ImpCML Kryptonite-N Coursework Report

JJ Lu-Hilson, Jin Wang, Ketong Ding, Xiwei Gao [1]

Abstract

The Kryptonite-n datasets were created to challenge the idea that machine learning models can be thought of as universal function approximators, and solve any challenge presented. The purpose of this paper is to refute the claim that machine-learning models cannot solve this binary classification problem, and in doing so support the claim that machine-learning models can be thought of as universal function approximators. In this paper, different machine learning models are explored to characterise this dataset, with each one being compared in terms of technical accuracy and complexness. It is found that with a relatively simple clustering model and some dimensionality reduction, a machine learning model can be generated which very accurately classifies new untested data, well above the established target accuracies.

1. Introduction

The problem of solving challenge datasets such as Kryptonite-n, which are created with the purpose of disproving the effectiveness of machine learning techniques, is an important one. If such problems cannot be solved then there is a case to be made against the effectiveness of machine learning techniques in general, and hence against the universal approximation argument (Hornik, 1991).

The aim of this paper is to find suitable solutions to the Kryptonite-n challenge datasets by utilising machine learning techniques hence proving that machine learning can be effectively used to solve these challenging datasets, providing evidence to support the universal approximation argument whilst refuting the claim that machine learning cannot solve the Kryptonite-n datasets (Quinn & Luther, 2024).

[1]Department of Computing, Imperial College London, London, United Kingdom. Correspondence to: JJ Lu-Hilson <jj.lu-hilson21@imperial.ac.uk>, Jin Wang <jin.wang24@imperial.ac.uk>, Keton Ding <keton.ding24@imperial.ac.uk>, Xiwei Gao <xiwei.gao20@imperial.ac.uk>.

Coursework. For review by instructors of Mathematics for Machine Learning (70015) . Do not distribute.

To do this, multiple experiments were performed to test hypotheses about which machine learning techniques will be able to best model the data in the Kryptonite-n datasets. Two overall architectures were implemented for classifying these datasets: a k-nearest neighbours classifier, and a multi-layer perceptron classifier.

Overall, the k-nearest neighbours classifier (with some careful manipulation of data) can classify these challenge datasets with much higher accuracy than the target thresholds provided. The multi-layer perceptron can also do so for lower dimensions, but struggles to do so for higher dimensional data, likely due to poor hyper-parameter choices. The highly successful performance achieved by the k-nearest neighbours classifier directly refutes the claim that machine learning models cannot perform well on the Kryptonite-n datasets.

2. Methodology

2.1. KNN

Given the lack of success of the logistic regression approach outlined in the Kryptonite-n paper, it would seem there is no continuous function that can separate the binary labels of this dataset with any level of success much better than that of random guessing. Based off of this knowledge, it would appear this dataset contains a more complex relationship between the data-points and corresponding labels. In cases like this, a clustering approach to training can work well. In the works done for this paper, the K-Nearest Neighbours (k-NN) model was used. This model classifies a data-point by assigning it the same label as the majority of the K-Nearest training data-points. This is a good approach to this challenge given that there is no apparent function that represents the trends in labels relative to the dataset, so instead we simply assign labels to data-points by directly comparing it to the training data.

The mathematical model for the k-NN approach to training looks as follows for assigning an output y given input x_{input}:

$$y = \max_j(p(y = j|X = x_{input})) = \max_j(\frac{1}{K}\sum_{i\in A} I(y^{(i)} = j))$$

where A is the set of K-nearest points to x_{input} and j can

take either of the values $(0, 1)$.

The points in A are found by finding the first K points satisfying:

$$x_K = \min_x(d(x_{input}, x)) \; \forall \; x \in X$$

where X is the set of training data-points, and $d(a, b)$ is the function for finding the distance between vectors a and b using a chosen distance metric, which for the purpose of this paper will be the Euclidean distance metric. The hyper-parameter being trained with this model is the value of K - how many nearest neighbours to compare our input point to.

A further improvement was made to this model, by utilising dimension reduction. With high dimensional data, the k-NN model encounters something known as the "curse of dimensionality" (Beyer, 1999), where the Euclidean distance becomes unhelpful due to many vectors being almost equidistant to the query vector. Hence to improve the model, dimension reduction was used as a pre-processing technique to remove the features providing the least information (with the least variance) from the training dataset, allowing the k-NN method to utilise what is essentially a more informative distance metric. This was done by selecting features with variance below some threshold level (a trainable hyper-parameter, with the possible values being the variances of the features present in the training data) to be removed from the data, allowing the k-NN model to be trained without this dimensionality problem on the most informative features. When the model is applied to unseen data, the same features are removed (not features with variance below the same threshold, as this might apply to different features in the unseen data, hence changing the model).

2.2. Multi-layer Perceptron

2.2.1. Outline

Another model was implemented to attempt to classify the data accurately: the Multi-Layer Perceptron (MLP). This is a type of feed-forward neural network, consisting of layers of fully connected neurons with non-linear activation functions (such as relu, tanh, and the logistic function). This model was chosen in the hopes that it could characterise the complex non-linear trends in data present in the Kryptonite-n dataset, utilising these non-linear activation functions in the hidden layers to do so.

In general, the mathematical model for an MLP network is as follows, generating predicted the label vector y (shape [number of input data-points, 1]) given the input matrix X (shape [number of input data-points, number of features]):

$$\begin{aligned} z^{(1)} &= X \\ z^{(i+1)} &= \sigma_i(W^{(i)} z^{(i)} + b^{(i)}) \\ y = z^{(L)} &= \sigma^{(L-1)}(W^{(L-1)} z^{(L-1)} + b^{(L-1)}) \end{aligned}$$

where $\sigma^{(i)}$, $W^{(i)}$ and $b^{(i)}$ respectively represent for the i-th layer of the model: the selected activation function, the weight matrix (shape [width of layer i, width of layer $(i-1)$]) and bias vector (shape [width of layer i, 1]). The width of each layer is a trainable hyper-parameter, except for the first and last layers for which the dimensions are fixed to match the shape of the input and the output (number of inputs will be equal to the number of outputs as the model will produce one binary output for each input data-point). Similarly, the total number of layers L (L includes the input as a layer, else the output would be $y = z^{(L-1)}$) and the activation functions used within the hidden layers are also trainable hyper-parameters to be tuned, except for the last layer which will always use the logistic function such that it produces a scalar output in the range [0,1] which can then be rounded to give the binary classification.

The weights and biases for each layer of the model are initialised randomly from the uniform distribution $W \sim U(-\sqrt{\frac{6}{width_{in}+width_{out}}}, +\sqrt{\frac{6}{width_{in}+width_{out}}}$ (the normalised initialisation (Glorot & Bengio, 2010)) before the training starts such that each neuron can focus on learning different trends in data and are not naturally biased towards a specific input. This allows the opportunity for a better characterisation of the dataset, hopefully reducing the final loss. The loss function of this model is the binary cross-entropy as Kryptonite-n's outputs are binary (0,1).

2.2.2. Gradient Descent

Adjusting the MLP model's weights and biases to minimise its loss for a training dataset is essentially an optimisation problem with a multi-variable differentiable function, hence gradient descent can be applied. The weights and biases are joined into one matrix, $\theta^{(i)} = \{W^{(i)}, b^{(i)}\}$ for layer i, for the working shown throughout the rest of this section. The parameters used in the first step of gradient descent, $\theta_0^{(i)} \; \forall \; i \in [1, L]$, are randomly initialised via the method discussed earlier and approach the true minimum step-by-step, with the values being adjusted along the negative gradient of the loss function (with respect to that parameter), ensuring $\mathcal{L}(\theta_0) \geq \mathcal{L}(\theta_1) \geq \cdots$ where $\mathcal{L}(\theta_i)$ is the loss function evaluated with parameters of gradient descent step i: θ_i.

For complex models with many parameters that depend on each other (such as the ones trained for the Kryptonite-n datasets), it is very inefficient to calculate the partial derivatives of the loss with respect to the parameters of each layer individually. To improve performance, the reverse-mode au-

tomatic differentiation algorithm is used, utilising the chain rule and exploiting the fact that the partial derivatives for each layer can be used to calculate the partial derivatives of earlier layers, avoiding unnecessary calculations (Griewank & Walther, 2008).

For each step in gradient descent, the predicted labels are generated for the current parameters (θ_i in the i-th step of the algorithm). This is the forward pass, and the derivative of the output of each layer (with respect to its parameters) through this calculation is stored for use in the calculation of gradients later on (for example a logistic activation function will have the derivative $\sigma(z)' = \sigma(z)(1-\sigma(z))$ where $\sigma(z)$ is the output from this function in the forward pass).

The derivative of the loss with respect to the input is:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial x} &= \frac{\partial \mathcal{L}}{\partial y} \cdot \frac{\partial y}{\partial z^{(L-1)}} \cdot \frac{\partial z^{(L-1)}}{\partial z^{(L-2)}} \cdots \frac{\partial z^{(2)}}{\partial x} \\ &= \frac{\partial \mathcal{L}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial z^{(L-1)}} \cdots \frac{\partial z^{(2)}}{\partial z^{(1)}}\end{aligned}$$

due to chain rule, where $\dfrac{\partial \mathcal{L}}{\partial y}$ is the derivative of the loss function with respect to the output label vector (the binary cross-entropy loss function).

Similarly, the derivative of the loss with respect to the parameters of layer i, $\theta^{(i)}$ is:

$$\nabla_{\theta^{(i)}} \mathcal{L}(f^\theta, X, y) = \frac{\partial \mathcal{L}}{\partial \theta^{(i)}} = \frac{\partial \mathcal{L}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial z^{(L-1)}} \cdots \frac{\partial z^{(i+1)}}{\partial \theta^{(i)}}$$

The first term is once again the binary cross-entropy, which will be passed down to the previous layer. The final term $(\frac{\partial z^{(i+1)}}{\partial \theta^{(i)}})$ is the derivative of the output of this layer with respect to its parameters, which was stored during the forward pass. Each layer then passes down their corresponding partial derivative, allowing for the gradients of each layer to be calculated in L steps. Including the forward pass, each step of gradient descent is $O(LT)$ where L is the number of layers and T is the time spent calculating these values.

The binary cross-entropy loss is calculated as follows:

$$-\frac{1}{N}\sum_{i=1}^{N}[y_i log(p_i) + (1-y_i) log(1-p_i)]$$

where y_i is the actual label value, N is the size of the output label vector (which is equal to the amount of input data-points), and p_i is the assigned probability of that point being classified as a 1 (the output for that point before being rounded to a binary classification).

The weights of layer i are then updated according to the gradient update equation:

$$\theta_{j+1}^{(i)} = \theta_j^{(i)} - \alpha \nabla_{\theta^{(i)}} \mathcal{L}(f^\theta, X, y)$$

where $\theta_j^{(i)}$ is the weight matrix of layer i on the j-th step of gradient descent, and α is the learning rate.

The learning rate, α, is not a constant and instead is updated in each step of gradient descent using the Adam algorithm (Kingma & Ba, 2017), which optimises the learning rate individually for each parameter using momentum and Root Mean Square Propagation (RMSProp).

The other automatic differentiation algorithm, forward-mode, is another viable option though it requires one iteration of the gradient descent algorithm per input feature, whereas the reverse-mode algorithm requires one iteration per output label. Since the model produces a one-dimensional label for each data-point, and each input data-point has dimensions in the range [9, 45], reverse-mode presents itself as the ideal choice with a much lower complexity.

2.3. Data Pre-processing

For the k-nearest neighbours model, the only data pre-processing operation that is applied is the removal of features with low variance, as discussed in section 2.1.

For the multi-layer perceptron model, the features are scaled such that the transformed data has a mean of 0 and variance of 1. This scaling is done to train the model fairly for each input feature due to the model being sensitive to the mean and variance of each feature, with features of higher mean and variance being prioritised by the model and altering the overall fit.

2.4. Auto Hyper-Parameter Searching

2.4.1. K-NN

For the k-NN model, the only hyper-parameters being searched over are the number of neighbours (k) and the variance threshold at which to remove features. The number of neighbours is chosen within the range $[1, 30]$, with only the odd values being searched so that the unlikely case of points balancing out and no classification being made is avoided. This search range produced good results within a relatively small search grid, which is computationally ideal. The variance threshold is determined by the variance of the features such that a variance of 0 (all features retained) is searched, as well as searching each unique variance level. There are only two hyper-parameters, so the grid-search is used.

2.4.2. Multi-Layer Perceptron

For the MLP model, there are many hyper-parameters to be tuned, so a grid-search is not practical as the time to train the model would be enormous, especially due to both the learning rate and momentum taking values in a continuous

Hyper Parameter	Selection Values
Number of neurons in each hidden layer (discrete)	[4,16]
Number of hidden layers (discrete)	[2,5]
Activation function for hidden layers	{relu, tanh}
Model learning rate (continuous)	[1e-4, 1e-2]
Gradient descent momentum (continuous)	[0.1, 0.9]

Table 1. Hyper-parameters tuned for the MLP model

range, which is inherently difficult to grid-search over. To optimise the hyper-parameter tuning, the Ray Tune library (Liaw et al., 2018) is used, which mobilises Optuna's search algorithm (Akiba et al., 2019) to search for the optimal hyper-parameters. This algorithm is given a time budget of 600 seconds for each of the Kryptonite-n datasets. The hyper-parameters affected by the search are listed in table 1.

3. Experimental Design

Note: For all of these experiments, the data is first partitioned into an 80/20 train-test split, with each model being trained on the same training split (randomly generated using a fixed seed) for fairness. Data pre-processing techniques are generated using the training data (for example the scaling applied to each feature in the MLP model). The models are then evaluated on the test split, with any data pre-processing techniques being applied in the same way as they are for the training split.

3.1. k-NN Experiments

3.1.1. Experiment 1: K-NN Performance

Hypothesis: A clustering approach, such as k-Nearest Neighbors (k-NN), to classifying the dataset will work well given the lack of apparent logistic trends in data.

Experiment Details: The hypothesis was tested by implementing a simple k-NN model and training it on the Kryptonite-n data-sets. A grid-search for the value of k is then performed, using 5-fold cross-validation to find the optimal value. For each dataset, the optimal k obtained from the grid-search was then selected to construct the final k-NN model. This final model was then evaluated on the test set, with the results being plotted in comparison to the provided target accuracies.

Validation of Results: To ensure the results are valid, the final models were trained 10 times using different random seeds for the train-test split, with the average accuracy and standard deviation being calculated across these 10 repetitions. Also, the error bars were included in performance plots to visually show confidence intervals and represent how much results can vary.

3.1.2. Experiment 2: K-NN with feature selection

Hypothesis: Implementing a form of dimensionality reduction before running the k-NN algorithm will improve performance by avoiding the "curse of dimensionality" (Beyer, 1999)

Experiment Details: To test this hypothesis, the same implementation of the k-NN algorithm was kept, but features were removed from the data-set depending on their variance level. To implement this, the variances of each feature are recorded, then iterated through. In each iteration, features with variance below the threshold are dropped, and then the experiment in 3.1.1 is performed again, expanding the grid-search. The final k-NN model trained on the reduced feature set was then compared to the k-NN model trained on the full feature set.

Validation of Results: To ensure the results are valid the same features are removed in the test data as the training data, rather than simply removing features below the same variance threshold, since there is no guarantee of these datasets having the same variance throughout their features. To ensure valid comparisons, both the reduced-feature k-NN model and the full-feature k-NN model were trained 10 times using the same set of random seeds as experiment 3.1.1 for the train-test split. The average accuracy and standard deviation were calculated across these 10 repetitions, and the error bars were included in performance plots to visually show confidence intervals and represent how much results can vary.

3.1.3. Experiment 3: Distance Weighted k-NN

Hypothesis: In the Kryptonite-n dataset, points are clustered in a way such that the training points in the feature space which are closest to an input data-point, should be more likely to share the same label, hence making distance-weighted k-NN a better choice than k-NN with uniform weighting.

Experiment Details: The same experiment outlined in section 3.1.2 was performed, but using a weighting which is inversely proportional to the distance of the neighbour from the input data-point (Dudani, 1976). Results were compared to those generated by the experiment in section 3.1.2.

Validation of Results: To ensure valid comparisons, both the weighted and uniform k-NN models were trained 10 times using the same set of random seeds as experiment 3.1.1

for the train-test split. The average accuracy and standard deviation were calculated across these 10 repetitions, and the error bars were included in performance plots to visually show confidence intervals and represent how much results can vary.

3.2. Multi-Layer Perceptron Experiments

3.2.1. Experiment 4: MLP performance

Hypothesis: A multi-layer perceptron model should characterise the Kryptonite-n datasets well, with respect to the target accuracies provided. The use of non-linear activation functions and multiple layers should enable the network to capture the complex trends in data present in the Kryptonite-n datasets.

Experiment Details: The hypothesis was tested by creating a multi-layer perceptron model, and training this on the Kryptonite-n datasets. Algorithms discussed in section 2.4.2 were implemented to efficiently search the hyper-parameters in Table 1. The best model found by the algorithm was then evaluated on the test set, with the results being plotted in comparison to the provided target accuracies.

Validation of Results: The best models for each data-set were trained 10 times using different random seeds for the train-test split. The average accuracy and standard deviation were calculated across these 10 repetitions, and the error bars were included in performance plots to visually show confidence intervals and represent how much results can vary.

4. Experimental Results

4.1. Experiment 1: k-NN Performance

Figure 1 shows that the basic k-NN algorithm performs well (just about on par with the provided target accuracies) up to dimension 12, but the accuracy then drops quickly as the number of dimensions increases. This shows that the hypothesis for this experiment was not fully accurate, driving experiment 3.1.2.

4.2. Experiment 2: k-NN with Feature Selection

Figure 2 shows that utilising dimensionality reduction significantly improves accuracy for the higher dimensional datasets, such that the final performance is well above the provided target accuracies, confirming this experiment's target hypothesis.

4.3. Experiment 3: Weighted k-NN

Figure 3 shows that the k-NN model accuracies do not noticeably change between using uniform weightings and

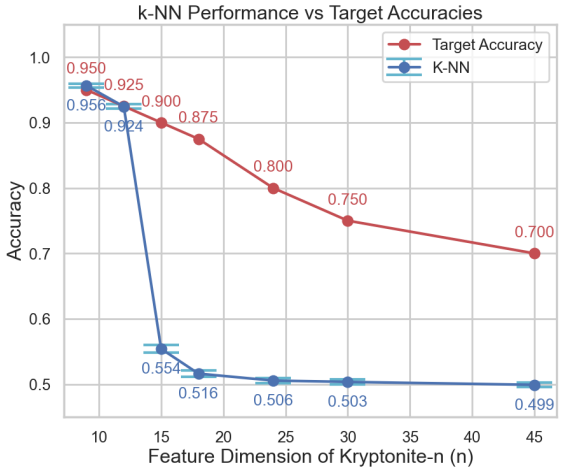

Figure 1. Experiment 1: k-NN Performance vs. Dimensionality.

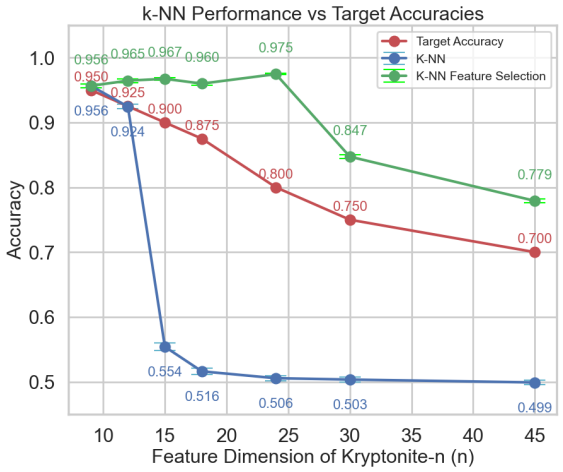

Figure 2. Experiment 2: Comparison of k-NN Performance with and without Feature Selection.

distance-based weightings. This rejects the experiment's hypothesis, possibly due to their being such a dense clustering of points in the feature space that the weightings end up approximately uniform anyway. Due to this, uniform weighting is used in our final model since it performs the same but is less computationally expensive.

4.4. Experiment 4: MLP performance

Figure 4 illustrates that with some level of hyper-parameter tuning, the Multi-Layer Perceptron model can accurately predict the labels of unseen data-points consistently for low-dimensional data (as seen by the tight error bars). For higher-dimensional data the model does not perform well, most likely due to ideal hyper-parameters not having been found yet (due to resource constraints). If the models were allowed to hyper-train for longer, and on broader search spaces, a network capable of characterising the complex

trends in the Kryptonite-n datasets might be formed.

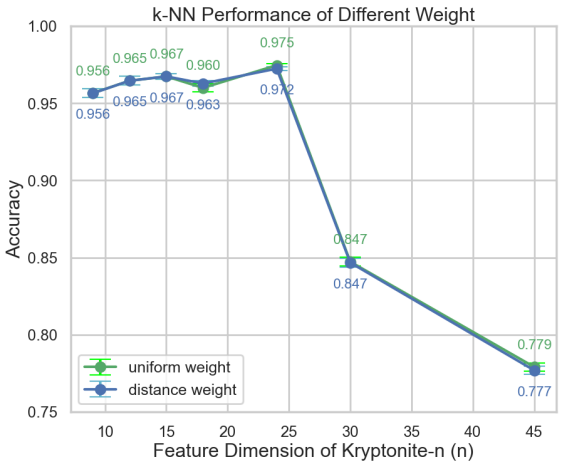

Figure 3. Experiment 3: Comparison of k-NN Performance with different weight.

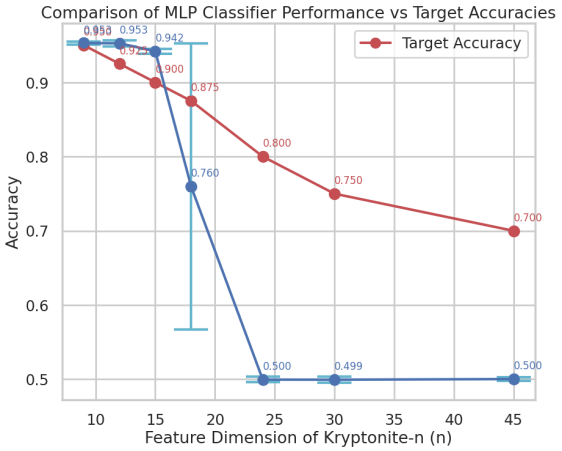

Figure 4. Experiment 4: Comparison of MLP Performance Against Target Accuracies.

5. Discussion

5.1. Conclusions

In this paper, two types of machine learning models were implemented to attempt to classify the Kryptonite-n datasets. One (k-NN) was successful to a high degree, with accuracies well above the provided targets for all dimensions. The other (MLP) was not as successful, but shows potential for producing highly accurate models, and still performs well enough to reject the claims made by Quinn and Luther (2024).

The success of the k-NN models showcase that the Kryptonite-n datasets are able to be classified with relatively simple machine learning models. While the k-NN method is not typically used to support the universal function approximation argument, it can still be represent a universal function approximator, with the functions being defined by the training points. This refutes the claim made by Quinn and Luther (2024) that the Kryptonite-n datasets best the universal function approximation argument.

5.2. Potential Improvements

Due to limited available computational resources, the MLP was not able to be trained to accurately fit high-dimensional Kryptonite-n datasets. With availability of faster processors and more time, it would have been ideal to run a broader and deeper search to see if the higher-dimensional datasets could be accurately modelled using a neural network.

The computational intenseness of the full grid-search undergone to train the k-NN model could have been significantly reduced using an early stopping algorithm to prevent unnecessary feature removals. Once a large enough portion of features are removed, removing features no longer aids the k-NN algorithm, instead removing useful information and degrading the overall performance.

Ideally, an in-depth analysis of the machine learning architectures used in the paper by Quinn and Luther (2024) would have been performed, with explanations of why these methods failed to accurately characterise the data-set. In the process, potential improvements may have been found, providing more evidence for the universal function approximation argument.

5.3. Impact assessment

The estimated carbon emissions generated by training the machine learning models created for this paper were tracked using the CodeCarbon library (Courty et al., 2024). The results are provided in table A1 in the appendix. These results show that training models with known hyper-parameters is relatively cheap (emitting in the order of 1e-04kg CO_2 or less). It also shows the large impact that intense hyper-training experiments have on the environment, with the emissions produced increasing by multiple orders of magnitude.

To put this in perspective however, the most demanding experiment ran (the full search of the k-NN hyper-parameter space), consumed energy equivalent to charging 2.4 smartphones (using the tool at https://www.epa.gov/energy/greenhouse-gas-equivalencies-calculator) and produced CO_2 emissions equivalent to burning 0.02 pounds of coal. These amounts in the grand scheme of things are relatively small but should not be ignored, thus any further optimisations to the experiments that reduce the computational resource requirements are beneficial both to the environment and to researchers.

References

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

Beyer, K. e. a. When is "nearest neighbor" meaningful? In *Database Theory—ICDT'99*, pp. 217–235, University of Wisconsin, 1999.

Courty, B., Schmidt, V., Luccioni, S., Goyal-Kamal, MarionCoutarel, Feld, B., Lecourt, J., LiamConnell, Saboni, A., Inimaz, supatomic, Léval, M., Blanche, L., Cruveiller, A., ouminasara, Zhao, F., Joshi, A., Bogroff, A., de Lavoreille, H., Laskaris, N., Abati, E., Blank, D., Wang, Z., Catovic, A., Alencon, M., Stechły, M., Bauer, C., de Araújo, L. O. N., JPW, and MinervaBooks. mlco2/codecarbon: v2.4.1, May 2024. URL https://doi.org/10.5281/zenodo.11171501.

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

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M. (eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pp. 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL https://proceedings.mlr.press/v9/glorot10a.html.

Griewank, A. and Walther, A. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, Second Edition*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 2008. ISBN 9780898717761. URL https://books.google.co.uk/books?id=xoiiLaRxcbEC.

Hornik, K. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(91)90009-T. URL https://www.sciencedirect.com/science/article/pii/089360809190009T.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E., and Stoica, I. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

Quinn, H. and Luther, L. Kryptonite-n: A simple end to machine learning hype? Technical report, Computer Science Department, Rutgers University, 2024.

Table A1. Emissions Produced By Experiments

Experiment Run	**Energy Used** (kWh)	**Emissions** (CO_2, kg)
k-NN search for k	2.639e-04	6.270e-05
k-NN search for k with feature reduction	7.513e-02	1.785e-02
k-NN with optimized k & feature reduction	4.466e-03	1.061e-05
MLP search for hyper-parameters	4.896e-02	1.163e-02
MLP with optimized hyper-parameters	2.034e-02	4.833e-04