
BBNR	0.0904	<u>0.0166</u>	0.9000		
kMIN	0.9000	0.9000	0.1795	<u>0.0409</u>	

Table 5
Pairwise p-values for the Nemenyi Post Hoc Test Results for Wine Dataset with Artificial Noise
(Significant differences have underlined p-values)

Algorithm	kNN	W.kNN	RENN	BBNR	kMIN
kNN					
W.kNN	0.9000				
RENN	0.8378	0.6676			
BBNR	0.0705	<u>0.0306</u>	0.4971		
kMIN	0.9000	0.9000	0.6676	<u>0.0306</u>	

Table 6
MAE on Regression Datasets without Artificial Noise
(The best performing algorithm for each dataset is highlighted in bold)
(Datasets where at least one algorithm differs significantly from another ($P < 0.05$) have underlined P-values)

Dataset	kNN	W.kNN	RENN	BBNR	kMIN	P-value
Automobile	**0.4657**	**0.4657**	0.7356	0.7299	0.6601	<u>0.0011</u>
Auto MPG	**0.1225**	**0.1225**	0.1761	0.1817	0.1700	<u>0.0014</u>
Liver	2.4089	**2.3297**	2.3991	2.3588	2.3504	0.0527
Real Estate	5.7758	**5.3359**	5.6789	5.7766	5.7107	<u>0.0250</u>
Student (M)	**3.2467**	3.2612	3.2531	3.3563	3.2888	0.0652
Student (P)	2.0946	**2.0816**	2.0841	2.0855	2.0979	0.1882

Table 7
MAE on Regression Datasets with Artificial Noise
(The level of noise that we injected is shown in parentheses)
(The best performing algorithm for each dataset is highlighted in bold)
(Datasets where at least one algorithm differs significantly from another ($P < 0.05$) have underlined P-values)

Dataset	kNN	W.kNN	RENN	BBNR	kMIN	P-value
Automobile (40%)	0.7809	0.7375	0.7577	0.7273	**0.7163**	0.5781
Auto MPG (10%)	0.2297	0.2297	0.2560	0.2575	**0.2008**	0.1249
Liver (50%)	2.3524	2.3547	2.4375	2.4298	**2.3392**	0.1933
Real Estate (50%)	6.1441	**5.9541**	6.2349	6.0926	6.0859	0.2598
Student (M,40%)	3.3037	3.3322	3.3578	3.3698	**3.2702**	0.1712
Student (P,10%)	2.1110	2.0986	2.1254	**2.0858**	2.1089	0.1424

Weighted-kNN is significantly better than BBNR. In no case, was kMIN significantly better or worse than the others. Given that kMIN is the focus of this paper and was not involved in any significant differences on these datasets, we have elected not to include the tables of Nemenyi p-values.

Table 7 shows the MAE of the algorithms on the datasets when there is injection of artificial noise. kMIN is now more than competitive: it is the top performer in four of the six datasets, namely: Automobile, Auto MPG, Liver and Student (M). However, in no case was there statistical significance according to the Friedman test ($P \geq 0.05$ in all cases) and so there is no reason to show tables of Nemenyi p-values.

6. Conclusions, Limitations and Future Work

6.1. Conclusions

Our empirical comparison reveals several key insights. In noise-free environments, kNN and Weighted-kNN consistently demonstrated robust performance across classification and regression tasks. The dataset editing algorithms, RENN and BBNR, did not consistently outperform the simpler kNN variants irrespective of whether we injected noise or not.

kMIN showed promise in handling noise for both classification and regression tasks, being the top performer in four of six datasets for both tasks, when we introduced artificial noise. However, in only a few cases were the results statistically significant. The post hoc analysis revealed that kMIN was statistically significantly better than BBNR for two datasets.

Overall, the findings are not encouraging for the dataset editing algorithms that we have tried — at least on these datasets. On the other hand, the results do hint at kMIN's potential in noisy data conditions (especially given it was the top performer on eight of twelve experiments where we injected artificial noise). However, given the lack of convincingly statistically significant results, further investigation is warranted to understand its advantages and fully optimise its performance. In particular, we need experiments on larger datasets, where it may be easier to establish statistical significance (Section 6.2.1).

As things stand, our findings show the importance of selecting the appropriate algorithm, based on the specific context of use, particularly the presence of noise in the data.

6.2. Limitations

This section summarises what we believe to be the limitations inherent to our approach and potential methods of overcoming them.

6.2.1. Statistical Significance

Pursuing statistical significance in our experimental results faced several challenges, limiting the robustness of our findings. Key among these was the scale and scope of the hyperparameter search space. To enhance the likelihood of obtaining statistically significant outcomes, we propose three strategies:

- **Increased Sample Size:** Expanding the number of datasets (especially if we can include larger datasets) or replicating experiments across more folds could enhance the statistical power of our analyses, making it easier to detect actual differences between algorithms.
- **Refined Hyperparameter optimisation:** Implementing more sophisticated hyperparameter search techniques, such as Bayesian optimisation, could lead to better-tuned models, thereby increasing the performance differential. As highlighted by Turner et al. [17], employing Bayesian optimisation for hyperparameter tuning can significantly enhance model performance over traditional search methods.
- **Alternative Statistical Tests:** Employing more sensitive statistical tests or adjusting significance levels may uncover subtle but meaningful differences between models.

While potentially increasing the computational complexity or resource demands of the research, these approaches could significantly improve the clarity and robustness of our conclusions regarding the comparative performance of the algorithms.

6.2.2. Diversity of Error Estimates

A limitation of our research is the reliance on a single error metric (MAE for regression and accuracy for classification) to evaluate algorithm performance. This approach may only partially capture the nuanced behaviours and strengths of the various algorithms under different conditions. For instance, metrics such as the root mean squared error (RMSE) could provide additional insight into regression performance by penalising significant errors more heavily. Similarly, precision, recall, and the F1 score could offer a more detailed understanding of performance in classification tasks, especially in imbalanced datasets where accuracy may not fully reflect an algorithm's effectiveness.

6.2.3. Different Kinds of Noise

Our approach to introducing artificial noise changed the target values and class labels, which may not fully capture the algorithm's performance under other noisy conditions found in real-world data, including feature noise, outliers, systematic noise, and missing values that may influence the algorithms differently. Moreover, our relatively clean and well-curated UCI Machine Learning Repository datasets could provide relatively weak indications of performance in the face of the challenges of inherently noisy real-world datasets.

6.3. Future Work

This section summarises potential future avenues of research that may merit exploration.

6.3.1. Incremental Reliability Calculations

In dynamic environments, new training data is likely to become available over time. The new training examples would require reliability scores and, especially in cases of concept drift, it is likely that the new training data would invalidate some of the reliability scores previously calculated. Calculating reliability scores incrementally could help address these problems in a more computationally efficient manner. Rather than re-evaluating reliability scores for the entire dataset, the algorithm could pinpoint and update only the scores of directly affected examples, significantly enhancing efficiency.

6.3.2. Evaluating Computational Efficiency

While we believe kMIN adheres closer to the lazy learning model than the editing algorithms, it still introduces a pre-processing step, not present in standard kNN, viz. the calculation of reliability scores, and the standardization of similarity and reliability scores. It would be interesting to compare the cost of kMIN's pre-processing with the cost of the dataset editing algorithms, bearing in mind that the dataset editing algorithms, by deleting training examples, thereby reduce inference costs.

6.3.3. Other Definitions of Reliability

An intriguing area for future exploration involves enhancing the reliability scores used in kMIN. One potential avenue of research would be using some invrese of the size of each example's *Liability Set* as the basis for these scores as opposed to the size of the *Coverage Set*, or even some combination of the two. Another possibility could be to use a recurrence relation, akin to PageRank [18], as proposed in [19]. This adaptation would assess an example's reliability in terms of the reliability of its neighbours.

Acknowledgments

This work has been supported by a grant from Science Foundation Ireland (SFI) under Grant Number 12/RC/2289-P2, which is co-funded under the European Regional Development Fund. We are grateful to the anonymous reviewers for their comments.

References

[1] E. Fix, J. L. Hodges, Discriminatory analysis - nonparametric discrimination: Consistency properties, International Statistical Review 57 (1989) 238–247. doi:10.2307/1403797, (Originally

published as Report Number 4, Project Number 21-49-004, USAF School of Aviation Medicine, Randolph Field, Texas, 1951).

[2] R. Halder, M. Uddin, M. Uddin, Enhancing K-nearest neighbor algorithm: A comprehensive review and performance analysis of modifications, Journal of Big Data 11 (2024) 1–25. doi:10.1186/s40537-024-00973-y.

[3] S. Uddin, I. Haque, H. Lu, M. A. Moni, E. Gide, Comparative performance analysis of K-nearest neighbour (KNN) algorithm and its different variants for disease prediction, Sci. Rep. 12 (2022). doi:10.1038/s41598-022-10358-x.

[4] S. A. Dudani, The distance-weighted k-nearest-neighbor rule, IEEE Transactions on Systems, Man, and Cybernetics SMC-6 (1976) 325–327. doi:10.1109/TSMC.1976.5408784.

[5] J. H. Friedman, J. L. Bentley, R. A. Finkel, An algorithm for finding best matches in logarithmic expected time, ACM Trans. Math. Softw. 3 (1977) 209—-226. doi:10.1145/355744.355745.

[6] J. W. Schaaf, Fish and shrink. a next step towards efficient case retrieval in large-scale case bases, in: Proceedings of the Third European Workshop on Advances in Case-Based Reasoning, Springer-Verlag, 1996, pp. 362—-376.

[7] D. W. Aha, Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms, International Journal of Man-Machine Studies 36 (1992) 267–287. doi:https://doi.org/10.1016/0020-7373(92)90018-G.

[8] I. Tomek, An experiment with the edited nearest-neighbor rule, IEEE Trans. Syst. Man Cybern. SMC-6 (1976) 448–452. doi:10.1109/TSMC.1976.4309523.

[9] D. L. Wilson, Asymptotic properties of nearest neighbor rules using edited data, IEEE Transactions on Systems, Man, and Cybernetics SMC-2 (1972) 408–421. doi:10.1109/TSMC.1972.4309137.

[10] S. J. Delany, P. Cunningham, An analysis of case-base editing in a spam filtering system, in: Procs. of the Seventh European Conference omn Case-Based Reasoning, 2004, pp. 128–141. doi:10.1007/978-3-540-28631-8_11.

[11] F.-X. Pasquier, S.-J. Delany, P. Cunningham, Blame-based noise reduction: An alternative perspective on noise reduction for lazy learning, Computer Science Technical Report TCD-CS-2005-29 (2005).

[12] A. P. Parsodkar, P. Deepak, S. Chakraborti, Never judge a case by its (unreliable) neighbors: Estimating case reliability for CBR, in: Case-Based Reasoning Research and Development, Springer International Publishing, Cham, 2022, pp. 256–270.

[13] M. Kordos, M. Blachnik, Instance selection with neural networks for regression problems, in: Artificial Neural Networks and Machine Learning – ICANN 2012, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 263–270.

[14] J. Martín, J. A. Sáez, E. Corchado, On the regressand noise problem: Model robustness and synergy with regression-adapted noise filters, IEEE Access 9 (2021) 145800–145816. doi:10.1109/ACCESS.2021.3123151.

[15] J. Demšar, Statistical comparisons of classifiers over multiple data sets, Journal of Machine Learning Research 7 (2006) 1–30.

[16] R. Hasan, C. H. Chu, Noise in datasets: What are the impacts on classification performance?, in: International Conference on Pattern Recognition Applications and Methods, 2022, pp. 163–170.

[17] R. Turner, D. Eriksson, M. J. McCourt, J. Kiili, E. Laaksonen, Z. Xu, I. M. Guyon, Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020, in: Neural Information Processing Systems, 2021, pp. 3–26. Conference Proceedings.

[18] S. Brin, L. Page, The anatomy of a large-scale hypertextual web search engine, Computer Networks and ISDN Systems 30 (1998) 107–117. doi:10.1016/S0169-7552(98)00110-X, proceedings of the Seventh International World Wide Web Conference.

[19] A. P. Parsodkar, D. P., S. Chakraborti, The case for circularities in case-based reasoning, in: S. Massie, S. Chakraborti (Eds.), Procs. of the 31st International Conference on Case-Based Reasoning, Springer, 2023, pp. 85–101.