PAPER • OPEN ACCESS

Logic replicant: a new machine learning algorithm for multiclass classification in small datasets

IOP Publishing Mach. Learn.: Sci. Technol. 6 (2025) 025012 https://doi.org/10.1088/2632-2153/adc86e

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
28 October 2024

REVISED
4 March 2025

ACCEPTED FOR PUBLICATION
1 April 2025

PUBLISHED
11 April 2025

PAPER

Logic replicant: a new machine learning algorithm for multiclass classification in small datasets

Pedro Corral*, Roberto Centeno and Víctor Fresno

UNED Research Group in Natural Language Processing and Information Retrieval, Universidad Nacional de Educación a Distancia, c/Juan del Rosal 16, Madrid 28040, Spain

* Author to whom any correspondence should be addressed.

E-mail: pedro@obdin.com, rcenteno@lsi.uned.es and vfresno@lsi.uned.es

Keywords: logic replicant, explainable machine learning algorithm, graphical interpretation, new machine learning model, small datasets, improved predictions

Abstract

Multiclass classification with small datasets often presents a significant challenge for conventional machine learning (ML) algorithms, predicting with an accuracy affected by this context of data scarcity. To remedy this, this papers presents a novel ML model based on a differentiable deterministic finite-state machine (DFSM) that improves the prediction performance compared with state-of-the-art multiclass classifiers applied in this ambit of small data per class. The proposed model uses a logic-arithmetic function that replicates the inherent classification logic of the problem rather than finding patterns of feature similarity. Our algorithm, called logic replicant, allows to learn problems that other classification models cannot. As the logic replicant is a DFSM it can learn any combinational logic, but it goes beyond this point learning other types of problems such as handwritten-digit recognition, and the detection of mice with Down syndrome based on the presence of 77 proteins. Our ML algorithm is also easy to interpret using quantitative diagrams, in comparison to less interpretable algorithms such as artificial neural networks, random forest, and others. The results obtained with different data sets related to math, physics, biology and image recognition show that our design based on a logic-arithmetic function and being a DFSM improves the generalisation capacity (better prediction accuracy) of the logic replicant compared to other state-of-the-art ML approaches.

1. Introduction

The importance of problems described by small data sets might seem to go unnoticed in a big data world [Gro20, And15], but it is a research area with clear applicability and increasing interest [BD20, RRK21, XJLL23, WDGK23, TWR+22]. Data is not always available or cheap and the real world has abundant cases of small data sets that are not easily solved using traditional machine learning (ML) [CWC+22]. This includes legal texts [CWC+22], screening of autism spectrum disorder in children [WDGK23], diabetes affecting Native Americans [JGS21], rare diseases such as Alkaptonuria [SCF+20], pandemics [FLD+20], particle physics [KMNS18], bioinformatics [ZZC+21], and drug discovery [ATRPP17] just to mention a fistful of them. The complexity and cost of annotated data in some domains is high because it needs to be performed by specialised experts like biomedical experts [WDGK23], layers or accountants [KKZ22] among others. Kokol, Kokol and Zagoranski [KKZ22] raise the question 'What is the small data problem in ML and how it is solved?', concluding that in general, the performance of ML is proportional to the size of the data set used for learning, while Faraway and Augustin [FA18] emphasise that small data sets with higher quality are preferable to larger data sets with lower quality. Wang, Duentsch, Guo and Khan [WDGK23] considered that the domain knowledge is less important when the data are abundant but very important when the data are scarce and wish to have 'a new ML algorithm that needs minimal feedback from human experts'. This last reflection is one of the main motivations of this study's proposal.

Moving one step above, a summary of some of the typical ML algorithms used for classification includes artificial neural networks (ANNs) [Yeg09] and all of their variants [Abr05], *support vector machines* [Nob06], *decision trees* [SY15] and ensembles based on these such as *random forest* (RF) [Bre21]. The category of *linear classifiers* [SD98] includes the *perceptron* [Ros58], *logistic regression* [Men02], *naive Bayes* [R+01], and *Fisher's discriminant* [BG98], being the *quadratic classifier* [Tha16] a generalisation of this last concept. The summary includes the *boosting* [MBGS14] ensemble meta-algorithm for reducing bias and variability, and the *genetic programming* [LP13] category covering *gene expression programming* [ZFO17], *multi expression programming* [OD02] and *linear genetic programming* [BBB07]. Finally, there are also *kernel estimation* [GM79, Weg18] and *learning vector quantisation* [Koh92], the precursor of self-organising maps [Koh13] (SOMs), one of the ANNs that started the summary. Among the presented models, the most commonly used models for dealing with small data sets [KKZ22] are decision trees [ZSC+16] and forest [SLD+19], convolutional networks [LD15, YNDT18, GNG+20], transfer learning [HPGG20, TEHD20], and support vector machines [ALS+16, CFQ+17, RZIX20]. Despite that some researchers like Weng, Song, Zhu, Yan, Sun, Grice, Yan and Yin [WSZ+20] have reached good performance in specific applications; in general these models are not specifically designed for small datasets. Therefore, depending on the problem it might be necessary to choose one or another, and complement them with additional techniques [XJLL23] to obtain the best results. This leads to an additional effort in model selection and the complementary technique, that might or might not provide the desired performance.

Some of the solutions applied for classification with small data sets are based on few-shot [WYKN20] and one-shot learning [VBL+16], where prior knowledge from a previous dataset is used to enhance the supervised learning on a small data set. Other techniques [LST15] use domain-specific prior knowledge to increase the original small dataset or guide the learning. Although that some of these solutions have proven to be effective, they are not always applicable in all scenarios, as it is not always possible to find prior knowledge and compatible data sets. Xu, Ji, Li and Lu [XJLL23] synthesise the overall problem as 'the essence of working with small data is to consume fewer resources to get more information', which leads us again to the ideal case of finding a ML algorithm that simply requires lesser data for providing reasonable performance.

Knowing the necessity, challenges and actual solutions around the small data, we propose a new ad hoc ML algorithm for multiclass classification that predicts with higher accuracy than some of the current ML alternatives. We call this model the *logic replicant*, and the following sections prove that it can even learn some mathematical [FC01] and physics [Wie18, WGU+10] problems that (to the best of our knowledge) are nonviable for current ML algorithms. The logic replicant is designed from scratch by applying four functional principles to address such small-data problems. We present these principles as hypotheses, and the present paper provides the evidence to support them:

- The logic replicant is **differentiable** so it can take advantage of modern and effective general optimisation methods, specifically the Adam [KB14] variation of the stochastic gradient descent.
- It can **replicate any deterministic finite-state machine (DFSM)**. In a simplified explanation, a DFSM [LY96] can model a given logic that maps different categorical input variables (input symbols) to a deterministic output class (output symbol) with the help of several states. Replicating any DFSM allows the replication any deterministic logic that involves categorical input and output variables.
- The prediction capability of our ML model is based on **replicating specifically the intrinsic logic that defines a problem**, that is, the logic that decides what inputs are mapped to which outputs beyond the similarities of the features. Some other ML algorithms may have this capacity [DAG98] to some extent, but the logic replicant is specifically designed to find and emulate such logic. For this specific purpose it contains a parametric function L that replicates the problem's logic while being differentiable, so it is fitted during the training [Rud16a]. It is assumed that the logic replicant will be used for learning problems that can be found in the real world; therefore, in one way or another, with more or less complexity, the logic of these problems can be defined [Teg08] using a mathematical expression [Sha83]. The goal of the function L is to be a good-enough approximation of the real function that describes the logic of a given problem. To achieve this goal, it is used an strategy based on low entropy isolations (LEIs) that is explained in the following sections.
- The fourth design principle is **explainability**, a characteristic opposed to the difficulty of interpreting other ML models [Rud19, ATS20]. This is important for some specific scenarios such as clinical diagnosis [ATS20], in which the logic that explains the prediction is as important as the prediction itself. For this purpose the logic replicant can generate 2-dimensional plots that allow easy interpretation of the classification logic and how different instances are closer or far under this logic. It is possible to find instances of the same class with similar features considerably far in the 2-dimensional space whilst others that have clearly different features can be found closer, depending on the learned logic. These colourful plots from the logic replicant might

resemble t-SNE [VdMH08] or SOMs [OKK03], but are rather connected to the replicant's learned logic, allowing us to quickly identify how good or bad the learning was and how precise the learned logic is.

In summary, our goal for the logic replicant is to cover two main objectives: (1) offering better performance in small data sets than a selection of ML algorithms, and (2) providing graphical explainability of the model in a given problem. The models selected for the comparison include the SOM, multilayer perceptron (MLP), RF and eXtreme Gradient Boosting (XGBoost) (sections 2.1–2.4) and the following sections will show evidence of the logic replicant satisfying the aforementioned requirements. This includes a validation of the logic replicant in datasets such as the parity function and nucleosynthesis (sections 4.1 and 4.3) which the selected ML algorithms simply cannot learn. In addition, a comparison of the logic replicant and the selected ML algorithms was evaluated in two additional small datasets, with the logic replicant being the algorithm with the highest accuracy.

Section 2 covers the state of the art of the selected multiclass classification ML algorithms. Section 3 describes a strategy based on low-entropy isolation used to evaluate how the logic replicant learns a specific problem and its underlying logic for generalizing future instances. Section 3 formally defines the logic replicant and demonstrates that it has the capacity of a DFSM. Section 4 explains some experiments performed to validate the design requirements of the logic replicant, and section 5 discusses the results. Finally, section 6 gathers our conclusions and intended future work.

2. State of the art

This section presents a selection of some ML algorithms used for multiclass classification in the context of small data sets. As it is far beyond the scope of this study to cover every existing multiclass classification ML algorithm, the proposed selection is based on the work of previous researchers, the expected predictive accuracy of such algorithms, and some similarities with the logic replicant.

Considering the good results obtained by Zhang *et al* [ZSC+16] and Shaikhina *et al* [SLD+19] using decision trees, RF, and Gaussian decision trees in some small data sets, it is interesting to have a small representation of some of these ML algorithms to compare them with the logic replicant. Among these, the RF was selected, as some authors [FT18] concluded that it almost always has better performance than single decision trees. The XGBoost is also added to this selection because of its good accuracy in many different problems [bib16, Nie16] when compared to single decision trees and other models, including the RF. As these two algorithms use to perform better than other tree-based models on a great variety of data sets, they can be assumed to be a reasonably good baseline for comparing the logic replicant without having to consider any other tree-based alternatives.

It is complicated to talk about ML without mentioning the ANNs and, in the context of small data sets, the conclusions of some researchers such as Pasini [Pas15], and Ingrassia and Morlini [IM05] may suggest that the MLP is an excellent candidate for our selection of ML multiclass classification algorithms. Considering that the scarcity of data might facilitate overfitting during training, the MLP with one hidden layer is a well-balanced configuration for achieving the best performance prediction with small data. It will be shown in the results (section 5) that this choice is justified because of the good performance of these models. In fact, after the logic replicant, MLP outperformed the rest of the proposed models.

The SOM is a special case in our ML selection because, despite being an ANN, it is not expected to provide exceptional predictive accuracy. It was selected because it shares two common points with our proposal to illustrate the effects of the logic replicant's design principles. The SOM is a non-linear generalisation of principal components analysis (PCA) [Yin08], which is a technique for reducing the dimensionality of a data set. This last concept is also used by the logic replicant when emulating logic of the problem into a reduced-dimension space. It can also provide graphical and interpretable representations of the learned dataset in a reduced dimension. Although the SOM has these two points in common with the logic replicant, it is interesting to see how their intrinsic differences have a significant effect on their predictive accuracy.

2.1. SOM

The SOM [Koh13] is an unsupervised ML technique used for automatic data-analysis, allowing the production of representations of high-dimensional data sets in lower dimensions. During the dimensionality reduction, the topological structure of the original dataset is respected, and it is not unusual to use a bidimensional representation that can be shown graphically; this is useful for visualising patterns. Dimensionality reduction is performed by distributing a training data set through a finite collection of components that are regularly distributed in the reduced space (map space). These components are not only called nodes, but also neurons, as the SOM is an artificial neuron network. These neurons are determined by

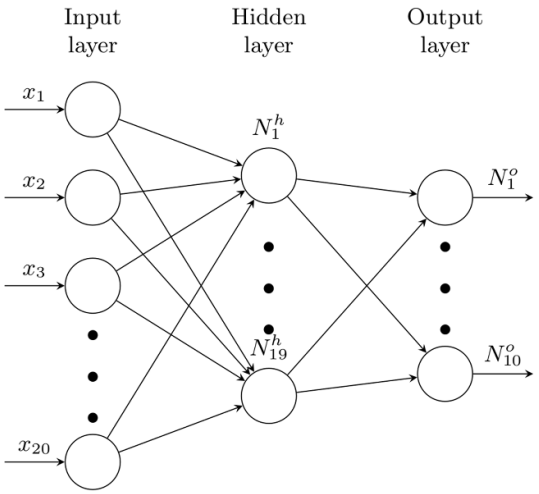

Figure 1. Generic representation of a MLP with one hidden layer, 20 input features captured in the input layer, 20 hidden neurons, and 10 different output classes. This feedforward network has all of its neurons from one layer fully connected to every of the neurons of the next one.

M weight vectors $\boldsymbol{w}_j = (w_{j1}, \ldots, w_{jF})^{\mathrm{T}} \in \mathbb{R}^F, j \in [1, M]$ of the same dimension F of the input values $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{iF})^{\mathrm{T}} \in \mathbb{R}^F$ in their original high-dimensional space, and these weights are initiated randomly or using other techniques [dBCV99] that can accelerate [OKK03] the learning. During the (competitive) training [dBCV99] of the SOM, the input items $\boldsymbol{x}_i$ are compared to all the nodes looking for the most similar weights $\boldsymbol{w}_j$ to find the best matching unit (BMU). This similarity comparison is performed using an Euclidean distance, so $|\boldsymbol{x}_i - \boldsymbol{w}_{\mathrm{BMU}}| \leqslant |\boldsymbol{x}_i - \boldsymbol{w}_j|, \forall j \in [1, M]$.

The final organisation of the weights through the map forms a topographic relationship of the data that can be used to obtain insights in a way that is often considered a non-linear generalization of PCA [Yin08]. When the dimension is reduced to a 2D-space, then the SOM allows the interpretation of the map obtained after the training, which is a similar way of providing explainability as the logic replicant despite both ML algorithms are very different internally.

In the specific scope of small datasets it is generally concluded that the SOM does not have an sufficient predictive accuracy. Hafiz [Haf22] points out how the data scarcity generates over-fitting and over-parameterization in some ANNs including the SOM, whilst researchers such as Lu and Segall [LS13] name this problem 'the bias caused by the insufficient training data'. The common solution to this problem is to create ensembles of SOM to improve the overall accuracy. As this approach complicates the interpretability of the ensemble compared with a single SOM, this alternative forces the choice between better accuracy (SOM ensemble) and graphical interpretability (stand-alone SOM). As it is shown in the following sections, our proposal covers both aspects at once.

2.2. MLP

The variety of ANNs is extensive and this subsection focuses only on the type of feedforward neural network (FNN) that will be used in the following sections during the experiments. FNNs differ from the recurrent neural network by having a uni-directional flow of information going from the input layer through the hidden layers to finally reach the output layer (figure 1). FNNs can be formed using several different approaches such as convolutional neural networks [MAM21], radial-basis function networks or transformers [NJL21]. The concrete case of the MLP has fully-connected neurons whose output is modified by an activation function that exhibits non-linear behaviour. This means that the MLP has the ability to classify data that are non-linearly separable when it has at least three layers. Each layer contains a concrete number of artificial neurons that are defined by a vector of real-valued weights $\boldsymbol{w}$ and compute their output as $f(\boldsymbol{w} \cdot \boldsymbol{x})$, where $\boldsymbol{x}$ is the output from the previous layer (in the case of the first hidden layer $\boldsymbol{x}$ corresponds to the original input instance itself), $\boldsymbol{w} \cdot \boldsymbol{x}$ is the dot product of $\boldsymbol{w}$ and $\boldsymbol{x}$, and f is the activation function that introduces the non-linear behaviour of the neuron. As the MLP is fully connected, every output $f(\boldsymbol{w} \cdot \boldsymbol{x})$ is connected to every neuron of the next layer, and these connections are defined with more weights in the next layer. Its original activation function was the Heaviside step function, but now is common to use activation functions such as sigmoid functions. Two samples of these sigmoid functions are the hyperbolic tangent $f(x) = \tanh(x)$ and the logistic function $f(x) = 1/(1 + \mathrm{e}^{-x})$. Another common activation function is the rectified linear unit (ReLU) [Bro19], which is used to overcome some numerical limitations found in sigmoids, and can be defined as $f(x) = \max(0, x)$.

The training of the MLP is supervised, mainly based on backpropagation [CR13] or some variants of the gradient descent algorithm [DLL+19], such as Adam [KB14]. This is an efficient training method that can be used with the logic replicants.

The general problem of FFNs with small data sets appears when they have a considerable proportion of parameters compared to the size of the data, tending to produce over-fitting and over-parameterizing [Haf22, LS13, JGS21, LD15, YNDT18, GNG+20]. In this general behaviour [HNP09], the case of the MLP is not an exception [BSY22, Aur17] because, unlike the logic replicant, the MLP or other FNNs are not designed from scratch with the specific purpose of dealing with small data sets. The solutions for this issue go from using batch normalization as a way of strong regularization [LD15, OWB18], data augmentation [ZZK+20, SK19], and transfer learning [YNDT18, HPGG20, TEHD20]. Unfortunately, these solutions are not implemented by default in the FNNs, and in the case of data augmentation and transfer learning, the solution is dependant on the type of data set to learn, so it is not always applicable, nor is it a general solution. Fortunately, some researchers [Pas15, IM05] have evinced that configuring the MLP with a smaller number of hidden neurons (or 'degrees of freedom' [IM05]) provides a better balance between the number of parameters and the data size, providing good accuracy when predicting. These conclusions suggest that an MLP configuration with few parameters may be a good candidate for comparing the performance in small data sets, and the results shown in section 5 support this assumption. These same results indicate that there is room for improvement in their performance, which is one of the reasons for developing the logic replicant. In addition, it is important to mention that the MLP is not as easily interpretable as the SOM, another of our initial requirements.

2.3. RF

The RF is an ensemble learning algorithm for classification and regression model created by Tin Kam Hot [Ho95] in 1995, which was improved and trademarked by Leo Breiman [Bre21]. RF combines several decision trees (generated during training) to predict its output. In the case of multiclass classification problems, it chooses the most selected class among all trees. The reason for this is to avoid the undesired effects of overfitting of individual trees, balancing their variance. Leo Breiman [Bre21] incorporated the concept of 'bagging' (bootstrap aggregating) [Bre96] that is derived from the concept of bootstrapping, and consists in sampling uniformly and with replacement a data set Ω into new m data sets $\Omega_i : \forall i \in \mathbb{N}+$, $i \in [1, m]$. This replacement during sampling allows forseveral Ω_i and $\Omega_j (i \neq j)$ to share some elements, but their content does not depend on those from the others. This bagging is combined in the RF with the original proposal from Tin Kam Hot [Ho95] using a random selection of features for every independent tree. In this manner, every decision tree works with a sampling of instances and features from the original instances that form Ω. This also helps to avoid obtaining the same structure on different trees, but also in reducing the total computation time, compared to using all the features and all the samples from the original data set when training every tree that forms the ensemble.

It is important to mention that our interest in the RF is in using it as a reference for evaluating the performance of our proposal, as there are a great number of studies proving that this ML algorithm has good classification accuracy [MJZ18]. There is general agreement that its good performance is related to its ability to avoid overfitting, but also to its tolerance to noise and outliers [MJZ18]. Within the scope of small datasets, successful cases of good predictive accuracy have been reported using RFs. Shaikhina *et al* [SLD+19] were able to obtain 'early prediction of acute antibody-mediated rejection' happening in kidney transplantation, based on the case of just 80 patients. Fernández-Delgado *et al* [FDCBA14] found that the RF (R implementation) performed well in 121 datasets available in the UCI Repository [DT17], with the majority being easily classified as small (less than 1000 instances) and very often also with less than 400 instances. For this reason, Olson *et al* [OWB18] used RFs as their baseline for predictive performance in small datasets. Other authors, such as Han *et al* [HWF21] appreciate such accuracy, but consider that additional methods (two-phase sampling) are needed to improve its accuracy in biomedical studies. In addition to this need, we also consider its difficulty to be interpreted through the hundreds of components of its ensemble; these two factors are the goals of the logic replicant.

2.4. XGBoost

XGBoost is an algorithm and open-source framework for applying gradient boosting. Similar to RF, XGBoost also uses an ensemble of weak predictors to orchestrate the final prediction for both multiclass classification and regression. In the context of this paper, it is an interesting algorithm because it has gained popularity as the winning option of several competitions focused on the performance of ML in different problems [bib16, Nie16]. XGBoost gathers some improvements that contribute to a good accuracy in multiclass classification, such as the Newton boosting algorithm [Nie16], cache-aware and sparsity-aware learning [CG16], shrinkage and column subsampling [CG16]. The shrinkage was initially proposed by Friedman [Fri02] and allows for

the improvement of the overall model by reducing the influence of each individual tree (weak predictor). In addition, column (feature) subsampling is used in the RF [Bre21, FP+03] algorithm. Beyond these design decisions, XGBoost also implements practical improvements related to computation speed and scalability when working with larger datasets. One of these practical improvements is the performance of the best split at the tree nodes. The usual *exact greedy algorithm* [CG16] evaluates all possible splits found in every feature, and in continuous features which requires considerable computational effort, especially when the data do not fit in the computer's memory. To solve this problem, XGBoost uses an approximate algorithm [CG16] that allows computation with lesser memory.

It goes beyond the scope of this paper to cover all the structural and implementation improvements, rather than summarising that the combination of these allows XGBoost to achieve good accuracy in many different problems [bib16, Nie16], which makes it a good baseline for comparing our proposal. In addition, some research has highlighted the good performance of XGBoost with smalldata sets. Zou *et al* [ZJQ+22] successfully applied it for predicting the relative density of parts made of alpha-beta titanium alloy Ti-6Al-4V and fabricated using Selective Laser Melting; these are used in chemical, aviation and medical fields. In other researches, the same researchers found results where the XGBoost outperformed other models, including some ANNs, and Ogunleye and Wang [OW19] compared several ML algorithms with small data about Chronic Kidney Disease diagnosis, and concluded that XGBoost was the best performing one. Liang *et al* [LLZW20] analysed and confirmed the good performance of XGBoost for predicting pillar stability in hard-rock mines. In parallel with the conclusions from the RF, we also consider XGBoost as good candidate for prediction in small datasets, but with the lack of easy interpretability. However, our results show how the logic replicant can significantly improve its performance.

2.5. Conclusions

The principal conclusion obtained after reviewing the existing proposals for dealing with small amounts of data is the lack of a unified methodology or ML algorithm that can be systematically applied to different problems. Considering that the mentioned main driver is the predictive accuracy, which forces researchers to try different models, even having frequently to enhance them with specific heuristics. For example, the SOM incorporates additional steps to obtain a single solution with both good accuracy and interpretability. Having a general solution that covers these aspects of good accuracy without having to resort to additional data (transfer learning) and explainability would significantly simplify the work of researchers who face this challenge, especially saving their time and investing it in other tasks that would have even higher value, for example, applying these results in real scenarios.

Considering this context, we propose a new ML algorithm, the logic replicant, that covers these two aspects: (1) offering a generally consistent good accuracy in small data sets in particular, and (2) providing a graphical explanation of the learned problem. This solution would be a starting point for a researcher who faces a problem with small data, providing a good solution for their problem, or in the worst scenario being a baseline for comparing with their ad hoc solutions developed for their specific problem.

The logic replicant is founded in some hypotheses applied to the problems defined with small data sets: (1) it is possible to find a simple logic that correlates the instance's features with its right class. (2) This logic replicates the real mechanism, cause, or reason that determines how features are evaluated for identifying the corresponding class. (3) This logic can be represented using a mathematical function that maps the feature space X to a continuous space Q where classification is easier. The following section explains how these hypotheses are implemented in the concrete shape of a logic replicant.

3. The logic replicant

The logic replicant is a supervised multiclass ML algorithm that learns the underlying logic of a problem rather than recognising similarities in its features. Every non-random classification problem has a reason that connects an instance to its class, independent of whether this criterion is known. These reasons can be extremely diverse and based on combinations of empirical processes such as the laws of physics, biology, social behaviour, and many of others, including those based on human interpretation, opinions, or abstraction from mathematics such as modern algebra. A logic that recognises similarities are the ones that can be found in models such as the RF, where every of its nodes evaluates a rule applied on a unique feature, forming a conjunction of several one-feature-based rules. Another sample can be found in the *k-nearest neighbors algorithm* [FH89] which directly compares the similarity (Euclidean distance) of the features to leared samples. This differs from *semantic descriptions* [Rud11], where the definition of an element (the intrinsic logic) can be, for example, the rational numbers expressed as a quotient of p/q or just the colours of the French flag. A well-defined logic, or compact logic is that it allows the identification of elements from a larger set by a relation that involves more than a feature at a time, for example, the individuals who spend

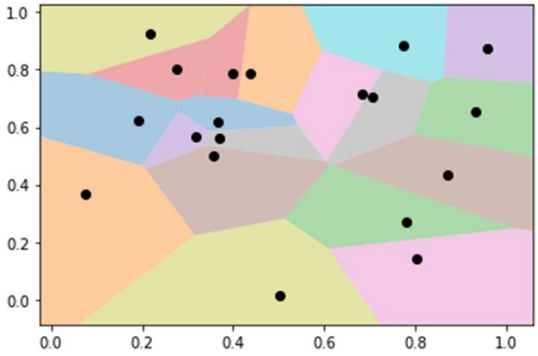

Figure 2. Voronoi diagram/tessellation formed by 15 points and their Voronoi cells.

more calories than they consume, or the phases of matter of a single constituent specified by the Clausius–Clapeyron equation [Kou12] simultaneously using the boiling temperature, vapor pressure, boiling temperature, heat of vaporization, and the ideal gas constant. Because the logic replicant is also a DFSM it can learn any combinational logic [ZRK20]; however, as shown in the following sections, it can also be applied to other types of problems. The classification logic is replicated using arithmetic operations that project any input instance $\boldsymbol{x}$ from space X of features to a transformed space Q of dimension $D = \dim(Q)$. This logic is completely captured by a function $L(\boldsymbol{x})$ (14) which works to form groups of instances $\boldsymbol{q} = L(\boldsymbol{x})$ in Q with as low local entropy as possible. Because this logic can be complex and not evident at all, it is perfectly possible to find cases where instances associated with the same class c but with significant different features at X might be considered extremely similar in this space Q and instances really similar at X (even if they share the same class c) might be evaluated as clearly different when projected by $L(\boldsymbol{x})$ to space Q. This is translated with the help of the Euclidean norm [CCK11], as both scenarios $\|L(\boldsymbol{x}_1) - L(\boldsymbol{x}_2)\| \ll \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|$ and $\|L(\boldsymbol{x}_1) - L(\boldsymbol{x}_2)\| \gg \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|$ might be perfectly possible depending on $L(\boldsymbol{x})$ and the problem to be learnt.

3.1. Intuitive introduction

The logic replicant uses a transformation function (14) $L\colon X \subset \mathbb{R}^F \mapsto Q \subset \mathbb{R}^D, (F, D) \in (\mathbb{N}^+, \mathbb{N}^+)$ evaluating F-dimensional instances $\boldsymbol{x}$ defined as a vectors of real numbers, mapping them as points $\boldsymbol{q}$ at the D-dimensional transformed space Q. We can choose the dimension D for our transformed space Q with any value that is convenient in terms of the model's performance, but it is interesting to mention that the case where $D = 2$ can be used to visually illustrate how the model interprets the transformation, but also for having a graphical explanation of the model's outcome after the learning process. The purpose of the transformation function L is to perform *quantification using ordered numbers* of the input instances $\boldsymbol{x}$ as points $\boldsymbol{q}$ in the space Q. This detail is important because the areas where the quons $\boldsymbol{q}$ are placed belong to a specific class, and they should be as isolated as possible from quons of other classes. This is easy to imagine as grouping the quons in space Q and can be forced by introducing (figurative) vortices that geometrically attract the quons to the different classes $c \in C$. This attraction is performed during the training using a fitness function. Every vortex tries to cover as much quons from its class as possible, defining the limits of the areas of every class c at the transformed space Q. A visual way of seeing this for the case of 2-dimensional spaces, is comparing the vortices and the areas these describe as a Voronoi tessellation in a Voronoi diagram (figure 2). This is not a equivalent analogy, but pretty intuitive; the vortices expand from their centre, trying to cover the greatest possible area before finding the limit with another vortex. The analogy is not complete because the vortices have properties that allow them to tessellate using functions a bit more sophisticated than the Euclidean distance, and also because several vortices belonging to the same class can be combined for enhancing their respective class.

To understand the next sections, it is important to keep in mind that the logic replicant uses a discriminant function $L(\boldsymbol{x})$ (14) that is able to differentiate the several classes of a problem by learning and replicate the inherent logic of a given problem. This means that the inherent logic of the problem is replicated using algebraic operations $\boldsymbol{q} = L(\boldsymbol{x})$, where its output $\boldsymbol{q}$ can be interpreted directly in Q. The function $L(\boldsymbol{x})$ (14) maps any input instance $\boldsymbol{x}$ to a transformed space Q where grouping is easy. This transformation is based on the problem's inherent classification logic but does not necessarily consider similarities between different instances at X ($\boldsymbol{x}$ and $\boldsymbol{x}'$ being close to the feature space X). This means that some instances of the same class can be found closer in Q than in the original feature space X, but this is not a requirement for all instances. Only some instances (similar or not at X) form groups in Q that are identified

with specific classes. The identification of these groups of quons $\boldsymbol{q}$ in Q to their final class c is performed using a mathematical object that can be called a (figurative) vortex. These vortices cover all areas of space Q and are associated with a unique class c in a many-to-one relationship. Every class c has one or several associated vortices. Every one of these vortices has a scalar field defined by the function $U(\boldsymbol{q})$ (15) all over the space Q, and a given point $\boldsymbol{q} = L(\boldsymbol{x})$, the vortex with the greatest scalar field classifies $\boldsymbol{q}$ to its corresponding class c. Keeping that in mind, the following section 3.2 introduces the rational supporting the usage of the function $L(\boldsymbol{x})$ and what requirements are a good choice for a proper selection of $L(\boldsymbol{x})$.

Let us finish this subsection by explaining at a high level what is required for having a proper function $L(\boldsymbol{x})$. Every useful $L(\boldsymbol{x})$ ought to apply the problem logic in such a way that the projection to space Q concentrates the quons q that have logical similarities rather than similar features. For example, in a hypothetical classification of musical instruments such as guitar or bass guitar, the number of strings might be used for grouping 6-string guitars in a different region in Q than the 4-string bass guitars. In such cases, the less typical 12-string guitars might form their own group differentiated from the 6-string guitars despite both being classified as guitars. In the case of 6-string bass guitars, this group might be closer (or not) to the 6-string guitar despite they belong to different classes. If the classification logic is found to be the size of the instrument instead of the number of strings, then $L(\boldsymbol{x})$ might simply group all the guitars into one unique group of similar body size and all the bass guitars in another with a greater body size. As can be seen in this illustrative case, the logic for classification depends on the commonalities in the instances features, and can vary depending on the training as well as other factors such as the number of vortices. Having just one vortex per class constrains the logic to use just one group per class, but having two or more vortices per class allows (but does not force) the logic to have different groups for 6-string or 12-string guitars. Independent of what logic is found, the projections of the instances to space Q ought to generate groups (one or many) of the same class where the average distance between them is closer to the original distance in feature space X. On the other hand, the distance between groups of instances of different classes ought to be the same or greater in space Q than in original space X. These groups do not need to be extremely compact or separate from the others, just enough for the vortex to differentiate them properly. When the logic is compact (as defined earlier), there is no need for many samples to determine the classification criterion. On the other hand, as space Q is continuous and shared by all the groups of the different classes, gathering one group from one class might also help to define other groups from the same or other classes. For example, in our sample of the number of strings for guitars and bass guitars, the formation of the group of 6-strings almost inevitably helps to form a group of 4-strings.

3.2. The strategy of LEIs

When selecting a proper transformation function L for a data set Ω, it is obvious that not every function is suitable for this task, so it is necessary to evaluate how good or bad it performs while projecting instances $\boldsymbol{x}$ from the space of features X to space Q. Not only this is required, but also a strategy for finding the optimal transformation function L. For doing this it can be used the spatial distribution of the entropy [Lin91, WZS+13] of the dataset Ω on the transformed space Q. The strategy is to find an L that isolates the transformed instances $\boldsymbol{q} = L(\boldsymbol{x})$ of the same class c to areas where some of them (not all of them) are separated from the isolations of other classes $c' \in C\colon\ c' \neq c$. Every isolation is associated with a given class c and could be seen as artificial clusters at the space Q, arranged to maintain a proper distance from the others. These isolations are areas where their boundaries and content are covered and delimited by figurative vortices identified with a given class c. With this spatial isolations where the probability p_c of belonging to a given class c is high or even 1, classifying new instances $\boldsymbol{x}$ only requires calculating $\boldsymbol{q} = L(\boldsymbol{x})$ and selecting the corresponding isolation at the space Q (with the help of a vortex), and assigning its class c to the instance. The characteristic required for each of these isolations is to have a low entropy locally; the lower the entropy, the easier and more likely it is to be classified into a concrete class. In terms of the variability of classes, a LEI for a given class will show a homogeneous distribution of instances in space Q, whereas areas with higher entropy are heterogeneous distributions of instances of different classes. The higher the homogeneity in an isolation, the lower entropy it will have, becoming a LEI. On the other hand, if there is high heterogeneity of classes within an isolation, it will not be considered a LEI. A perfect mapping between the original space of features and a practical transformed space Q guarantees that are clear LEIs are found. With a transformation function L able to produce proper LEIs, the vortices are placed to delimite these LEIs and identify them (and any instance $\boldsymbol{q}$ falling inside) as belonging to concrete class c. This requirement of low entropy for the isolations generated by L, guarantees that the vortices that will cover the LEIs will also have low entropy and a high probability of being classified correctly.

3.3. Formal definition

Once the involved elements and their associated concepts are presented, let us define them more precisely. The classification strategy behind the logic replicant is to transform a given data set Ω from its original space of features $X \subset \mathbb{R}^F$: $F \in \mathbb{N}^+$ to a transformed space $Q \subset \mathbb{R}^D$: $D \in \mathbb{N}^+$ where it is easier to classify. To do this, we use the entropy of the dataset Ω associated with the given problem. In a multiclass problem with a dataset Ω, with $|C|$ possible output classes with a probability distribution $p_c : \mathbb{R}^F \mapsto [0,1], F \in \mathbb{N}^+$ for every instance $\boldsymbol{x} \in \Omega$, the Shannon entropy [Lin91, WZS+13] is given by

$$H(\Omega) = -\sum_{c \in C} p_c \log_2 (p_c) \tag{1}$$

where $H(\Omega)$ is the entropy for the data set Ω and p_c is the fraction of instances classified as c. The entropy (1) provides information about the problem as a whole, but not about the space Q that can be used to facilitate the classification. The aim is to find a convenient transformed space Q with isolated areas where the entropy is as low as possible because in this low entropy isolations (LEIs), the classification should be easier. Even after such a transformation, when considering the whole entropy of all the LEIs together (whole Ω), its value is still equal to $H(\Omega)$. To generate LEIs, it is required to map many instances from a class c to an area of Q containing the fewest instances associated with other classes $c' \neq c$. Inside an area of Q with a high frequency of appearance of class c, we have

$$\lim_{p_c \to 1} p_c \log_2 (p_c) = 0 \tag{2}$$

$$\lim_{p_{c'} \to 0} p_{c'} \log_2 (p_{c'}) = 0, \forall c' \in C\colon c' \neq c \tag{3}$$

which guarantees that it has a low entropy. To asses whether an area of Q is a LEI, it is employed a scalar field based on the interaction of all instances of Ω. When a pair of instances $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ is transformed into Q as $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$, we can define a function of the relationship for the pair with the expression

$$\zeta (\boldsymbol{q}_a, \boldsymbol{q}_b) = \frac{1}{1 + \rho^2 (\boldsymbol{q}_a - \boldsymbol{q}_b)^2}. \tag{4}$$

The function (4) is inspired by the potential energies from classical mechanics but avoids infinite values and discontinuities when $\boldsymbol{q}_a = \boldsymbol{q}_b$, with $\zeta(\boldsymbol{q}, \boldsymbol{q}) = 1$. The last case of $\boldsymbol{q}_a$ being as close to $\boldsymbol{q}_b$ (if they belong to the same class) could even be considered a goal for making the classification process easier. This is a problem when using classical potential expression for two particles, which takes the form

$$U(\boldsymbol{q}_a, \boldsymbol{q}_b) = \frac{k}{|\boldsymbol{q}_a - \boldsymbol{q}_b|} \tag{5}$$

where k depends on the mass or charge of the pair of particles. The potential (5) takes infinite values at $\boldsymbol{q}_a = \boldsymbol{q}_b$, but that can be avoided using the term $\epsilon + |\boldsymbol{q}_a - \boldsymbol{q}_b|$ as non-zero denominator, so $\epsilon > 0$. Even so, the potential of the form $\frac{k}{\epsilon + |\boldsymbol{q}_a - \boldsymbol{q}_b|}$ would be continuous at $\boldsymbol{q}_a = \boldsymbol{q}_b$ but not its derivative. This can be avoided using $\epsilon + (\boldsymbol{q}_a - \boldsymbol{q}_b)^2$ instead of $\epsilon + |\boldsymbol{q}_a - \boldsymbol{q}_b|$, having an expression like

$$\frac{k}{\epsilon + (\boldsymbol{q}_a - \boldsymbol{q}_b)^2} \tag{6}$$

which is equivalent to (4) when introducing the convention $\zeta(\boldsymbol{q}, \boldsymbol{q}) = 1$. That leads us to conclude $\epsilon/k = 1$ and $\rho^2 = 1/k$. The reason for selecting ρ^2 was to emphasize that $\rho^2 > 0$. Using the function ζ (4), it is easy to evaluate for a given $\boldsymbol{q}_c$ belonging to a class c, its interactions with the rest of instances at the space Q. These interactions can be divided into two groups: interactions with other instances of the same class c, and interactions with instances of other classes c'. These are

$$W_{\text{hom}} (\boldsymbol{q}_c) = \sum_{\substack{\boldsymbol{q}_i \neq \boldsymbol{q}_c \\ \boldsymbol{q}_i \mapsto c}} \zeta (\boldsymbol{q}_c, \boldsymbol{q}_i) = \sum_{\substack{\boldsymbol{q}_i \neq \boldsymbol{q}_c \\ \boldsymbol{q}_i \mapsto c}} \frac{1}{1 + \rho^2 (\boldsymbol{q}_c - \boldsymbol{q}_i)^2} \tag{7}$$

$$W_{\text{het}} (\boldsymbol{q}_c) = \sum_{\substack{\boldsymbol{q}_i \neq \boldsymbol{q}_c \\ \boldsymbol{q}_i \mapsto c' \neq c}} \zeta (\boldsymbol{q}_c, \boldsymbol{q}_i) = \sum_{\substack{\boldsymbol{q}_i \neq \boldsymbol{q}_c \\ \boldsymbol{q}_i \mapsto c' \neq c}} \frac{1}{1 + \rho^2 (\boldsymbol{q}_c - \boldsymbol{q}_i)^2} \tag{8}$$

where W_{hom} is the homogeneous scalar field and W_{het} is the heterogeneous scalar field. W_{hom} measures the closeness of $\boldsymbol{q}_c$ from the other instances of the same class c; therefore the higher it is, the easier it should be to

isolate these instances in a closed area with low entropy and a high probability of classifying as class c. An instance $\boldsymbol{q}_c \in Q$ belonging to class c, which is surrounded by other n instances $\boldsymbol{q}_i$ belonging to the same class, will experience

$$\lim_{\substack{\boldsymbol{q}_i \to \boldsymbol{q}_c \\ \forall \boldsymbol{q}_c \in \Omega:\ \boldsymbol{q}_c \mapsto c}} W_{\text{hom}}(\boldsymbol{q}_c) = n = |\Omega| p_c \tag{9}$$

where $|\Omega|$ is the cardinality of Ω and p_c is the same fraction of instances classified as c from (1)–(3). This limit (9) is only achieved if all the $\boldsymbol{q} \mapsto c$ are placed at the same point in Q; that would be a perfect generalization for the class c. The other scalar function, W_{het} measures how close and mixed it can be found $\boldsymbol{q}_c$ to instances belonging to other classes $c' \neq c$, so the lower its value, the easier it is to reduce the entropy around $\boldsymbol{q}_c$. A space Q where the instances of different classes are far apart from each other will lead to

$$\lim_{\substack{|\boldsymbol{q}_c - \boldsymbol{q}_{c'}| \to \infty \\ \forall \boldsymbol{q}_{c'} \in \Omega:\ \boldsymbol{q}_{c'} \mapsto c',\, c' \neq c}} W_{\text{het}}(\boldsymbol{q}_c) = 0 \tag{10}$$

where $|\boldsymbol{q}_c - \boldsymbol{q}_{c'}| \to \infty$ denotes this large distance between the instance $\boldsymbol{q}_c$ and other instances $\boldsymbol{q}_{c'}$ belonging to different classes. Unfortunately, the values obtained by W_{hom} and W_{het} usually do not reach these extreme values, but others are in between. To combine these two scalar fields and evaluate the effectiveness of our LEIs, we define the uniformity of the distribution of all instances $\boldsymbol{q} \in \Omega$ as

$$B = \frac{\Gamma_{\text{het}} - \Gamma_{\text{hom}}}{\Gamma_{\text{het}} + \Gamma_{\text{hom}}} \tag{11}$$

$$\Gamma_{\text{hom}} = \sum_{\boldsymbol{q} \in \Omega} W_{\text{hom}}(\boldsymbol{q}) \tag{12}$$

$$\Gamma_{\text{het}} = \sum_{\boldsymbol{q} \in \Omega} W_{\text{het}}(\boldsymbol{q}). \tag{13}$$

The function B (11) is an LEI evaluation of the dataset Ω. It can also be interpreted as an estimation of the balance between the homogeneous and heterogeneous scalar fields over all instances $\boldsymbol{q}$ in transformed space Q. It is important to remark that B can be evaluated on any alternative spaces, even in the original space X of the features. For a completely heterogeneous space with all the instances mixed in close areas, then $B = 1$; alternatively, in a perfect isolation of classes, then $B = -1$. This can be deduced directly by considering (9) and (10) in (11).

Definition 1. A transformation function $L : X \subset \mathbb{R}^F \mapsto Q \subset \mathbb{R}^D, (F, D) \in (\mathbb{N}^+, \mathbb{N}^+)$ can be labeled as *leietanic* for a deterministic classification problem with a dataset Ω, when B_Q int the space of Q satisfies $B_Q < B_X$, compared with B_X in the original space of features X.

According to definition (1), a leietanic transformation function cannot be rigid; it explicitly excludes any linear transformation that preserves the Euclidean distance (Euclidean isometry), such as rotations, translations, reflections and combinations of these.

Definition 2. A logic replicant is a tuple $\langle L, V \rangle$ able to *replicate* the hidden classification logic of a problem; it receives $F \in \mathbb{N}^+$ inputs signals x_1 through x_F where:

- $L : X \to Q$ is a LEIETANIC TRANSFORMATION FUNCTION, that maps every input case $\boldsymbol{x}$ into a point $\boldsymbol{q}$ in the D-dimensional space Q, where

$$L(\boldsymbol{x}) = \boldsymbol{\Lambda}_a \boldsymbol{x} + \sum_{s=1}^{S} \sum_{r=1}^{R} \boldsymbol{\xi}_{sr} \circ |\boldsymbol{\Lambda}_s \boldsymbol{x} + \boldsymbol{\beta}_{sr}| \tag{14}$$

  where $\boldsymbol{\Lambda}_a$ and the other $\boldsymbol{\Lambda}_s : s \in [1, S]$ are $D \times F$ matrices that perform a multiplication of the vector instance $\boldsymbol{x}$. The vector collections $\boldsymbol{\beta}_{sr}$ and $\boldsymbol{\xi}_{sr}$ have the same dimension D. The operator '$\circ$' is the Hadamard product [Mil07], defined for matrices and vectors as 'entrywise multiplication'.
- V is the set of VORTICES and their respective scalar fields $U_{ck} \in V, U_{ck} : Q \mapsto \mathbb{R}, c \in C, k \in K \subset \mathbb{N}^+ \wedge C \cdot K = |V|$ that collectively decide what class to assign to the output of the LEIETANIC FUNCTION. Every c, kth-vortex (defined by its scalar field U_{ck}) is assigned to only one specific output class c, where

$$U_{ck}(\boldsymbol{q}) = \frac{\Phi_{ck}}{1 + \rho_{ck}^2 (\boldsymbol{q} - \boldsymbol{v}_{ck})^2} \tag{15}$$

is based on three parameters, its centre $\boldsymbol{v}_{ck} \in \mathbb{R}^D$, its density $\rho_{ck} \in \mathbb{R}$ and its intensity $\Phi_{ck} \in \mathbb{R}$. Every class c has assigned exactly K vortices, and the scalar field U_c of this cth-class is defined by the sum of the scalar fields of its K vortices (15) as

$$U_c(\boldsymbol{q}) = \sum_{k=1}^{K} U_{ck}(\boldsymbol{q}) = \sum_{k=1}^{K} \frac{\Phi_{ck}}{1 + \rho_{ck}^2 (\boldsymbol{q} - \boldsymbol{v}_{ck})^2}. \tag{16}$$

Every cth-class' scalar field is evaluated at point $U_c(L(\boldsymbol{x}))$, and then it is selected as final classification for $\boldsymbol{q} = L(\boldsymbol{x})$, the class ω with the greatest scalar field $U_\omega(\boldsymbol{q})$:

$$\omega = \underset{\omega}{\operatorname{argmax}}\, U_\omega(\boldsymbol{q}) = \{\omega \in C\colon\, U_\omega(L(\boldsymbol{x})) \geqslant U_c(L(\boldsymbol{x})), \forall c \in C\}. \tag{17}$$

The rationale for selecting a scalar field with (15) is based on the following geometrical explanation. The ckth-vortex's centre $\boldsymbol{v}_{ck}$ defines where it is placed within the space Q, while its density ρ_{ck} and intensity Φ_{ck} define how big or small these are, but also how much intensively a quon (transformed instance) belongs to a class. Connecting both components: The leietanic function transforms the input instances $\boldsymbol{x}$ into quons $\boldsymbol{q}$ (in space Q) that are identified with a class c if it has the greatest scalar field (16) $U_c(\boldsymbol{q}) = U_{\max}(\boldsymbol{q})$ at $\boldsymbol{q}$.

Lemma 1. *For configurations where $D = dim(Q) = 2$ and the total number of vortices is equal to the number of classes, $|V| = |C|$ so $K = 1$, the class selection using the greatest scalar field (17) is analogous to a 2-dimensional Voronoi tessellation where the distance metric corresponds to the inverse of the vortex's scalar field $U_c(\boldsymbol{q})^{-1}$ (15), where d defined as*

$$d(\boldsymbol{q}, \boldsymbol{v}_c) = \frac{1 + \rho_c^2 (\boldsymbol{q} - \boldsymbol{v}_c)^2}{\Phi_c}. \tag{18}$$

The lemma 1 allows the identification of every vortex's centre at the space Q with the location of a seed in the equivalent 2-dimensional Voronoi diagram. The area that covers every ith-vortex mapping the classification to a given class c, requires that its scalar field $U_c(\boldsymbol{q})$ (15) is the greatest (within this area) among all the others. As there is only one vortex per class ($|V| = |C| \wedge K = 1$), the scalar field of a class c is reduced from (16) to $U_c(\boldsymbol{q}) = \frac{\Phi_c}{1+\rho_c^2(\boldsymbol{q}-\boldsymbol{v}_c)^2}$. The greatest of the scalar fields means the smallest of its inverse $U_c(\boldsymbol{q})^{-1}$ (18), therefore, this inverse of the scalar field can be used as a distance metric for the equivalent Voronoi tessellation. In the equivalence vortex-Voronoi, the areas where the cth-vortex is the greatest are the Voronoi cells, where the distance is the smallest. The equivalence between the greatest scalar field $U_c(\boldsymbol{q})$ and the smallest distance $d = U_c(\boldsymbol{q})^{-1}$ applies to every possible point $\boldsymbol{q} \in Q$ of the vortices or 2-dimensional Voronoi diagram, where both representations are equivalent.

3.4. Connection with curve fitting and piecewise-linear interpolation

To have a deeper understanding of the logic replicant, it is interesting to know the connection of its leietanic function with curve fitting, and more concretely with piecewise-linear interpolation. Curve fitting [O'H78, GG12] is a process of defining a mathematical object (normally a function) that best fits a series of data points. Different approaches exist, such as interpolation, smoothing or regression analysis. The interpolation theory [Dav75] covers aspects such as the construction of families of interpolation spaces and the understanding of their characteristics and properties, as the *real interpolation spaces* and the *complex interpolation spaces* are two of the most applicable of these families [L+09]. The smoothing methods [GG12] are a bridge between a parametric approach with strong assumptions about the structure of the data and a non-parametric approach with no assumptions on the formal structure of the data points [Sim12]. Within the regression analysis [DS98], polynomial regression is important in our proposal as some of the functions that form the logic replicant are inspired by it. Polynomial regression models the relationship between the predictor variables and the predicted ones using polynomials, and it can be applied when there are reasons for suspect that the relationship between predictors and predicted variables is curvilinear [Ost12]. The polynomial regression in the case of a single variable takes the form

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \ldots + \beta_k x_i^k + \epsilon_i \tag{19}$$

where y_i and x_i are the ith-value of the predicted variable y and predictor variable x, k is the degree of the polynomial, and ϵ_i is the residual of the polynomial at the ith-approximation of y_i. This can be estimated using the mean square error (MSE), such as

$$\text{MSE} = \frac{\sum_i^n (y_i - \hat{y}_i)^2}{n - k - 1} \tag{20}$$

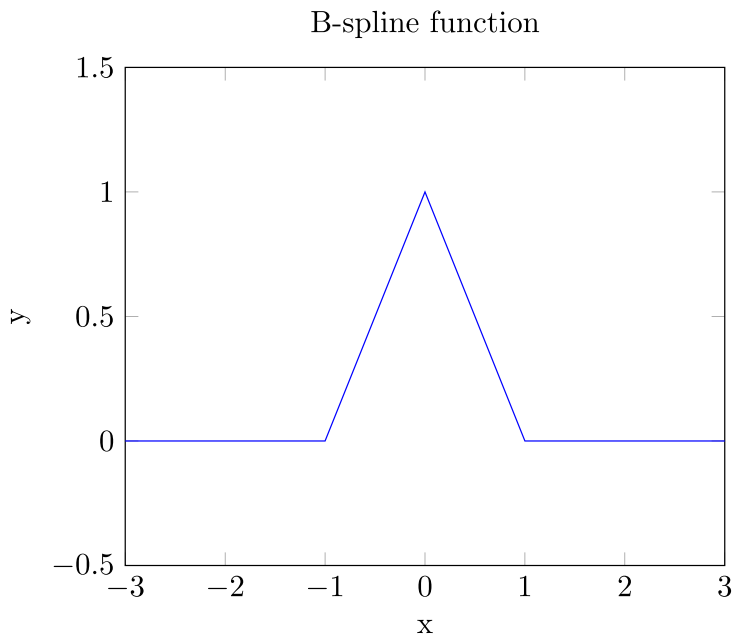

Figure 3. Plot of the 'hat' function $\Lambda(x)$ defined as (22) and used for building a piecewise-linear interpolation as (21).

that is an unbiased estimator of the variance σ^2 of the random error, being $\hat{y}_i$ the estimated value of y_i, so $y_i = \hat{y}_i + \epsilon_i$, and n is the total number of data points for the regression. The function (19) is related to non-convex functions in the sense that it can have $k-1$ inflection points that might be minima or maxima; these inflection points change the sign of the curvature. Although this is a very powerful characteristic when fitting curves, it has the practical drawback of dealing with very large numbers when $|x_i| > 1$ and the degree k of polynomial (19) becomes significant.

One solution to the problem of having many inflection points in the data to be fitted, is to use some kind of alternative to the polynomials that might require large values of k. This is the case of piece-wise linear interpolation [BTU04], a method that dates back to the Babylonian times [Ram99]. Given a function $f(x)$ that originates from a sequence of samples $\forall y_n = f(nT), n \in \mathbb{Z}$ with step value $T \in \mathbb{R}^+$, the interpolated function $f_T(x)$ is built as

$$f_T(x) = \sum_{n\in\mathbb{Z}} y_n \Lambda\left(\frac{x}{T} - n\right) \tag{21}$$

where the linear B-spline takes the form

$$\Lambda(x) = \begin{cases} 1-|x| & |x| \leqslant 1 \\ 0 & |x| > 1 \end{cases}. \tag{22}$$

This linear B-spline (22) is sometimes called 'hat' function [Ram99], as it takes the form described in figure 3. This specific approach has a direct influence on the leietanic function (14). When $L(\boldsymbol{x})$ works in a feature space $X \subset \mathbb{R}$ so its dimension is $F = 1$ and we choose $S = 1$, then it takes the form

$$L(x) = \lambda_a x + \sum_{r=1}^{R} \xi_r |\lambda_s x + \beta_r| \tag{23}$$

where instead of working with matrices and vectors, we work directly with scalars like the unique values $(\lambda_a, \lambda_s) \in (\mathbb{R}, \mathbb{R})$ and the series of R pairs of scalars $\forall(\xi_r, \beta_r), r \in [1, R] : R \in \mathbb{N}^+ \wedge (\xi_r, \beta_r) \in (\mathbb{R}, \mathbb{R})$. In this special case, function $L(x)$ (23) is a linear interpolator with R segments that change their slope at every given point β_r. A similarity can be identified with the terms $|\lambda_s x + \beta_r|$ and the 'hat' function (22), with the difference (among others) that the coefficients β_r are not constrained by any specific distance between them. Another obvious difference is that the 'hat' function is 0 for every $|x| > 1$, whereas the absolute value applied in (23) continuously decreases or increases depending on the sign of x. Function (23) is also differentiable, although its derivative is not continuous at the R points that satisfy $\lambda_s x + \beta_r = 0$. However, the piece limits delimited by every β_r can be learnt during the training of the model, similar to any other parameter. This

applies not only to (23), but also to the original leietanic function (14). All this is part of the design decisions of the leietanic function $L(\boldsymbol{x})$ so it can be flexible when modelling a problem's logic into arithmetic operations that map to space Q. It is not the intention to provide complete freedom to $L(\boldsymbol{x})$ for capturing every logic and its possible exceptions, but to find the smooth [GG12] middle point that balances a compact representation of the logic with the avoidance of overfitting.

Another way of interpreting expression (23) is as a linear combination of R non-static but parametric wavelets in a discrete wavelet transform [ZZ19]. This analogy is not perfectly accurate as it does not pretend to be it, but rather provides an intuitive interpretation of the leietanic function from the angle of approximating functions using linear combinations of wavelets. Once again, this approach has the advantage of using a differentiable expression with a differentiable (incomplete) base of functions $|\lambda_s x + \beta_r|, \forall r \in [1, R]$ that attempts to imitate an orthonormal basis but without being complete or satisfying all the requirements of a canonical discrete wavelet transform.

3.5. Generating low-entropy isolations and compact logics

The logic replicant uses the leietanic function and the vortices to complement one another and obtain compact logics that generate isolated regions of low entropy. The leietanic function can model the space Q with a high degree of freedom. As demonstrated in section 3.4, it can approximate curves in one dimension; however, the leietanic function is not limited to this and can also approximate functions of several real variables. Consequently, when it transforms the feature space X into the space Q, the resulting distribution of quons may be perfectly compact or, alternatively, diffuse and mix instances from different classes.

The leietanic function is highly flexible when transforming instances from the feature space X, and the more parameters S and R the leietanic function (14) has, the more expressive and capable it is of differentiating the quons. Because of this, an element is required to guide it in generating isolated low-entropy regions. This is the role of the vortices; they create isolations within the scalar field, constraining the leietanic function to adapt the quons to these specific areas for proper classification. The isolations in the space Q are the areas where the scalar field for one class is greater than the others. A reasonable (though not perfect) rule of thumb is that each vortex can generate only one area, or sometimes none. It is also possible to combine several closely situated vortices to form a single isolated area. The fewer vortices there are, the fewer distinct areas can be created for each class. In this manner, the vortices enforce a compact expression, compelling the leietanic function to map instances that may be significantly different into quons confined to restricted areas.

Furthermore, since the scalar field is a smooth and continuous function, properly isolating the quons requires that the distances between them to be greater for quons belonging to different classes. If the regions of the scalar field formed by the vortices are not particularly large, this also forces quons of the same class to be close together, forming compact groups. Both requirements -for isolating and compacting the quons- are introduced by the vortices, thereby limiting the flexibility of the leietanic function in modelling the space Q. The vortices compel the leietanic function to model a logic that is compact; otherwise, the classification would be incorrect. In the absence of vortices, the leietanic function would not be required to form any compact logic; however, having relatively simple isolations within the scalar field forces the leietanic function to conform to these requirements.

It is worth noting that with very few parameters S and R for the leietanic function (14), its modelling capability may be limited, whereas an excessive number of vortices could generate scalar fields capable of approximating functions of $D = \dim(Q)$ real variables. Although this is mathematically possible, it is advisable not to use too many vortices in order to maintain a simple and compact logic. This simplification and compaction of the scalar field is balanced by allowing the leietanic function to utilise as many parameters as necessary to be sufficiently flexible to satisfy the constraints imposed by the scalar field for forming compact and isolated groups.

3.6. DFSM

Before entering some practical applications, it is interesting to explore the logic replicant's theoretical computational capacity, as it is a complete DFSM [WSC19]. Intuitively and oversimplifying it, this could be understood as the logic replicant being able to learn any possible problem where its inputs are a finite number of discrete finite variables, and the output is a finite number of classes. More formally, from [LY96], we can borrow their definition of a Mealy machine [Mea55],

Definition 3. A DFSM consist of a quintuple $(I, O, S, \delta, \lambda)$ where I is the set of all input symbols, O is the set of all output symbols, S correspond to the set of all possible states, being $\delta : S \times I \mapsto S$ the transition function for the states and $\lambda : S \times I \mapsto O$ the output function.

As the logic replicant accepts any finite number of real variables, any the input symbol $\forall \iota \in I$ can be assigned to the feature $x_I \in [1, \ldots, |I|]$, where $|I|$ is the cardinality of I. This means that every input symbol ι can be codified in the feature x_I by assigning an integer to it, so its values goes from 1 up to $|I|$. The same approach can be applied to the current state, introduced as feature $x_S \in [1, \ldots, |S|]$. It does not matter that we use the same codification based on integer for different features. To learn the logic behind the transition functions δ and λ, it is necessary to join both in just one transition-output function $\omega\colon S \times I \mapsto S \times O$. This transition-output function ω is simple the application of δ and λ to any element $(s, \iota) \in S \times I$, producing a tuple of results $(s, o) \in S \times O$. The Cartesian product of $S \times O$ is mapped to different output classes c of the replicant. The replicant can learn this transition-output function ω for every input $(s, \iota) \in S \times I$, providing the corresponding class c that univocally identifies a concrete ordered pair $(s, o) \in S \times O$, with as many classes c as $|S \times O|$ (the cardinality of $S \times O$). Because both S and I are finite, it is also the total number of classes c. With this in mind, the leietanic function can take the following form for a two-dimensional case:

$$L(x_I, x_S) = (x_I, x_S)\,. \tag{24}$$

It is interesting to note that, in this case, the transformation function is not even required to be leietanic, but we will enter later on this requirement and why, in general, is extremely convenient. On another note, our demonstration is based on the two-dimensional case, but can be easily extended to any superior number of dimensions, which also includes the case of a one-dimensional leietanic function without losing any rigor. As in (24), because both x_I and x_S are discrete, every value (x_I, x_S) is completely equivalent to a table of size $|S| \times |I|$, with $|S|$ rows and $|I|$ columns, but is represented in a two-dimensional plane $\mathbb{N}^+ \times \mathbb{N}^+$. Now, when the vortices enter into the game, having $|S| \times |I|$ possible output values from L, it is necessary to use $|S| \times |I|$ vortices whose centres are exactly these values. Every ith-vortex's centre can be defined as a two-dimensional vector $\boldsymbol{v}_i = (u_{iI}, u_{iS})$, intensity $\Phi_i = 1$ and density $\rho_i = 1$. With such a distribution of vortices, the centre (u_{iI}, u_{iS}) of the ith-vortex will be placed into a concrete possible input (x_I, x_S). This means that every possible (x_I, x_S) will be exact to one (and only one) ith-vortex (u_{iI}, u_{iS}) and, according to its scalar field function (15), its value will be $U_i(x_I, x_S) = 1$. Any other possible jth-vortex will be at a distance equal to or greater than 1 from the ith-vortex, being its scalar-field function (15) $U_j \leqslant 1/2$. This systematically guarantees that $U_i(x_I, x_S) > U_j(x_I, x_S)$, which is the maximum scalar field, so (x_I, x_S) is classified as the ith-vortex assigned to class c, which ought to correspond to the output order pair of symbol and state (o, s). It is interesting to note that the logic replicant, instead of requiring two units for learning two different logics δ and λ, can synthesise both just into another logic ω which is the one to be learned.

Let us now explain the implications of a logic replicant being able to replicate a DFSM. DFSMs can reproduce any combinational logic [ZRK20], which is an important feature because this is one of the types of logic that it is intended to be replicated. This means, that in theory, the logic replicant should be able to learn any combinational logic simply by having a sufficient number of parameters. In the real world, this type of logic can be found, but it is frequent to find other problems where the input features are not discrete but continuous values. To validate the capacity of the logic replicant to solve such problems, the following section introduces a more practical validation with a selection of specific datasets.

3.7. Training of the logic replicant

One of the characteristics of the logic replicant is its differentiability, although its derivative is not always continuous in every case, but in the majority it is. This characteristic is important because, as Nocedal and Wright [Wri06] state, 'it is not possible in general to identify a minimizer of a general discontinuous function'. In the case of differentiable models, optimizations methods such as quasi-Newton [Sch01] or gradient descent [Rud16b], among others, can be employed. Furthermore, gradient descent is widely preferred for certain ML algorithms (such as neural networks) due to its efficiency and scalability [GBCB16, Bot12], particularly in high-dimensional parameter spaces. The differentiability of the logic replicant facilitates converge towars a good local or a global optimum, whereas discontinuities complicate the search for an optimum. Additionally, differentiable models provide a natural framework for sensitivity analysis [BP16], helping to analyse how small pertubations in the inputs affects the output. Although it is possible to use a generic method such as grid search or evolutionary strategies for training the logic replicant, these typically rely on heuristic evaluations over the entire parameter space, making them computationally expensive and often impractical for high-dimensional problems [Bot12].

Our choice for training the logic replicant is the adaptive moment estimation (Adam [KB14]) variant of the stochastic gradient descent. In terms of optimisation, it achieves the best balance between training times and model performance, but also for showing excellent results in several ML problems [HLH21]. Adam [KB14] is a combination of the root mean square propagation method (invented in 2012 by James Martens and Ilya Sutskever) and the main feature of the momentum method (introduced by Rumelhart, Hinton and

Williams for backpropagation). Adam performs an average of the moments and second moments of the gradients with exponential forgetting, implemented as:

$$m_\alpha^{(t+1)} = \beta_1 m_\alpha^{(t)} + (1-\beta_1)\nabla_\alpha H^{(t)} \tag{25}$$

$$v_\alpha^{(t+1)} = \beta_2 v_\alpha^{(t)} + (1-\beta_2)\left(\nabla_\alpha H^{(t)}\right)^2 \tag{26}$$

$$\hat{m}_\alpha = \frac{m_\alpha^{(t+1)}}{1-\beta_1^t} \tag{27}$$

$$\hat{v}_\alpha = \frac{v_\alpha^{(t+1)}}{1-\beta_2^t} \tag{28}$$

$$\alpha^{(t+1)} = \alpha^{(t)} - \eta\frac{\hat{m}_\alpha}{\sqrt{\hat{v}_\alpha}+\epsilon} \tag{29}$$

where α represents every coefficient of the logic replicant adjusted during the training, $m_\alpha^{(t)}$ is the moment of the coefficient at iteration t, $v_\alpha^{(t)}$ is the second moment of the coefficient at iteration t, $\nabla_\alpha H^{(t)}$ is the partial derivative of the cross-entropy $H^{(t)}$ at iteration t with respect to the coefficient α, and β_1 and β_2 are the forgetting factors of the first and second moments respectively. It is important to mention that while α represents any of the coefficients of the logic replicant, β_1 and β_2 are constants with fixed values $\beta_1 = 0.9, \beta_2 = 0.99$. The concrete expression of the cross-entropy for the logic replicant takes the form

$$H(\boldsymbol{Y},\boldsymbol{T}) = -\sum_{i=1}^{N}\sum_{j=1}^{|C|} t_{ij}\cdot\log\left(\sigma_j(\boldsymbol{u}_i)\right) + \left(1-t_{ij}\right)\cdot\log\left(1-\sigma_j(\boldsymbol{u}_i)\right) \tag{30}$$

where $\boldsymbol{Y}$ and $\boldsymbol{T}$ are $N\times|C|$ matrices representing the predicted values $\boldsymbol{Y}=(y_{ij})$ and target values $\boldsymbol{T}=(t_{ij})$, considering N instances (index $i\in[1,N]$) and $|C|$ classes (index $j\in[1,|C|]$). The function $\sigma(\boldsymbol{u}_i)$ is the softmax function applied to the output of the jth-vortex, considering the vector $\boldsymbol{u}_i$ as the output of all the $|C|$ vortices for the ith-instance $\boldsymbol{x}_i$. That is

$$\sigma_j(\boldsymbol{u}_i) = \frac{\mathrm{e}^{u_{ij}}}{\sum_{j'=1}^{|C|}\mathrm{e}^{u_{ij'}}} \tag{31}$$

where every u_{ij} is the scalar field of the vortices of class j for the ith-instance; therefore, so $\boldsymbol{u}_i = (u_{i1}, u_{i2}, \ldots, u_{i|C|})$, and every jth-class-scalar-field is calculated using (16).

We have changed the way Adam is evaluated for training of the logic replicant. To achieve faster convergence and avoid extreme values of $\nabla_\alpha H^{(t)}$ in (25) and (26), we modulated these values by using the alternative expressions

$$m_\alpha^{(t+1)} = \beta_1 m_\alpha^{(t)} + (1-\beta_1)\frac{\nabla_\alpha H^{(t)}}{a} \tag{32}$$

$$v_\alpha^{(t+1)} = \beta_2 v_\alpha^{(t)} + (1-\beta_2)\left(\frac{\nabla_\alpha H^{(t)}}{a}\right)^2 \tag{33}$$

where the modulator value a is defined as $a = \max(1, |\nabla_{\alpha_1} H^{(t)}|, |\nabla_{\alpha_2} H^{(t)}|, \ldots)$ where $\{\alpha_1, \alpha_2, \ldots\}$ are all different parameters of the logic replicant. These alternative expressions (32) and (33) use a which is at least 1 or the maximum absolute value that an evaluation $\nabla_\alpha H^{(t)}$ might have occured during iteration t. This guarantees that the increments of $m_\alpha^{(t+1)}$ and $v_\alpha^{(t+1)}$ are not greater than $(1-\beta_1)$ and $(1-\beta_2)$ in every iteration. The same applies when $\nabla_\alpha H^{(t)}$ is negative, avoiding values more negative than (β_1-1) and (β_2-1).

Finally, it is worth mentioning that the batch size used during the training was $N=32$, setting the limit of the training when the cross-entropy of the whole dataset was 0.

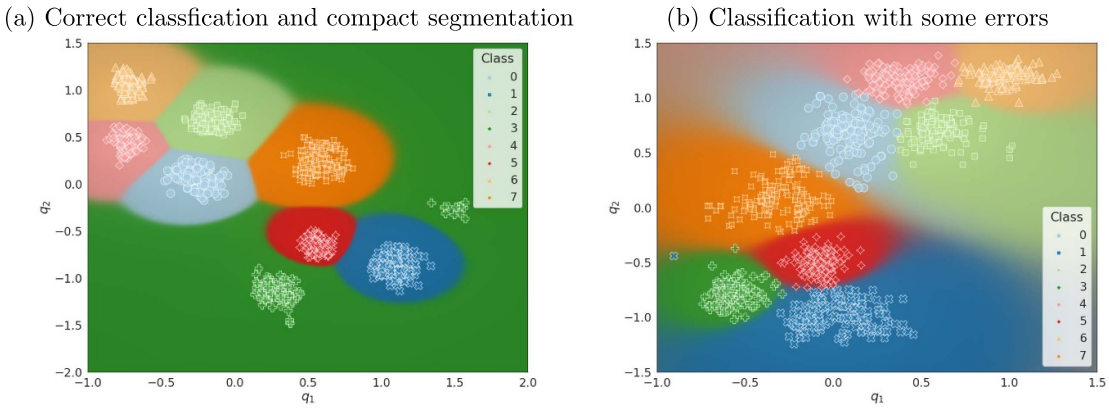

Figure 4. Small dataset with 1080 instances and 8 classes, about a mouse model of Down syndrome 4.2, from Higuera, Gardiner and Cios [HGC15]. Every class has a specific colour used for the instances and the scalar field from the vortices (shown as background).

3.8. *Q*-space visualisation and interpretability

For the specific case of two-dimensional *Q*-spaces, it is possible to generate a visual representation of the instances and the scalar field derived from the vortices. Although this does not replace any evaluation metric, it provides a quick and intuitive means to interpret the problem and the logic applied by the replicant to segment the instances. These plots assign a specific colour to each class, which is applied to both the instances and the scalar field. The scalar field is displayed as the background behind the instances. Thus, when instances of the same colour appear together against a background of the same colour, they can be considered well classified, as the colour (class) of the instances and that of the scalar field match. Conversely, if instances of a given colour (class) appear against a background of a different colour, they are classified as a different class, as indicated by the colour of the scalar field.

An illustrative example of well-isolated segmentation and classification is shown in figure 4(a), whereas figure 4(b) demonstrates a classification with some errors and less well-isolated groups of instances. In figure 4(b), groups of instances belonging to different classes are more closely clustered compared with figure 4(a), where the groups are better isolated. In figure 4(a), the groups are more compact, occupy smaller areas, and the distances between groups are greater and better defined than in figure 4(b). The more compact and well isolated the groups of instances, the more clearly defined is the logic applied by the replicant. If this is accompanied by the correct classification and placement of instances in the appropriate scalar field with the same background colour, it indicates that the replicant is more precise than one with relatively wider groups (in relation to the isolation distance between groups). Finally, figure 4(a) shows that it is possible to have two different groups of instances of the same class, segmented as logically distinct and placed in different areas of the scalar field, despite belonging to the same class. In this case, there are two groups of class 3 that are so distant that instances of other classes are found between them. There is no limit to the number of segments (different groups) that can be found for a class, and this depends directly on the logic applied by the replicant to solve the problem.

In the case of small datasets, this two-dimensional plot of *Q*-spaces is easy to interpret owing to the reduced number of instances. When the number of instances is small, they can be identified at a glance, whereas larger datasets might exhibit more or wider groups of instances. Thus, these two-dimensional plots provide an accessible means of interpreting how a replicant learns a problem defined by a small dataset. For larger datasets, it is also possible to use these two-dimensional plots of *Q*-spaces to visually evaluate the logic applied by the replicant; however, if this logic is not compact and does not generate small groups, more time may be required to examine the different groups and their numerous instances.

4. Validation with practical cases

To illustrate how the logic replicant can learn some problems and be applied in real environments, it will be used in four completely different cases. Each of these problems has its own characteristics, but all of them are selected for the following reasons: (1) the problem is characterised by a small data set, (2) the problems vary from a standard difficulty to be learnt to being extremely difficult, and (3) except for the SOM, the rest of the selected algorithms do not provide an out-of-the-box graphical interpretability, so the current ML algorithms either provide a reasonably good accuracy or graphical interpretability but not both, and sometimes none at all. In addition to these characteristics, the problems can be described as:

- **Parity function**: this is two-category problem with well-defined logic that, however can not be generalised by other classification algorithms. In its 8-variables version it consists of a small data set of 256 instances. Despite its difficulty, this problem is a practical sample where the logic replicant confirms the theory of being a DFSM able to learn any combinational logic, and it predicts with good accuracy where other models fail, but also providing a good explanation of the logic learnt from the problem.
- **Down's syndrome in mice**: a multiclass biological problem involving the expression levels of 77 proteins in mice and a possible treatment with memantine, with 8 possible outcomes. Despite of being a small data set (1080 instances), the logic replicant is able to learn the logic behind the complex interaction of trisomic mice with a drug able to alter their capacity to learn. The logic replicant not only predicts with good accuracy but also provides an interpretation of what has been learnt, which might be of special interest to researchers who need to understand the mechanism that determines the classification as much as having a good classifier. This is a perfect sample of many cases in science, where logic replicants are needed for performance and interpretability.
- **Nucleosynthesis**: a problem from experimental physics with concrete and deterministic logic emerging from the interactions of protons, neutrons and other particles. The generation of atomic nuclei is an important and fundamental process in physics that is expected to be found in the whole Universe. Despite its deterministic behaviour, it is extremely complicated for other classification algorithms to generalise it from learning samples. This is a good sample because it is a tiny dataset with only 23 instances and is extremely difficult. The logic replicant can learn it with a reasonably good accuracy, while it is able to provide a good representation of the learned logic.
- **Optical recognition of handwritten digits**: this is a typical problem related to image recognition, where several ML models use to perform well. The 10 different classes (digits to be recognised) can be written very differently by several individuals or even by the same person. In the logic replicant the leietanic function has to find a logic that is flexible enough to capture all this variability, in contrast to other logics that are more compact. This data set allows us to confirm that the logic replicant also provides excellent results and interpretability in standard ML problems beyond the other algorithms.

The reason for selecting these problems was to cover several aspects related to the design and application of the logic replicant. These data sets cover a compact and deterministic logic (parity function) as well as more organic logic (Down's syndrome in mice). Nucleosynthesis allows to explore how well the logic replicant can deal with a problem with a considerably high number of classes (21), where the input data is very scarce (30 samples) when the underlying logic is well-defined. This dataset will demonstrate the efficiecy of the logic replicant in generalising from little data compared to other ML models. Finally, the optical recognition of handwritten digits allows us to evaluate the logical replicant in a more mature environment, such as image recognition. This provides a reference for the performance of the logical replicant compared to other well-established models.

4.1. Parity function

This function is based on a unique logic that is well-defined and has no exceptions. This is one of the ideal cases for applying the logic replicant because despite it being a binary classification problem, being a DFSM makes it convenient for learning and replicating its logic. This problem is specially interesting as it is not easy to generalize because minimum changes in the input produces a different output. Obviously, a given ML algorithm can memorise every given instance and its expected outputs, but this would give no clues about what might be the output of unseen instances. There is no way to be able to generalize this problem from the given samples without finding a way to learn and replicate its underlying logic.

The parity function is a generalization of the XOR function, with special importance within the domain of the Boolean functions [J+12] and circuit complexity [Vol99]. The parity function f takes $N \in \mathbb{N}$ binary input values, coded as $\boldsymbol{x} \in \{0,1\}^N$ and returns 0 when it has an even number of ones in $\boldsymbol{x}$ and 1 when it contains an odd number of ones. More formally, $f\colon \{0,1\}^N \mapsto \{0,1\}$, where the solution of the discrete equation $\boldsymbol{x}\cdot\boldsymbol{x} = 2n + m\colon\ n \in \mathbb{N}, m \in \{0,1\}$ determines the output of function $f(\boldsymbol{x}) = m$.

From the point of view of the classification models, the root cause of this problem is the difficulty in learning, which underlies the high non-linear separability of the parity function. This significantly reduces the convexity of the solution space. The reason for this is that switching the value of one input variable, produces the opposite output; which complicates the aggregation of instances based on their similarity because it does not provide any information about the classification of an instance.

For this experiment, the 8-bit parity function was selected, which is translated into a dataset of 256 instances obtained from the 2^8 possible combinations of the two values of every 8-bits input. This dataset is generated using a small algorithm that produces these 2^8 permutations and is available online[1].

4.2. Mouse model of Down syndrome

This problem was selected to cover two main characteristics. The first is that it is a real-world problem with its own scientific interest, while on the other hand, it is not necessarily described simply by one specific logic without exceptions, as the data comes from the expression levels of proteins in a complex system, that is, a living creature with all its implicit complexity and the additional factor of reacting to memantine. This is not necessarily non-deterministic, but the variety of mice and circumstances can alter the main logic with greater or minor effects, being a perfect scenario for finding additional factors expressed as secondary logics altering the main one. Additionally, as it has scientific interest in its own, we can see how in the case of a two-dimensional space Q it is easy to graphically show how the model interprets the logic for splitting the different classes, and it is very easy to intuitively understand what differentiates one class from another, but also subgroups within the same class. This is a problem, in which the presence of several vortices per class can segment their subgroups.

Chromosome 21 (Hsa21) has a (complete or partial) third copy, producing the genetic disorder trisomy 21 [Pat09], popularly known as Down's syndrome. Unfortunately, this can result in mild to moderate general learning disabilities in humans, affecting approximately one in 1000 births. Higuera, Gardiner and Cios [HGC15] conducted a preclinical evaluation of memantine in trisomic mice, but also a control group with non-altered genotype, where 77 proteins of mice were measured in a situation of context fear conditioning (CFC). Ts65Dn is a partial trisomy mouse model of Down syndrome that is not able to learn in CFC, unless treated with memantine; these cases are known as rescued learning. However, memantine did not affect learning in the mice of the control group. Higuera, Gardiner and Cios [HGC15] showed that the protein levels in three different cases (untreated Ts65n failing to learn, Ts65n mice rescued for learning and the control group) were significantly different. They used the S1 Dataset[2] that gathers 1080 records associating with the different levels of the 77 proteins to the 8 different classes of mice formed by combining 3 binary characteristics coded as $\{c, t\} \times \{\mathrm{CS}, \mathrm{SC}\} \times \{m, s\}$. These characteristics are as follows:

- *Genotype group* can be control (c) or trisomy (t)
- *Stimulation to learn* defines whether a mouse is allowed to explore and then shocked.
  - Context-shock (CS) is the selection of mice placed in a new cage, allowed to explore, and, shocked several minutes after.
  - Shock-context (SC) is the other group, in which immediately after being placed in the new cage, they are shocked and allowed to explore.
- *Memantine* is a binary attribute that defines whether the mice were injected with memantine (m) or saline (s).

4.3. Nucleosynthesis

This is a multiclass problem with 17 possible classes, with a deterministic underlying logic that responds purely to the basic forces of physics creating new atomic nuclei through nuclear fusion and fission of other nuclei, nucleons and additional particles and energy. Dataset[3] has only 23 input instances for training and testing the models, forcing them to squeeze any information in these records to predict properly. The problem is especially interesting for two main reasons: (1) it is a basic process of the nature expected to be found in the whole Universe and (2) because of its very small size, it helps to evince how good the models are in understanding the data and generalising from it when it is scarce. This dataset contains empirical data observed in the processes of fusion and fission of some atomic nuclei and nucleons (protons and neutrons), which also intervene in additional particles, radiation, and energy release/absorption. It includes the carbon–nitrogen–oxygen (CNO) cycle [Wie18] composed of 4 branches, and also contains the 3 hot CNO (HCNO). The CNO cycle (also known as Bethe–Weizsäcker cycle) describes one type of fusion reaction that occurs in stars when hydrogen is converted to helium using the isotopes of carbon, nitrogen, and oxygen as catalysts. The HCNO cycle [WGU+10] can be found in astronomical objects and events subjected to high temperatures and pressures. These are thermonuclear flashes such as the x-ray burst [CAH+16] occurring on the surface of neutron stars, novae [GS78], and non-terminal thermonuclear eruptions occurring on the surface of white dwarfs that form binary systems [BDV05]. The general processes of fusion and fission are

[1] https://github.com/pedrocorral/8bit-parity-function-dataset.
[2] http://dx.doi.org/10.6084/m9.figshare.1421985.
[3] https://github.com/pedrocorral/nucleosynthesis-dataset.

influenced by the interaction of several fundamental forces, such as electromagnetic force and strong and weak interactions. Despite the different actors that contribute to it, it has been experimentally proven that their behaviour is extremely precise. This means that there is a well-defined primary logic behind the process, without room for exceptions.

Table 2 shows the 23 records used as tiny dataset, where the repeated cases have been removed to avoid duplicates; in this case the cycle is labeled as partial. The CNO cycle can take different paths, with CNO-I having the following cycle of transformations:

$$^{12}_{6}\mathrm{C} \rightarrow {}^{13}_{7}\mathrm{N} \rightarrow {}^{13}_{6}\mathrm{C} \rightarrow {}^{14}_{7}\mathrm{N} \rightarrow {}^{15}_{8}\mathrm{O} \rightarrow {}^{15}_{7}\mathrm{N} \rightarrow {}^{12}_{6}\mathrm{C} \tag{34}$$

starting and ending with the carbon isotope $^{12}_{6}\mathrm{C}$. CNO-II is a minor branch of CNO-I that can be found at Sun's core 0.04% of its time:

$$^{15}_{7}\mathrm{N} \rightarrow {}^{16}_{8}\mathrm{O} \rightarrow {}^{17}_{9}\mathrm{F} \rightarrow {}^{17}_{8}\mathrm{O} \rightarrow {}^{14}_{7}\mathrm{N} \rightarrow {}^{15}_{8}\mathrm{O} \rightarrow {}^{15}_{7}\mathrm{N}. \tag{35}$$

The CNO-III branch is an alternative to branch CNO-II that occurs only in massive starts, starting when oxygen-17 is transformed into fluorine-18 following this sequence:

$$^{17}_{8}\mathrm{O} \rightarrow {}^{18}_{9}\mathrm{F} \rightarrow {}^{18}_{8}\mathrm{O} \rightarrow {}^{15}_{7}\mathrm{N} \rightarrow {}^{16}_{8}\mathrm{O} \rightarrow {}^{17}_{9}\mathrm{F} \rightarrow {}^{17}_{8}\mathrm{O}. \tag{36}$$

The CNO-IV branch is an alternative to CNO-III; therefore, it is also found in massive starts. It forks from CNO-III when from the $^{18}_{8}\mathrm{O}$ is produced a fluorine-19 and a photon as alternative to produce nitrogen-15 and an α particle. This chain of reactions follows the sequence:

$$^{18}_{8}\mathrm{O} \rightarrow {}^{19}_{9}\mathrm{F} \rightarrow {}^{16}_{8}\mathrm{O} \rightarrow {}^{17}_{9}\mathrm{F} \rightarrow {}^{17}_{8}\mathrm{O} \rightarrow {}^{18}_{9}\mathrm{F} \rightarrow {}^{18}_{8}\mathrm{O}. \tag{37}$$

HCNO-I is related to the CNO-I when the $^{13}_{7}\mathrm{N}$ captures a proton $^{1}_{1}\mathrm{H}^{+}$ as an alternative to decay, producing the following process:

$$^{12}_{6}\mathrm{C} \rightarrow {}^{13}_{7}\mathrm{N} \rightarrow {}^{14}_{8}\mathrm{O} \rightarrow {}^{14}_{7}\mathrm{N} \rightarrow {}^{15}_{8}\mathrm{O} \rightarrow {}^{15}_{7}\mathrm{N} \rightarrow {}^{12}_{6}\mathrm{C}. \tag{38}$$

The HCNO-II difference from CNO-II relies on the $^{17}_{9}\mathrm{F}$ capturing a proton $^{1}_{1}\mathrm{H}^{+}$ rather than decaying, so the full sequence is:

$$^{15}_{7}\mathrm{N} \rightarrow {}^{16}_{8}\mathrm{O} \rightarrow {}^{17}_{9}\mathrm{F} \rightarrow {}^{18}_{10}\mathrm{Ne} \rightarrow {}^{18}_{9}\mathrm{F} \rightarrow {}^{15}_{8}\mathrm{O} \rightarrow {}^{15}_{7}\mathrm{N}. \tag{39}$$

Finally, the HCNO-III cycle is an alternative to HCNO-II by the $^{17}_{9}\mathrm{F}$ capturing a proton $^{1}_{1}\mathrm{H}^{+}$, becoming the full process as follows:

$$^{18}_{9}\mathrm{F} \rightarrow {}^{19}_{10}\mathrm{Ne} \rightarrow {}^{19}_{9}\mathrm{F} \rightarrow {}^{16}_{8}\mathrm{O} \rightarrow {}^{17}_{9}\mathrm{F} \rightarrow {}^{18}_{10}\mathrm{Ne} \rightarrow {}^{18}_{9}\mathrm{F}. \tag{40}$$

The 17 classes $c \in C$ to be predicted are the different isotopes found in the presented cycles, that is, $C = \{\ ^{1}_{1}\mathrm{H},\ ^{4}_{2}\mathrm{He},\ ^{12}_{6}\mathrm{C},\ ^{13}_{6}\mathrm{C},\ ^{13}_{7}\mathrm{N},\ ^{14}_{7}\mathrm{N},\ ^{15}_{7}\mathrm{N},\ ^{15}_{8}\mathrm{O},\ ^{16}_{8}\mathrm{O},\ ^{17}_{8}\mathrm{O},\ ^{18}_{8}\mathrm{O},\ ^{17}_{9}\mathrm{F},\ ^{18}_{9}\mathrm{F},\ ^{19}_{9}\mathrm{F},\ ^{18}_{10}\mathrm{Ne},\ ^{19}_{10}\mathrm{Ne}\}$. As this dataset has only 23 instances, it needs to be augmented to generate more training samples without including any of the cases that are predicted. To augment the dataset, it has been considered two concepts from physics that can be applied with full rigour: (1) conservation of the quantum state in the absence of external interaction or spontaneous behaviours and, (2) time reversibility of quantum mechanical systems [JM05]. The conservation of the quantum state allows the use of cases where an isotope $c \in C$ does not change at all its states (no fusion or fission), as it does not have any external interaction or spontaneous behaviour. In that case, when having an isotope $c \in C$ that has not changed at all, it is still c, which can be considered a kind of non-reactive case such as $^{15}_{7}\mathrm{N} \rightarrow {}^{15}_{7}\mathrm{N}$ or $^{18}_{8}\mathrm{O} \rightarrow {}^{18}_{8}\mathrm{O}$, which allows an increase in the base dataset with 17 new instances from every cth-isotope class in C. Despite the time reversibility of quantum mechanical systems [JM05], which has been discussed since 1932 [Wig93], nothing impedes us from training our ML algorithms to predict the outcome of a transformation also as time-reversed. This can be seen as teaching the ML algorithm to predict the outcome, providing its source elements, or predicting the source elements from a given outcome. This added 23 additional instances to the dataset. From each of the given instances (original and augmented), every of the $|C|$ nuclides of the outcome is used as an instance, and the other outcome elements (nuclei, particles, radiation or energy) are considered residual elements: therefore, the problem is given the input of a reaction and the residuals, and the generated isotope should be predicted. E.g., for the case of the CNO-II reaction $^{17}_{8}\mathrm{O} + {}^{1}_{1}\mathrm{H} \rightarrow {}^{14}_{7}\mathrm{N} + {}^{4}_{2}\mathrm{He} + 1.19\,\mathrm{MeV}$, it is given as input both the left part of the reaction $^{17}_{8}\mathrm{O} + {}^{1}_{1}\mathrm{H}$ and the residuals $^{4}_{2}\mathrm{He} + 1.19\,\mathrm{MeV}$, whilst the class to be predicted is $^{14}_{7}\mathrm{N}$. From this reaction it is also a valid instance the same input $^{17}_{8}\mathrm{O} + {}^{1}_{1}\mathrm{H}$ with the residuals $^{14}_{7}\mathrm{N} + 1.19\,\mathrm{MeV}$ for predicting the isotope $^{4}_{2}\mathrm{He}$.

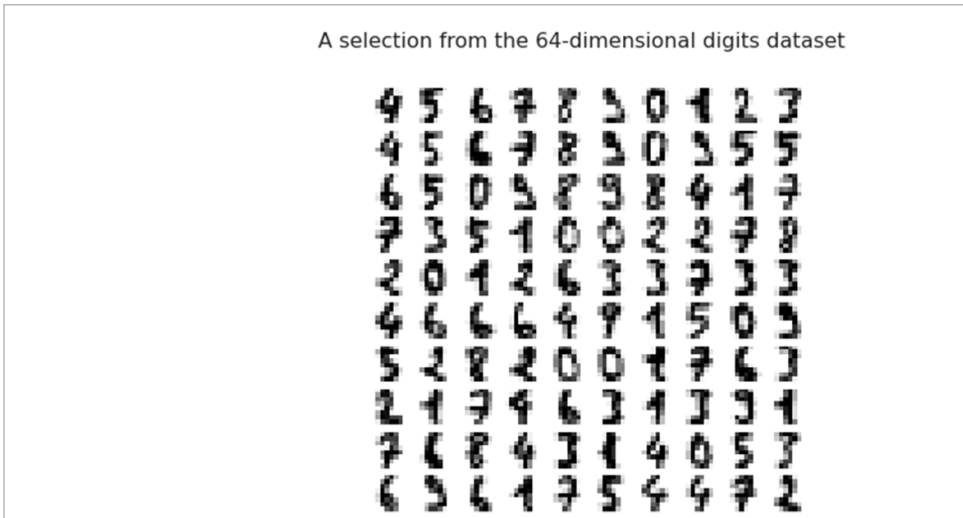

Figure 5. Sampling of 100 instances of digits from the 'Optical recognition of handwritten digits' [DT17]. These cases where normalised from a resolution of 32 × 32 into matrices of 8 × 8 pixels. The original data set includes 5620 instances in total.

4.4. Optical recognition of handwritten digits

This dataset was formed by a selection of 1797 handwritten digits created by 43 people. These digits have been normalised into matrices of 8 × 8 pixels where every pixel's value goes from black to white and is quantified by an integer in the range [0,15]. The data set is publicly available at UCI ML Repository [DT17] with the name 'Optical Recognition of Handwritten Digits'[4]. Figure 5 shows a sampling of 100 of these images on a gray scale.

The goal of this problem is to infer the right class (going from 0 to 9) from a given image. As there is variability in the digits and even between instances belonging to the same class, this makes the logic of this problem less concrete than that of the parity function. The classification logic should be able to cover an area of variability for every class and be especially efficient in defining the limits between classes. This is especially important among these instances, which look similar but belong to different classes. Figure 5 shows some samples that are well-defined and differentiated from other digits, but it can also be found that others are more ambiguous and difficult to evaluate, even for a person. This problem is interesting because it allows a direct evaluation of the logic replicant in a classical task, where other ML models perform well.

4.5. Baseline models

Once the problems to be used for evaluating the logic replicant are defined, let us also introduce the baseline models that will serve as reference for comparing results and behaviour. These models are the MLP, RF, XGBoost and the SOM, all of which were mentioned in the previous sections. These can show the performance of different types of classification models, from easily-interpretable models like SOM, to ensembles of weak predictors and small FNNs as a representation of the state of the art for multiclass classification in small data sets. Although not all possible variants of any possible classification model can be tested, they can provide a good reference for understanding the performance of the logic replicant. The experimental implementation of every one the listed algorithms is described in the following list:

- *RF*: there are several libraries that provide this model with predetermined configurations, so it can be used out of the box. It was used the Python library *scikit-learn*[5], specifically the *sklearn.ensemble.RandomForestClassifier* class configured with default parameters. This allows the RF to create as many nodes as needed to reduce the error with the learning data set as much as possible.
- *MLP*: MLP is used with only one hidden layer and a number of neurons in this hidden layer which is the minimum required for learning the training dataset with the minimum possible error. The hidden layer uses ReLU as activation function, whilst the output layer uses a softmax function. This configuration was implemented using *Keras API*[6] in the *Tensor Flow*[7] python library. As there is not a default configuration for the

[4] https://doi.org/10.24432/C50P49.
[5] https://scikit-learn.org/.
[6] https://keras.io/.
[7] www.tensorflow.org/.

MLP (total number of hidden neurons) suitable for all problems, each experiment has its own configuration. These can be found at the results summary within the table 1.

- *XGBoost*: to implement this algorithm, the Python library *xgboost*[8] was selected, which is similar to the previous ones and allows it to be used the with default configuration. For different experiments, the parameter *max_depth* was changed to obtain the best value for each experiment.
- *SOM*: the evaluation of the SOM was performed using a Python library called *MiniSom*[9] which offers the basic features for working with SOMs, so it required a bit of extra code to support the classification. All the experiments were performed with a 2-dimensional isometric/squared distributions of neurons and the total number of nodes was varied for every one of the experiments to obtain the best results.

4.6. Evaluation metrics

One of the characteristics of the logic replicant is that it requires fewer parameters than other equivalent models, and it would be interesting to use the Akaike information criterion [SIK86] to compare the three different models in every experiment. Unfortunately, the RF cannot be considered a parametric model because the total number of parameters are not fixed initially in the model; therefore, this metric may not be the most convenient for this purpose. In this context, we consider cross-validation to be a good alternative because it is a valid an extended way to compare the results of ML models. More specifically, the selected method for estimating the prediction accuracy for the logic replicant (and the baseline models) is a non-exhaustive cross-validation of repeated random sub-samplings. These random sub-samplings were performed 10 times, where the entire set were randomly divided into a proportion of 80% for the training set and the other 20% for the test set. The performance metrics for both the training and test are based on the average accuracy, precision and recall computed over 10 repetitions. For the training phase, the average accuracy is reported, while for the test phase, the average accuracy, precision, and recall are presented. The metrics evaluated on the test datasets provide insight into the model's generalisation capability.

Although accuracy is the primary evaluation metric, precision and recall are also reported to offer greater transparency and a complementary perspective. *Macro precision* and *macro recall* have been employed to address the multi-class nature of three of the evaluated problems. For each class $c \in C$ the total number of *true positives* TP_c, *false positives* FP_c and *false negatives* FN_c it is calculated. The macro precision is computed as the arithmetic mean of the precision values from each class:

$$\text{macro precision} = \frac{1}{|C|}\sum_{c\in C}\frac{\mathrm{TP}_c}{\mathrm{TP}_c + \mathrm{FP}_c}. \tag{41}$$

The macro recall calculated using the arithmetic mean of the recall from each class:

$$\text{macro recall} = \frac{1}{|C|}\sum_{c\in C}\frac{\mathrm{TP}_c}{\mathrm{TP}_c + \mathrm{FN}_c}. \tag{42}$$

Finally, the main metric, the accuracy is defined as follows:

$$\text{accuracy} = \frac{\text{number of correct predictions}}{\text{total number of predictions}}. \tag{43}$$

The random split (80% training and 20% test) of each of these 10 repetitions was equally used for all three models to be evaluated, the MLP, the RF, the SOM, and the logic replicant; thus, at the end, three of them are evaluated on the same exact data splits. The reason for selecting these same data splits was to reduce the impact of randomness in the final results. This effect is then removed, or at least fairly distributed in an equal manner, for the three models in terms of the data, but it does not completely disappear from MLP and the logic replicant. These two models rely on an initial random generation of their parameters, and because they are linked to completely different mathematical operations, there is no way to ensure a perfectly balanced selection of these initial parameters. It is considered that having 10 evaluations per model and experiment can provide some randomness, specially if the model requires more than 100 parameters (i.e. the case of MLP). For the logic replicant in the parity function, where it requires fewer than 100 parameters, it is easier to fall into an initialisation that is far or even opposite to the optimal for the logic replicant, so the training method has an added challenge to try to fix this and find the best solution. Applying this approach for every classifier and experiment, which is specifically translated for the parity function as 205 training instances for training and 51 instances for testing. Down's syndrome in mice corresponds to 864 instances for

[8] https://xgboost.ai/.
[9] https://libraries.io/pypi/MiniSom.

training and 216 instances for testing. Nucleosynthesis, using data augmentation, ends in 204 instances for training and 5 for testing. Finally, the optical recognition of handwritten digits was split into 1387 instances for training and 410 instances for testing.

Although this accuracy metric is simple, we consider it sufficient for a valid comparison of the models, being objective regarding their strengths and weakness, but also a good starting point for the analysis of the results.

5. Results

After evaluating the different ML algorithms on every dataset, as expected, the accuracies obtained differed between training and test subsets. In some specific cases, such as the parity function, this difference can be very high compared to other datasets. Table 1 summarises the results of this study. The column **Problem** indicates the problem evaluated with every ML algorithm from our selection (section 4.5) that were presented at the state of the art (section 2). The column **Model** determines for every row what model (RF, MLP, logic replicant, SOM or XGBoost) has been used for a given problem. The column **Accur. training** shows the average numeric value of the accuracy for every training combination of *problem-model*. In an equivalent way, the column **Accur. test** shows the average accuracy for every tested combination of *problem-model*. The column **Precis. test** contains the precision obtained from the test datasets, whereas the column **Recall test** shows the recall results for the same test dataset. The last column **Configuration** is very problem-dependant and shows the model personalised configuration that was used during the experiment. For the case of MLP this column indicates how many neurons it has in its hidden layer. For the case of the logic replicant, the **Configuration** column shows the values of the numbers S, R used within the leietanic function (14), the dimension D of the space Q, $D = \dim(Q)$, and the used total number of vortices $|V|$. For the case of SOM, it indicates the total number of neurons, but also the dimensions of the grid. For the cases of RF and XGBoost, as it was mainly used the default configuration from the library, the column **Configuration** shows only the parameters that have been altered for improving the results; these are the total number of estimators and the *max_depth* for the XGBoost. In general it can be seen that the different problems are associated to different results and behaviours from the implicated models, but in all cases the logic replicant had the highest accuracy. To understand these differences, the following subsections provide the context to interpret the results on each one of the problems.

5.1. Results for the parity function experiment

This is one of the experiments in which the differences between the evaluated ML algorithms were more significant. In the RF case, it is remarkable that there is a significant difference between the training and test results. The RF is able to memorise the entire training dataset, having a training accuracy of 100% whereas the accuracy from its prediction falls to 0.20%. This is directly produced by the inductive bias of RF [Bax00], which is not suitable for learning the logic behind the parity function but just memorising and group instances that are similar. Unfortunately, for the parity function, if an instance is changed just 1 bit, it also produces a change in its classification. That means that the RF cannot gather several cases within one node as these are never repeated, forcing the model to know every case individually and eliminating any possible capacity for an extrapolation. This is also a problem with XGBoost, where aggregation of instances with similar features is not a suitable strategy for the parity function. In this case, the learning process itself gives a 64% of accuracy, but the test has a 5.10%, which is not as small as the 0% of RF, but is still extremely low.

The results for the MLP evince the same problem of generalization but with a more reduced gap between the accuracies for the training and the test, decreasing the accuracy 38.41% vs the 99.80% accuracy drop of the RF. This explanation is similar despite the fact that the internal processes of the MLP are different. Putting some context into the 57.25% accuracy of the MLP during the test, the baseline probability is just 50% as it is a problem perfectly balanced with just two classes. This may be because the MLP can improve the chances of a purely random model, but to a lesser degree. This suggests that the MLP was not able to properly replicate the logic of the parity function, but was successful in finding a heuristic that slightly improved its prediction.

For SOM the results also showed a low accuracy, from 64.68% during the training to 24.71% in the tests. This is understandable because of the nature of the SOM, looking for a case that is closer to the given instance. Similar, but less acute, to what happens with the RF and the XGBoost, the closest possible is one bit different and belongs to a different class. Therefore, this strategy used for classifying accumulated systematic errors.

The logic replicant, requiring a minimal configuration $S = 0, R = 0, D = 1, |V| = 8$, is able to learn the logic with a high accuracy (99.80% of training accuracy), and extrapolate to unseen results with a small reduction of the accuracy to 98.63%. The reason behind this is the inductive bias [Bax00] of the logic replicant, which can exploit compact logics with a well-defined rules and almost no exceptions. It is

Table 1. Table summarizing the results from the three experiments performed with the logic replicant, the random forest and the MLP. The MLP includes the number of hidden neurons in the parameters' column, whilst the logic replicant includes the parameters S, R, the number of total vortices as $|V|$ (cardinality of V) and the dimension D of the space Q.

Problem	Model	Accur. (train)	Accur. (test)	Precis. (test)	Recall (test)	Configuration
Parity function	Replicant	99.80%	**98.63%**	**98.67%**	**98.60%**	$S=0, R=0, D=1, \|V\|=8$
	RF	**100.00%**	0.20%	0.21%	0.18%	100 estim., all features
	MLP	95.66%	57.25%	42.61%	59.72%	20 neurons, 1 hidden layer
	SOM	64.68%	24.71%	24.71%	18.39%	256 neurons (16×16)
	XGBoost	64.00%	5.10%	4.74%	5.27%	100 estim., max depth = 6
Mouse model of Down S.	Replicant	99.99%	**99.34%**	**99.38%**	**99.40%**	$S=0, R=0, D=5, \|V\|=40$
	RF	**100.00%**	94.31%	95.54%	94.56%	100 estim., all features
	MLP	99.70%	98.89%	98.89%	98.90%	7 neurons, 1 hidden layer
	SOM	92.30%	87.04%	87.79%	87.07%	3600 neurons (60×60)
	XGBoost	99.98%	93.29%	93.50%	93.50%	100 estim., max depth = 7
Nucleo-synthesis	Replicant	99.99%	**72.87%**	**73.56%**	**72.72%**	$S=0, R=0, D=1, \|V\|=17$
	RF	99.17%	8.33%	12.50%	7.25%	100 estim., all features
	MLP	**100.00%**	10.20%	10.48%	11.05%	5 neurons, 1 hidden layer
	SOM	**100.00%**	8.33%	9.65%	8.75%	121 neurons (11×11)
	XGBoost	17.08%	6.67%	8.61%	6.46%	100 estim., max depth = 7
Recognit. of hand-written digtis	Replicant	99.99%	**98.21%**	**98.21%**	**98.21%**	$S=4, R=4, 3\ D=8, \|V\|=20$
	RF	**100.00%**	89.22%	93.38%	89.47%	100 estim., all features
	MLP	**100.00%**	97.49%	97.49%	97.55%	50 neurons, 1 hidden layer
	SOM	93.83%	93.23%	92.28%	93.36%	10 000 neurons (100×100)
	XGBoost	99.64%	93.70%	93.71%	93.89%	100 estim., max depth = 11

important to highlight that these results prove that the logic replicant is the only model that has been able to replicate the logic of the problem, not memorising instances or having a partial understanding of the problem, but providing an equivalent representation of the original logic.

Beyond the general results, it is interesting to take a closer look at some of the two-dimensional solutions of the logic replicant, as it is directly interpretable. Figure 6(a) shows a 2-dimensional scatter plot of instances at space Q, performed by a logic replicant after being trained with the parity function and configuration $S=0, R=0, D=2, |V|=8$. As it can be seen, despite being just a single logic, this logic groups instances with the same number of 0 and 1 bits, that is, values with the same output that also share the configuration. This is because the parity function is symmetric, and permuting two input bits produces the same output. Figure 6(b) shows the placement of several vortices that cover the instances in figure 6(a) in space Q; these vortices should have a spatial distribution that matches the subgroups of the instances at Q. As there are 256 total instances for the 8-bit parity function, split into 9 groups (5 for the class 0, 4 for the class 1), which helps to generalize when evaluating new instances, especially in the groups and vortices where more instances can be found, as only a few cases help to shape them, they can be used for properly classifying new instances.

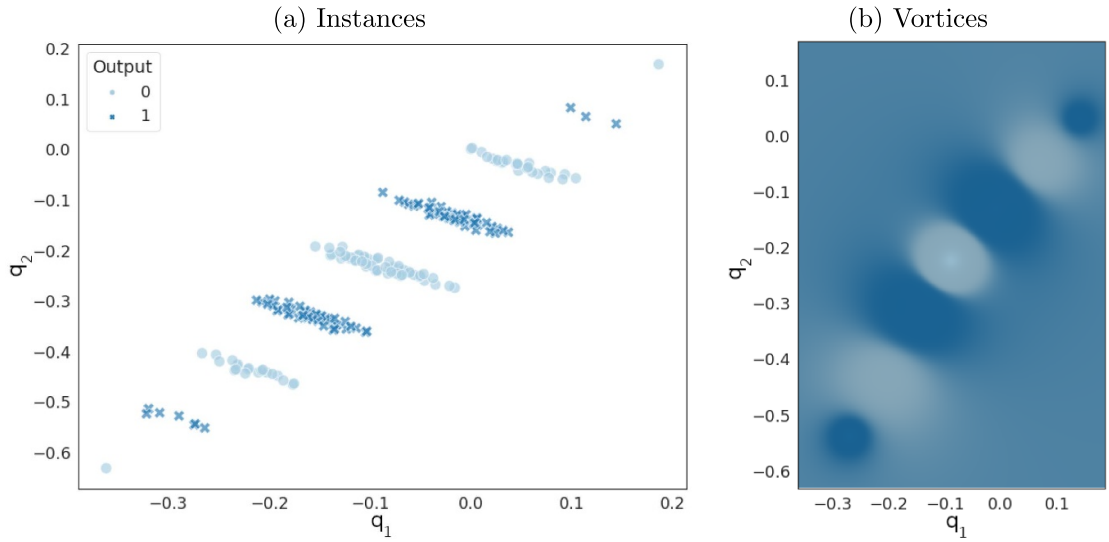

Figure 6. Plots of the instances (a) and vortices (b) from the dataset *parity function*, projected to a 2-dimensional space Q using a leietanic function $L(\boldsymbol{x})$ configured as $S = 0, R = 0, D = 2$ and 4 vortices per class.

5.2. Results for the experiment of Down's syndrome in mice

As this problem depends on several factors involving organic components and living creatures, it is not expected to be described by a completely rigid logic with no outliers. This is a context in which baseline models can deliver their best results. The RF method had the highest accuracy during training of 100%, and a reasonable 94.31% for the test. The difference in behaviour vs the parity function can be explained by the nature of the problem; the current dataset has more similar cases that can be grouped in final nodes. That is, similar input instances can have similar results (except for some outliers) and the RF can exploit that in its internal segmentation. As this problem has up to 77 different explanatory variables, RF can to use several of them in every tree to profile the highest probability of classification. Unfortunately, its error is higher than that of other options, such as the MLP or the logic replicant.

MLP is the second one with better prediction results with 98.89% accuracy. The rationale for this is similar to that expressed in the previous paragraph. This logic allows for the aggregation of similar instances. A small change in the input variables does not necessarily mean a change in the classification; except for some outliers. Because of that, having enough cases for learning allows to safely extrapolate new predictions based on similar learnt cases. Despite the differences in the way of classifying the instances, both the RF and the MLP (but also the XGBoost and the SOM) can use the proximity of the instances at the feature space X for improving the predictions.

The interpretation of the results from the SOM is completely aligned with that from the RF and the MLP. The SOM looks for the closest node for every instance in terms of the Euclidean distance evaluated on the features, which is reasonable for this dataset because, in general, similar instances might share the same class. The dataset has 1080 instances, and the SOM has 3600 neurons available for learning, but it has the lowest accuracies for both training and testing. If we compare this to the MLP, which only has 15 neurons in total (7 in the hidden layer and 8 in the output layer), it shows that MLP has a better performance if we only consider the total number of parameters required (weights from the neurons).

The case of XGBoost is similar to the RF from the results perspective, 99.98% accuracy during the training and 93.29% during the test. Both showed good training, but their performance during the predictions decreased considerably when compared to the other algorithms. The drop in accuracy (from training to test) was 6.69%. for XGBoost, which was the highest of all models. The RF had an accuracy drop of 5.69%, the second highest one, the SOM has a drop in accuracy of 5.70%, and the MLP had a drop of 0.81%, the second smallest one. In this case, the tree-based algorithms seem to learn the training subset better, but they have worse extrapolation than the evaluated ANNs (SOM and MLP).

The results from the logic replicant show the best accuracy during the test, 99.34%, with a drop of 0.64%pt. versus its own training. These two numbers are the best of all tested ML algorithms. To understand the improvement of the logic replicant versus the MLP, is better to focus on the error (100%−accuracy) reduction, because the MLP already has a high accuracy (98.89%). When comparing the errors during the

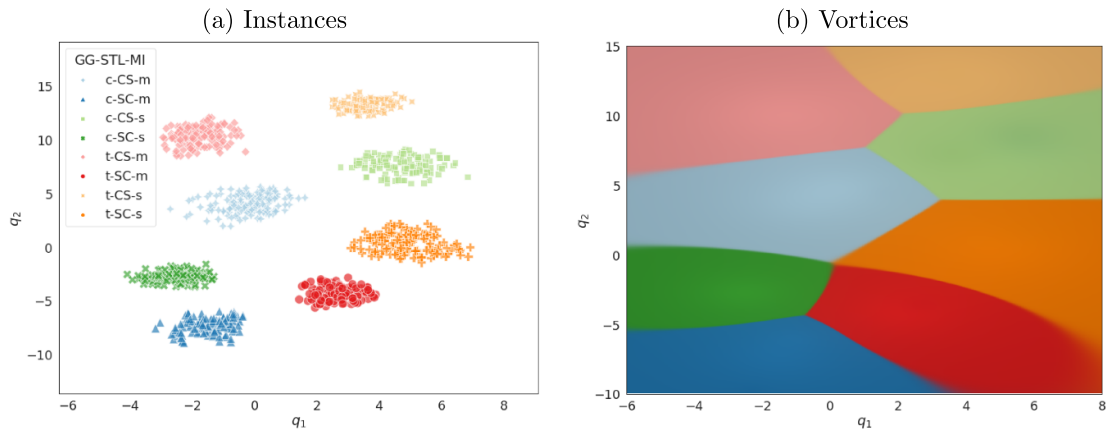

Figure 7. Plots of the instances (a) and vortices (b) from the dataset *Down's syndrome in mice*, projected to a 2-dimensional space Q using a leietanic function $L(\boldsymbol{x})$ configured as $S = 4, R = 4, D = 2$ and 3 vortices per class.

test for the logic replicant and the MLP, we see that the reduction in the error is 40.54%. This allows us to conclude that the logic replicant can take advantage of learning the problem's logic and using it for prediction, despite the fact that it is perfectly possible to find outliers. On the other hand, it can also exploit the similarities of the instances if it is the simplest logic or even combines several compatible logics for projecting to space Q. In datasets such as Down's syndrome in mice, where the other ML models have an already high accuracy, the approach of the logic replicant helps to make the difference and improve the prediction's performance significantly.

To gain a better understanding of the results and the problem itself, it is interesting to review the two-dimensional graphical interpretation of a logic replicant which has learnt this problem (figure 7) using the configuration $S = 4, R = 4, D = 2, |V| = 24$. This configuration is suficcient to reduce several groups to just one big group per class, and if the parameters S and R were reduced, then the distribution would not be class-isolated and some classes would require several groups. This problem is defined by a main logic that splits the different classes, but not the symmetric shape that appears in the parity function. This can be seen in figure 7(a), which shows the equivalent 2-dimensional scatter plot of instances in space Q. As expected, the grouping logic is not as regular as that of the parity function, but main groups related to each class can still be observed. These groups show less symmetry than those in the parity function, but are still relatively compact, with some exceptions found outside the main groups. When comparing the groups formed by the instances with the vortices shown in figure 7(b), it can be seen that the majority of instances within a group match an area covered by a vortex identified with the corresponding class, although the exceptions found away from the vortices of their class are misclassified. The training of the logic replicant attempts to place the vortices as closer as possible to the instances from figure 7(a), and the distribution of the vortices shows that although a clear logic that can be extracted, it exhibits irregular patterns per class, lacking of specific symmetries.

5.3. Results of the nucleosynthesis

The previous problems support the idea that the logic replicant can learn well-defined logics (parity function) but can also deal with several classes (Down's syndrome in mice) in an environment where variability is expected in the cases and, maybe, even in the outliers. The results for the problem of nucleosynthesis show the capacity of the logic replicant for generalisation when having a greater number of classes (17) as possible output, but ruled by a clearly defined logic. It can be argued that 17 classes are not necessarily a great number, but when compared to the total number of instances (23, 204 augmented), the proportion is high. In general, the results show that this is really challenging for all the evaluated ML algorithms.

In this problem, XGBoost is the model with the worst performance, with a test accuracy of 6.67%, but also the lowest training accuracy (17.08%). This problem seems to be quite complicated to learn by the XGBoost. This would require a dedicated analysis in itself, but it seems that the general configuration used does not help XGBoost to deal with a problem in which there are only a few samples of every class. This might be the reason behind, or it is an intrinsic limitation of XGBoost if we consider that this general configuration performed better in all the other evaluated problems.

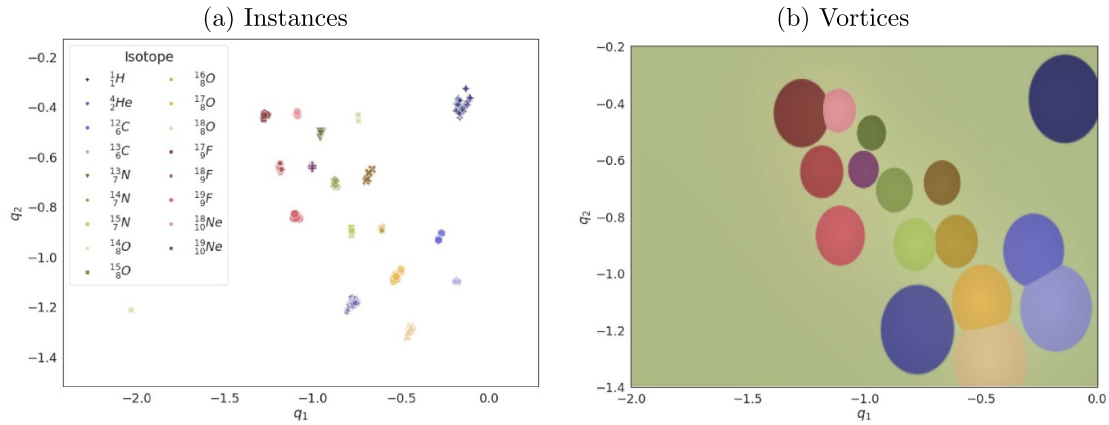

Figure 8. Plots of the instances (a) and vortices (b) from the dataset *nucleosynthesis*, projected to a 2-dimensional space Q using a leietanic function $L(\boldsymbol{x})$ with the configuration $S = 1, R = 2, D = 2$ and 1 vortex per class.

Both the RF and the SOM have the same test accuracy of 8.33% but high training accuracies, that is 100% for the SOM and 99.17% for the RF. It is reasonable to assume that the low number of instances per class does not allow them to learn with consistency. The similarity in the features cannot be used to assume that resembling instances belong to the same class, but rather penalise the model that uses this approach. The reason for this is the same as for the parity function; small changes in the features lead to a completely different classification.

The logic replicant shows the highest accuracy, which is 72.87%, with also a considerably high training accuracy of 99.99% where the difference between them is 27.12%pt. The strategy of finding a common logic is the most suitable in this case, as this problem is the result of the fundamental forces that operate with the particles that constitute the nuclei, and these forces accept no exception. That means that the inherent logic of the problem is clearly compact and can be generalised, so the logic replicant can use this approach as an advantage.

As in the previous problems, the case of the logic replicant in a two-dimensional space Q has special interest because of its interpretability. As shown in figure 8(a), the outcome of a logic replicant trained with 80% of the nucleosynthesis dataset projecting these instances to a 2-dimensional space Q with axes q_1 and q_2, and a configuration $S = 1, R = 2, D = 2, |V| = 18$. The different colours and shapes represent the 18 isotopes involved in the CNO and HCNO cycles with their own variants. The model can group elements of the same isotope in dense areas except for the $^{14}_{8}$O and the $^{17}_{9}$F. Figure 8(b) represents the 18 vortices of the same logic replicant, having a one-to-one correspondence with the 17 class of isotopes, covering a total of 18 groups generated by the leietanic function L as shown in figure 8(a).

5.4. Results of the optical recognition of handwritten digits

As this is an image multiclass classification problem, the results of the ML algorithms are not surprising in terms of high accuracy. Among them, the RF shows the lowest performance with a test accuracy of 89.22% despite its training having 100% accuracy. This is the highest drop in accuracy when moving from training to testing. Although it is a good multiclass classification model [MJZ18], all models show better performance in this particular problem.

The SOM, using 10 000 neurons, was able to achieve a test accuracy of 93.23% with a reduced drop from the training of 0.60%pt., which is the smallest one. Despite being the second-worst result in absolute prediction, there wass no significant loss in performance when moving from training to test compared to the other models. This is the best scoring of the SOM in all the different analysed datasets, and it is not very distant from other ML algorithms.

XGBoost is the one in the middle, having the third worst accuracy test with 93.70%. This is significantly better than the other tree-based models, the RF, and more or less close to the SOM (93.23%), but far from the first- and second-best performing cases.

The highest accuracies during the test were obtained by the MLP (97.49%) and the logic replicant (98.21%), with the latter having the highest one in the test. Although the MLP does a good job, the logic replicant is able to exploit its internal logic representation in an 8-dimensional space Q that allows to easy

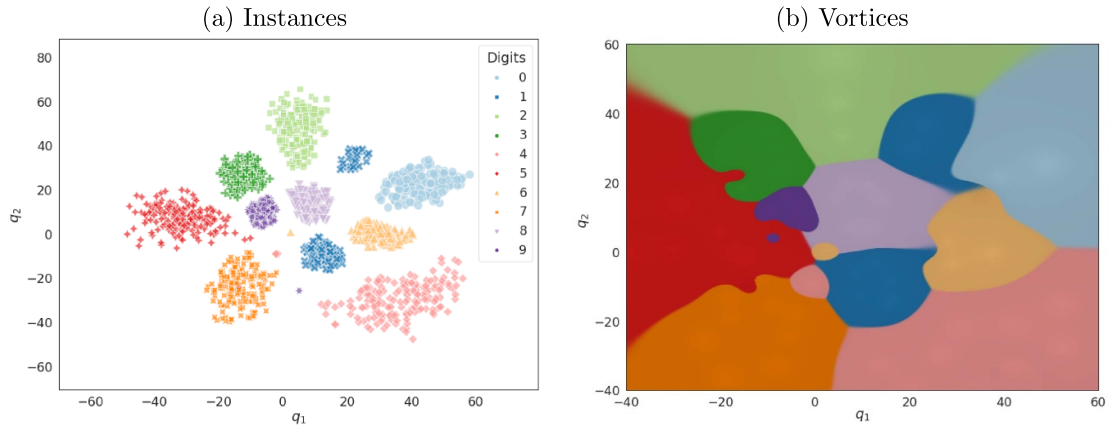

Figure 9. Plots of the instances (a) and vortices (b) from the dataset *optical recognition of handwritten digits*, projected to a 2-dimensional space Q using a leietanic function $L(\boldsymbol{x})$ restricted to the configuration $S = 10, R = 10, D = 2, |V| = 50$.

splitting of the instances in no more than 2 groups per class. Despite the accuracies from both are high, it means reducing the error (100%−accuracy) 28.69% compared to the MLP.

As in the previous subsections, it is interesting to analyse the logic replicant in 2-dimensional space Q. This can be seen in figure 9(a), which shows how well the problem is learnt; thus, the logic replicant is able to learn the logic that splits the different classes in different groups. The configuration used for the trained replicant is $S = 10, R = 10, D = 2, |V| = 50$. The leietanic function is able to properly discriminate the majority of the instances creating well-delimited groups, despite clear irregularities in their shape and size. Digit 1 forms a pair of groups, while the rest have only 1 group. A few outliers can also be observed from this logic, appearing as isolated instances of the digits 4 and 9. In addition, figure 9(b) shows the distribution of the vortices at this 2-dimensional space Q, delimiting the space that every class takes. This latter figure 9(b) shows how there is a big logic that distribute them all-around the 2-dimensional plane, but evincing a clear irregularity in the borders among classes. The digit number 1 requires two main groups, and some vortices from this class are completely isolated the ones from the others. That means that there are general exceptions in the found logic, or, in other words, sub-logics nested into the main one.

5.5. Analysis of results

The review of the results allows the evaluation of the logic replicant from two different angles its accuracy and interpretability. To understand the former, it is especially relevant to the cases of the parity function and nucleosynthesis because of the higher accuracy of the logic replicant compared to the other ML algorithms. This is mainly due to the compact logics found within the parity function and nucleosynthesis. The logic replicant performs better when these logics are completely free of exceptions, as can be easily seen for the parity function in figure 6(a), which shows regularity and a well-structured distribution of the instances. Because these problems use discrete variables, it helps the logic replicant is a DFSM. As we can see with the other problems (*Down's syndrome in mice* and the *optical recognition of handwritten digits*), containing continuous variables does not mean that the problem cannot be learnt, but that the distribution of the continuous values should be aligned with the distribution of the output classes (figures 7(a) and 9(a)). If this requirement is not satisfied, then a distribution of the input values in space Q will present frequent changes in the output classes. On the contrary, this is exactly what happens to the parity function and nucleosynthesis, but in these cases, the categorization of their input variables allows the logic replicant to segment and find a suitable logic for reproducing these problems. We assume that this simple logic behind the parity function may be better represented by a DFSM that uses significantly fewer parameters. This may be explained by the fact that fewer parameters provide less room for overfitting than other models with more parameters.

The other important aspect of the logic replicant is its interpretability, both for knowing the quality of the learning of the model and for understanding the problem itself better. When it is used in a 2-dimensional space Q, the transformed instances $\boldsymbol{q}$ are shown segmented by a logic that isolates classes, but it can also be seen where the vortices are placed delimiting these classes. This is relevant when there is an interest in going beyond the black-box prediction of new instances and the problem requires deeper analysis. Figures 6(a) and (b) show the regularity and symmetry of the logic behind the parity function, whereas Down's syndrome (figures 7(a) and (b)) evinces an aggregation logic that is not perfectly regular, showing some subgroups

within the same class. This is exactly what could be expected from a case with an expected primary logic that may be affected by other factors, such as different mice, different responses to the drug and other characteristics. As interpretability is an important factor in the case of Down's syndrome in mice, the logic replicant is useful for understanding how the learned logic groups same-class instances and how it produces some irregularities in their form. Could this be an indicator of other factors affecting the classification? Finally, this capacity to graphically plot the projected results and learned vortices in the 2-dimensional space Q also allows an intuitive understanding of how well the model discriminates every class from the others, or if there are some overlapping that causes incorrect classifications.

It is worth mentioning that applying PCA to the dataset and taking the two main components is another way of representing the problem in a 2-dimensional space, but this is not what the logic replicant does. The PCA is not intended to find the low-entropy isolations and split the classes into isolated areas as the leietanic function does. At this point, the logic replicant has the advantage of providing a 2-dimensional representation that directly shows the classification logic of the problem applied to all the given instances, but also the distribution of the vortices that explain where the limits of the classes are and how new instances would be classified with the learned logic.

5.6. Summary

The experimentation with the different considered problems has shown how the proposed ML algorithm, the logic replicant, provides a better prediction capacity for multiclass classification in the selection of small datasets. The selected four different datasets are a representative selection of different types of problems to show the predictive power of the logic replicant in small datasets. The logic replicant learns and replicates compact logics with better accuracy than other ML alternatives such as RF, one-hidden-layer MLP, XGBoost or SOMs. The logic replicant is not limited to these compact logics and learns more regular problems improving the prediction accuracy compared to the baseline models. On the other hand, when the interpretability of the problem is a key factor, the two-dimensional space Q allows the logic replicant to be completely interpretable in two main aspects: (1) by understanding the problem's logic and how the learnt instances are distributed at the space Q revealing a class-based coherent segmentation; (2) showing whether the learning process was correct and all the instances where properly learnt or if there were outliers or misclassified instances. This visual interpretation of the logic replicant is quick and intuitive, simply identifying if there are misclassified instances or, on the contrary, all the LEIs contain only elements of the same class.

The leietanic function (14) in combination with vortices (15) has been proven to be a DFSM that can replicate the inherent logic of several different problems in a practical way, proving that this approach is not only possible but also effective. These results are good in small datasets and especially outperforming when the problem has a well-defined logic (e.g. the parity function and nucleosynthesis problems in sections 4.1 and 4.3). This is a solid foundation for continuing the research in this direction, extending it to other problems and greater data-sets to obtain better prediction accuracies.

One important conclusion from this work is the identification of problems that cannot be generalised by other ML approaches; however, the logic replicant understands and predicts them properly with only some instances. The problem of nucleosynthesis is especially remarkable in this context as it only responds to deterministic natural laws and has scientific interest in itself [CFOY16], which is a sample case, but demonstrates that the logic replicant is a great option in similar cases where the availability of data is limited.

6. Conclusions

In this article, we proposed and demonstrated the viability of a new ML algorithm designed to work with small datasets for: (1) providing a consistently good performance when predicting multiclass classification, and (2) offering a graphical interpretation of the classification logic that explains the problem as well as the quality of the learning itself. This has been achieved by applying a design based on the differentiability of the model, so it can be trained using gradient-descent optimisations, but also being a DFSM that (in theory) can reproduce any deterministic logic based on categorical variables; however, in practice, it also works well with other kinds of problems. This is performed using a (leietanic) function that replicates the intrinsic logic of a problem using linear interpolation to generate isolated groups of instances with low entropy.

The logic replicant obtains better results in the selection of small datasets than some standard ML algorithms, but beyond this point, it has been identified some problems (*parity function* and *nucleosynthesis*) cannot be generalised by other ML approaches while the logic replicant understands and predicts them properly with only some instances. The problem of nucleosynthesis is especially remarkable in this context, as it only responds to deterministic natural laws and has scientific interest in itself [CFOY16], demonstrating that the logic replicant is a great option in similar cases where the availability of data is limited. This is a solid

foundation for continuing the research in this direction, extending it to other problems and greater datasets, to obtain better prediction accuracies.

6.1. Future work

After establishing a solid first step by introducing the logic replicant, our future work is motivated by the desire to improve it from two distinct angles. On the one hand, we aim to discover mathematical expressions and algorithmic implementations that further enhance the prediction capacity of the logic replicant. There exist alternative mathematical formulations for the leietanic function and the vortex's scalar fields, and a reasonable hypothesis is that some of these alternative designs or implementations might offer better generalisation, leading to higher prediction accuracies.

The other primary direction for future research is to adapt the replicant to tackle problems involving larger datasets. To achieve this, the aforementioned improvements to the model itself would naturally scale the size of the problems it can handle; however, the computation times associated with medium and large datasets must be optimised. To address this, we plan to develop a speed-optimised Python library that facilitates experimentation with larger datasets while maintaining reasonably low computation times. This library will be developed using a high-performance programming language such as C, which allows effective utilisation of the CPU cache (L1, L2 and L3) [TASS07] to reduce memory access times. The library will also incorporate Advanced Vector Extensions (AVX2 and AVX-512) [Lom11] to perform faster single operations on multiple data (SIMD) [Lom11], processing eight 32-bit single-precision floating-point numbers in parallel. Together, these two optimisations are expected to improve computation speed by approximately eight times, not only due to the parallel calculations but also through the optimal utilisation of cache memory. This will enable the processing of more data in less time, which is beneficial for training logic replicants with larger datasets.

Following this low-level CPU optimisation, the library will also be ported to GPUs [MRT19] for even faster computation times. This approach is feasible because Adam optimisation permits some calculations to be performed concurrently rather than sequentially. In particular, the leietanic function (14) includes several linear algebra operations that can be parallelised -especially for larger values of F, S, R or D- using GPUs. Larger datasets not only imply more instances but often an increase in the dimensionality F of the feature space X as well. The greater the number of parallel mathematical operations, the more advantageous the use of GPUs becomes in reducing computation times during the training of the logic replicants.

7. Limitations

The main limitation nowadays may occur in theory if we deal with big datasets. In these cases the parameters S and R in the leietanic function (14) could become large and its computation generates 'NaN' values. Another theoretical limitation may occur for big datasets with many of exceptions to the main intrinsic logic, so the leietanic function (14) would require a large number of local minima and maxima, as the linear interpolation is not sufficient for replicating them properly. Despite these limitations are just theoretical, a small part of our future work will be to identify whether these really happen, and under what circumstances.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://github.com/pedrocorral/logic-replicant-datasets.

Acknowledgments

This work has been financed by the Spanish Ministry of Science, Innovation and Universities (project FairTransNLP PID2021-124361OB-C32) funded by MCIN/AEI/10.13039/501100011033 and by ERDF, EU A way of making Europe.

Appendix. Nucleosynthesis dataset

In this appendix, table 2 shows the known reactions included in the nucleosynthesis dataset covering fusion and fission of stable and unstable nuclei, as well as positrons e^+, neutrons (n), gamma radiation γ, electron neutrinos ν_e and some energy release (normally positive). Some cycles are shown only partially for not repeating the reactions present in the process listed in the same table. This includes 4 branches of the CNO cycle and 3 additional HCNO cycles.

Table 2. Known reactions included in the nucleosynthesis dataset formed with the 4 branches of the carbon–nitrogen–oxygen (CNO) cycle and 3 additional hot CNO cycles.

CNO-I cycle			
$^{12}_{6}\mathrm{C} + {}^{1}_{1}\mathrm{H}$	$\rightarrow$	$^{13}_{7}\mathrm{N} + \gamma$	+ 1.95 MeV
$^{13}_{7}\mathrm{N}$	$\rightarrow$	$^{13}_{6}\mathrm{C} + \mathrm{e}^{+} + \nu_{\mathrm{e}}$	+ 1.20 MeV (half-life of 9.965 min)
$^{13}_{6}\mathrm{C} + {}^{1}_{1}\mathrm{H}$	$\rightarrow$	$^{14}_{7}\mathrm{N} + \gamma$	+ 7.54 MeV
$^{14}_{7}\mathrm{N} + {}^{1}_{1}\mathrm{H}$	$\rightarrow$	$^{15}_{8}\mathrm{O} + \gamma$	+ 7.35 MeV
$^{15}_{8}\mathrm{O}$	$\rightarrow$	$^{15}_{7}\mathrm{N} + \mathrm{e}^{+} + \nu_{\mathrm{e}}$	+ 1.73 MeV (half-life of 122.24 s)
$^{15}_{7}\mathrm{N} + {}^{1}_{1}\mathrm{H}$	$\rightarrow$	$^{12}_{6}\mathrm{C} + {}^{4}_{2}\mathrm{He}$	+ 4.96 MeV
CNO-II (partial) cycle			
$^{15}_{7}\mathrm{N} + {}^{1}_{1}\mathrm{H}$	$\rightarrow$	$^{16}_{8}\mathrm{O} + \gamma$	+ 12.13 MeV
$^{16}_{8}\mathrm{O} + {}^{1}_{1}\mathrm{H}$	$\rightarrow$	$^{17}_{9}\mathrm{F} + \gamma$	+ 0.60 MeV
$^{17}_{9}\mathrm{F}$	$\rightarrow$	$^{17}_{8}\mathrm{O} + \mathrm{e}^{+} + \nu_{\mathrm{e}}$	+ 2.76 MeV (half-life of 64.49 s)
$^{17}_{8}\mathrm{O} + {}^{1}_{1}\mathrm{H}$	$\rightarrow$	$^{14}_{7}\mathrm{N} + {}^{4}_{2}\mathrm{He}$	+ 1.19 MeV
CNO-III (partial) cycle			
$^{17}_{8}\mathrm{O} + {}^{1}_{1}\mathrm{H}$	$\rightarrow$	$^{18}_{9}\mathrm{F} + \gamma$	+ 5.61 MeV
$^{18}_{9}\mathrm{F}$	$\rightarrow$	$^{18}_{8}\mathrm{O} + \mathrm{e}^{+} + \nu_{\mathrm{e}}$	+ 1.656 MeV (half-life of 109.771 s)
$^{18}_{8}\mathrm{O} + {}^{1}_{1}\mathrm{H}$	$\rightarrow$	$^{15}_{7}\mathrm{N} + {}^{4}_{2}\mathrm{He}$	+ 3.98 MeV
CNO-IV (partial) cycle			
$^{18}_{8}\mathrm{O} + {}^{1}_{1}\mathrm{H}$	$\rightarrow$	$^{19}_{9}\mathrm{F} + \gamma$	+ 7.994 MeV
$^{19}_{9}\mathrm{F} + {}^{1}_{1}\mathrm{H}$	$\rightarrow$	$^{16}_{8}\mathrm{O} + {}^{4}_{2}\mathrm{He}$	+ 8.114 MeV
HCNO-I (partial) cycle			
$^{13}_{7}\mathrm{N} + {}^{1}_{1}\mathrm{H}$	$\rightarrow$	$^{14}_{8}\mathrm{O} + \gamma$	+ 4.63 MeV
$^{14}_{8}\mathrm{O}$	$\rightarrow$	$^{14}_{7}\mathrm{N} + \mathrm{e}^{+} + \nu_{\mathrm{e}}$	+ 5.14 MeV (half-life of 70.641 s)
HCNO-II (partial) cycle			
$^{17}_{9}\mathrm{F} + {}^{1}_{1}\mathrm{H}$	$\rightarrow$	$^{18}_{10}\mathrm{Ne} + \gamma$	+ 3.92 MeV
$^{18}_{10}\mathrm{Ne}$	$\rightarrow$	$^{18}_{9}\mathrm{F} + \mathrm{e}^{+} + \nu_{\mathrm{e}}$	+ 4.44 MeV (half-life of 1.672 s)
$^{18}_{9}\mathrm{F} + {}^{1}_{1}\mathrm{H}$	$\rightarrow$	$^{15}_{8}\mathrm{O} + {}^{4}_{2}\mathrm{He}$	+ 2.88 MeV
HCNO-III (partial) cycle			
$^{18}_{9}\mathrm{F} + {}^{1}_{1}\mathrm{H}$	$\rightarrow$	$^{19}_{10}\mathrm{Ne} + \gamma$	+ 6.41 MeV
$^{19}_{10}\mathrm{Ne}$	$\rightarrow$	$^{19}_{9}\mathrm{F} + \mathrm{e}^{+} + \nu_{\mathrm{e}}$	+ 3.32 MeV (half-life of 17.22 s)
$^{19}_{9}\mathrm{F} + {}^{1}_{1}\mathrm{H}$	$\rightarrow$	$^{16}_{8}\mathrm{O} + {}^{4}_{2}\mathrm{He}$	+ 8.11 MeV

ORCID iD

Pedro Corral https://orcid.org/0000-0003-1908-6986

References

[Abr05] Abraham A 2005 Artificial neural networks *Handbook of Measuring System Design*

[ALS+16] Alvarsson J, Lampa S, Schaal W, Andersson C, Wikberg J E S and Spjuth O 2016 Large-scale ligand-based predictive modelling using support vector machines *J. Cheminf.* **8** 1–9

[And15] Anderson K M 2015 Embrace the challenges: software engineering in a big data world *2015 IEEE/ACM 1st Int. Workshop on big Data Software Engineering* (IEEE) pp 19–25

[ATRPP17] Altae-Tran H, Ramsundar B, Pappu A S and Pande V 2017 Low data drug discovery with one-shot learning *ACS Cent. Sci.* **3** 283–93

[ATS20] Azodi C B, Tang J and Shiu S-H 2020 Opening the black box: interpretable machine learning for geneticists *Trends Genet.* **36** 442–55

[Aur17] Aurélien G 2020 *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems* 2nd edn (O'Reilly Media, Inc.) pp 1135–6

[Bax00] Baxter J 2000 A model of inductive bias learning *J. Artif. Intell. Res.* **12** 149–98

[BBB07] Brameier M, Banzhaf W and Banzhaf W 2007 *Linear Genetic Programming* vol 1 (Springer)

[BD20] Barz B and Denzler J 2020 Deep learning on small datasets without pre-training using cosine loss *Proc. IEEE/CVF Winter Conf. on Applications of Computer Vision* pp 1371–80

[BDV05] Benvenuto O G and Alejandra De Vito M A 2005 The formation of helium white dwarfs in close binary systems-II *Mon. Not. R. Astron. Soc.* **362** 891–905

[BG98] Balakrishnama S and Ganapathiraju A 1998 Linear discriminant analysis-a brief tutorial *Inst. Signal Inf. Process.* **18** 1–8
[bib16] Chen T 2016 XGBoost: a scalable tree boosting system *Proc. 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining* (KDD '16) ed C Guestrin (ACM) pp 785–94
[Bot12] Bottou L 2012 *Stochastic Gradient Descent Tricks* (*Neural Networks: Tricks of the Trade*) 2nd edn (Springer) pp 421–36
[BP16] Borgonovo E and Plischke E 2016 Sensitivity analysis: a review of recent advances *Eur. J. Oper. Res.* **248** 869–87
[Bre96] Breiman L 1996 Bagging predictors *Mach. Learn.* **24** 123–40
[Bre21] Breiman L 2021 Random forests *Mach. Learn.* **45** 5–32
[Bro19] Brownlee J 2019 A gentle introduction to the rectified linear unit (RELU) *Machine Learning Mastery* vol 6
[BSY22] Bahtiyar H, Soydaner D and Yüksel E 2022 Application of multilayer perceptron with data augmentation in nuclear physics *Appl. Soft Comput.* **128** 109470
[BTU04] Blu T, Thevenaz P and Unser M 2004 Linear interpolation revitalized *IEEE Trans. Image Process.* **13** 710–9
[CAH+16] Cyburt R H, Amthor A M, Heger A, Johnson E, Keek L, Meisel Z, Schatz H and Smith K 2016 Dependence of x-ray burst models on nuclear reaction rates *Astrophys. J.* **830** 55
[CCK11] Celebi E M, Celiker F and Kingravi H A 2011 On Euclidean norm approximations *Pattern Recogn.* **44** 278–83
[CFOY16] Cyburt R H, Fields B D, Olive K A and Yeh T-H 2016 Big bang nucleosynthesis: present status *Rev. Mod. Phys.* **88** 015004
[CFQ+17] Cao J, Fang Z, Qu G, Sun H and Zhang D 2017 An accurate traffic classification model based on support vector machines *Int. J. Netw. Manage.* **27** e1962
[CG16] Chen T and Guestrin C 2016 XGBoost: a scalable tree boosting system *Proc. 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, KDD'16 (Association for Computing Machinery New York, NY, USA)* pp 785–94
[CR13] Chauvin Y and Rumelhart D E 2013 *Backpropagation: Theory, Architectures and Applications* (Psychology Press)
[CWC+22] Chen H, Wu L, Chen J, Lu W and Ding J 2022 A comparative study of automated legal text classification using random forests and deep learning *Inf. Process. Manage.* **59** 102798
[DAG98] Duch W, Adamczak R and Grabczewski K 1998 Extraction of logical rules from neural networks *Neural Process. Lett.* **7** 211–9
[Dav75] Davis P J 1975 *Interpolation and Approximation* (Courier Corporation)
[dBCV99] de Bodt E, Cottrell M and Verleysen M 1999 Using the Kohonen algorithm for quick initialization of simple competitive learning algorithm *ESANN* **99** 19–26
[DLL+19] Du S, Lee J, Li H, Wang L and Zhai X 2019 Gradient descent finds global minima of deep neural networks *Int. Conf. on Machine Learning* (PMLR) pp 1675–85
[DS98] Draper N R and Smith H 1998 *Applied Regression Analysis* vol 326 (Wiley)
[DT17] Dheeru D and Taniskidou E K 2017 UCI machine learning repository (available at: https://archive.ics.uci.edu/datasets)
[FA18] Faraway J J and Augustin N H 2018 When small data beats big data *Stat. Probab. Lett.* **136** 142–5
[FC01] Franco L and Cannas S A 2001 Generalization properties of modular networks: implementing the parity function *IEEE Trans. Neural Netw.* **12** 1306–13
[FDCBA14] Fernández-Delgado M, Cernadas E, Barro S and Amorim D 2014 Do we need hundreds of classifiers to solve real world classification problems? *J. Mach. Learn. Res.* **15** 3133–81
[FH89] Fix E and Lawson Hodges J 1989 *Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties* (*Int. Stat. Rev./Revue Int. Stat.*) vol 57 pp 238–47
[FLD+20] Fong S J, Li G, Dey N, González Crespo R, and Herrera-Viedma E 2020 Finding an accurate early forecasting model from small dataset: a case of 2019-ncov novel coronavirus outbreak (arXiv:2003.10776)
[FP+03] Friedman J H and Popescu B E 2003 Importance sampled learning ensembles *J. Mach. Learn. Res.* **94305** 1–32
[Fri02] Friedman J H 2002 Stochastic gradient boosting *Comput. Stat. Data Anal.* **38** 367–78
[FT18] Fratello M and Tagliaferri R 2018 Decision trees and random forests *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics* vol 374
[GBCB16] Goodfellow I, Bengio Y, Courville A and Bengio Y 2016 *Deep Learning* vol 1 (MIT Press)
[GG12] Guest P G and Guest P G 2012 *Numerical Methods of Curve Fitting* (Cambridge University Press)
[GM79] Gasser T and Müller H-G 1979 Kernel estimation of regression functions *Smoothing Techniques for Curve Estimation: Proc. Workshop Held in Heidelberg* (*2 April–4 April 1997*) (Springer) pp 23–68
[GNG+20] Guo F, Ng M, Goubran M, Petersen S E, Piechnik S K, Neubauer S and Wright G 2020 Improving cardiac MRI convolutional neural network segmentation on small training datasets and dataset shift: a continuous kernel cut approach *Med. Image Anal.* **61** 101636
[Gro20] Grover V 2020 Do we need to understand the world to know it? Knowledge in a big data world *J. Glob. Inf. Technol. Manage.* **23** 1–4
[GS78] Gallagher J S and Starrfield S 1978 Theory and observations of classical novae *Annu. Rev. Astron. Astrophys.* **16** 171–214
[Haf22] Hafiz A M 2022 A novel som ensemble classification technique for limited data *Futuristic Trends for Sustainable Development and Sustainable Ecosystems* (IGI Global) pp 76–88
[HGC15] Higuera C, Gardiner K J, Cios K J and Herault Y 2015 Self-organizing feature maps identify proteins critical to learning in a mouse model of Down syndrome *PLoS One* **10** e0129126
[HLH21] Huang F, Li J and Huang H 2021 Super-Adam: faster and universal framework of adaptive gradients *Advances in Neural Information Processing Systems* vol 34 pp 9074–85
[HNP09] Halevy A, Norvig P and Pereira F 2009 The unreasonable effectiveness of data *IEEE Intel. Syst.* **24** 8–12
[Ho95] Ho T K 1995 Random decision forests *Proc. 3rd Int. Conf. on Document Analysis and Recognition* pp 14–16
[HPGG20] Hall L O, Paul R, Goldgof D B, and Goldgof G M 2020 Finding covid-19 from chest x-rays using deep learning on a small dataset (arXiv:2004.02060)
[HWF21] Han S, Williamson B D and Fong Y 2021 Improving random forest predictions in small datasets from two-phase sampling designs *BMC Med. Inf. Decis. Making* **21** 1–9
[IM05] Ingrassia S and Morlini I 2005 Neural network modeling for small datasets *Technometrics* **47** 297–311
[J+12] Jukna S *et al* 2012 *Boolean Function Complexity: Advances and Frontiers* vol 5 (Springer)
[JGS21] Jain V, Goel M and Shah K 2021 Deep learning on small tabular dataset: using transfer learning and image classification *Int. Conf. on Artificial Intelligence and Speech Technology* (Springer) pp 555–68
[JM05] Jacobs T and Maes C 2005 Reversibility and irreversibility within the quantum formalism (arXiv:quant-ph/0508041)
[KB14] Kingma D P and Ba J 2014 Adam: a method for stochastic optimization (arXiv:1412.6980)

[KKZ22] Kokol P, Kokol M and Zagoranski S 2022 Machine learning on small size samples: a synthetic knowledge synthesis *Sci. Prog.* **105** 00368504211029777
[KMNS18] Komiske P T, Metodiev E M, Nachman B and Schwartz M D 2018 Learning to classify from impure samples with high-dimensional data *Phys. Rev.* D **98** 011502
[Koh92] Kohonen T 1992 Learning vector quantisation and the self organising map *Theory and Applications of Neural Networks: Proc. 1st British Neural Network Society Meeting, (London)* (Springer) pp 235–42
[Koh13] Kohonen T 2013 Essentials of the self-organizing map *Neural Netw.* **37** 52–65
[Kou12] Koutsoyiannis D 2012 Clausius–Clapeyron equation and saturation vapour pressure: simple theory reconciled with practice *Eur. J. Phys.* **33** 295
[L+09] Lunardi A *et al* 2009 *Interpolation Theory* vol 9 (Edizioni della Normale Pisa)
[LD15] Liu S and Deng W 2015 Very deep convolutional neural network based image classification using small training sample size *2015 3rd IAPR Asian Conf. on Pattern Recognition (ACPR)* (IEEE) pp 730–4
[Lin91] Lin J 1991 Divergence measures based on the Shannon entropy *IEEE Trans. Inf. Theory* **37** 145–51
[LLZW20] Liang W, Luo S, Zhao G and Wu H 2020 Predicting hard rock pillar stability using GBDT, XGBoost and LightGBM algorithms *Mathematics* **8** 765
[Lom11] Lomont C 2011 Introduction to intel advanced vector extensions *Intel White Paper* **23** 1–21
[LP13] Langdon W B and Poli R 2013 *Foundations of Genetic Programming* (Springer)
[LS13] Lu S and Segall R S 2013 Multi-SOM: an algorithm for high dimensional, small size datasets *Proc. 16th World Multi-Conf. on Systemics, Cybernetics and Informatics: WMSCI 2012* pp 17–20
[LST15] Lake B M, Salakhutdinov R and Tenenbaum J B 2015 Human-level concept learning through probabilistic program induction *Science* **350** 1332–8
[LY96] Lee D and Yannakakis M 1996 Principles and methods of testing finite state machines-a survey *Proc. IEEE* **84** 1090–123
[MAM21] Mansour R F and Al-Marghilnai A 2021 Glaucoma detection using novel perceptron based convolutional multi-layer neural network classification *Multidimens. Syst. Signal Process.* **32** 1217–35
[MBGS14] Mayr A, Binder H, Gefeller O and Schmid M 2014 The evolution of boosting algorithms *Methods Inf. Med.* **53** 419–27
[Mea55] Mealy G H 1955 A method for synthesizing sequential circuits *Bell Syst. Tech. J.* **34** 1045–79
[Men02] Menard S 2002 *Applied Logistic Regression Analysis* (*Number*) vol 106 (Sage)
[Mil07] Million E 2007 The hadamard product *Course Notes* **3** 1–7
[MJZ18] Man W, Ji Y and Zhang Z 2018 Image classification based on improved random forest algorithm *2018 IEEE 3rd Int. Conf. on Cloud Computing and Big Data Analysis (ICCCBDA)* (IEEE) pp 346–50
[MRT19] Ma Y, Rusu F and Torres M 2019 Stochastic gradient descent on modern hardware: multi-core CPU or GPU? Synchronous or asynchronous? *2019 IEEE Int. Parallel and Distributed Processing Symp. (IPDPS)* (IEEE) pp 1063–72
[Nie16] Nielsen D 2016 Tree boosting with XGBoost-why does XGBoost win "every" machine learning competition? *Master's Thesis* NTNU
[NJL21] Nogueira R, Jiang Z and Lin J 2021 Investigating the limitations of transformers with simple arithmetic tasks (arXiv:2102.13019
[Nob06] Noble W S 2006 What is a support vector machine? *Nat. Biotechnol.* **24** 1565–7
[OD02] Oltean M 2021 Multi expression programming - an in-depth description *Research Square Platform LLC* (Accessed September 2021) (https://doi.org/10.21203/rs.3.rs-898407/v1)
[O'H78] O'Hagan A 1978 Curve fitting and optimal design for prediction *J. R. Stat. Soc.* B **40** 1–24
[OKK03] Oja M, Kaski S and Kohonen T 2003 Bibliography of self-organizing map (SOM) papers: 1998–2001 addendum *Neural Comput. Surv.* **3** 1–156
[Ost12] Ostertagová E 2012 Modelling using polynomial regression *Proc. Eng.* **48** 500–6
[OW19] Ogunleye A and Wang Q-G 2019 XGBoost model for chronic kidney disease diagnosis *IEEE/ACM Trans. Comput. Biol. Bioinform.* **17** 2131–40
[OWB18] Olson M, Wyner A and Berk R 2018 Modern neural networks generalize on small data sets *Advances in Neural Information Processing Systems* vol 31
[Pas15] Pasini A 2015 Artificial neural networks for small dataset analysis *J. Thoracic Dis.* **7** 953
[Pat09] Patterson D 2009 Molecular genetic analysis of Down syndrome *Hum. Genet.* **126** 195–214
[R+01] Rish I *et al* 2001 An empirical study of the naive Bayes classifier *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence* vol 3 pp 41–46
[Ram99] Ramponi G 1999 Warped distance for space-variant linear image interpolation *IEEE Trans. Image Process.* **8** 629–39
[Ros58] Rosenblatt F 1958 The perceptron: a probabilistic model for information storage and organization in the brain *Psychol. Rev.* **65** 386
[RRK21] Riekert M, Riekert M and Klein A 2021 Simple baseline machine learning text classifiers for small datasets *SN Comput. Sci.* **2** 178
[Rud11] Ruda F 2011 *Hegel's Rabble: an Investigation Into Hegel's Philosophy of Right* (Bloomsbury Publishing)
[Rud16a] Ruder S 2016 An overview of gradient descent optimization algorithms (arXiv:1609.04747)
[Rud16b] Ruder S 2016 An overview of gradient descent optimization algorithms (arXiv:1609.04747)
[Rud19] Rudin C 2019 Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead *Nat. Mach. Intell.* **1** 206–15
[RZIX20] Razzak I, Zafar K, Imran M and Xu. G 2020 Randomized nonlinear one-class support vector machines with bounded loss function to detect of outliers for large scale IoT data *Future Gener. Comput. Syst.* **112** 715–23
[SCF+20] Spiga O *et al* 2020 Machine learning application for development of a data-driven predictive model able to investigate quality of life scores in a rare disease *Orphanet J. Rare Dis.* **15** 1–10
[Sch01] Schoenberg R 2001 Optimization with the quasi-Newton method *Aptech Systems Maple Valley WA* 1–9
[SD98] Skurichina M and Duin R P W 1998 Bagging for linear classifiers *Pattern Recognit.* **31** 909–30
[Sha83] Shapiro S 1983 Mathematics and reality *Phil. Sci.* **50** 523–48
[SIK86] Sakamoto Y, Ishiguro M and Kitagawa G 1986 *Akaike Information Criterion Statistics* D Reidel 81 26853
[Sim12] Simonoff J S 2012 *Smoothing Methods in Statistics* (Springer)
[SK19] Shorten C and Khoshgoftaar T M 2019 A survey on image data augmentation for deep learning *J. Big Data* **6** 1–48
[SLD+19] Shaikhina T, Lowe D, Daga S, Briggs D, Higgins R and Khovanova N 2019 Decision tree and random forest models for outcome prediction in antibody incompatible kidney transplantation *Biomed. Signal Process. Control* **52** 456–62

[SY15] Song Y-Y and Ying L U 2015 Decision tree methods: applications for classification and prediction *Shanghai Arch. Psychiatry* **27** 130
[TASS07] Tam D, Azimi R, Soares L and Stumm M 2007 Managing shared L2 caches on multicore systems in software *Workshop on the Interaction Between Operating Systems and Computer Architecture* (Toronto) pp 26–33
[Teg08] Tegmark M 2008 The mathematical universe *Found. Phys.* **38** 101–50
[TEHD20] Tits N, El Haddad K and Dutoit T 2020 Exploring transfer learning for low resource emotional TTS *Intelligent Systems and Applications: Proc. 2019 Intelligent Systems Conf. (IntelliSys)* vol 1 (Springer) pp 52–60
[Tha16] Tharwat A 2016 Linear vs. quadratic discriminant analysis classifier: a tutorial *Int. J. Appl. Pattern Recogn.* **3** 145–80
[TWR+22] Tong J, Wang Z, Rui X and Loddo A 2022 A multimodel-based deep learning framework for short text multiclass classification with the imbalanced and extremely small data set *Comput. Intell. Neurosci.* **2022** 1–12
[VBL+16] Vinyals O, Blundell C, Lillicrap T and Wierstra D 2016 Matching networks for one shot learning *Advances in Neural Information Processing Systems* vol 29
[VdMH08] Van der Maaten L and Hinton G 2008 Visualizing data using t-SNE *J. Mach. Learn. Res.* **9** 2579–605
[Vol99] Vollmer H 1999 *Introduction to Circuit Complexity: a Uniform Approach* (Springer)
[WDGK23] Wang H, Duentsch I, Guo G and Ali Khan S A 2023 Special issue on small data analytics *Int. J. Mach. Learn. Cybern.* **14** 1–2
[Weg18] Weglarczyk S 2018 Kernel density estimation and its application *ITM Web Conf.* **23** 00037
[WGU+10] Wiescher M, Görres J, Uberseder E, Imbriani G and Pignatari M 2010 The cold and hot CNO cycles *Ann. Rev. Nucl. Part. Sci.* **60** 381–404
[Wie18] Wiescher M 2018 The history and impact of the CNO cycles in nuclear astrophysics *Phys. Perspect.* **20** 124–58
[Wig93] Wigner E P 1993 *Über die Operation der Zeitumkehr in der Quantenmechanik* (Springer)
[Wri06] Wright S J 2006 Fundamentals of unconstrained optimization *Numerical optimization* 2nd edn (Springer) p 17
[WSC19] Wang Y, Sun Z and Chen Z 2019 Energy management strategy for battery/supercapacitor/fuel cell hybrid source vehicles based on finite state machine *Appl. Energy* **254** 113707
[WSZ+20] Weng B, Song Z, Zhu R, Yan Q, Sun Q, Grice C G, Yan Y and Yin W-J 2020 Simple descriptor derived from symbolic regression accelerating the discovery of new perovskite catalysts *Nat. Commun.* **11** 3513
[WYKN20] Wang Y, Yao Q, Kwok J T and Ni L M 2020 Generalizing from a few examples: a survey on few-shot learning *ACM Comput. Surv.* **53** 1–34
[WZS+13] Wu Y, Zhou Y, Saveriades G, Agaian S, Noonan J P and Natarajan P 2013 Local Shannon entropy measure with statistical tests for image randomness *Inf. Sci.* **222** 323–42
[XJLL23] Xu P, Ji X, Li M and Lu W 2023 Small data machine learning in materials science *npj Comput. Mater.* **9** 42
[Yeg09] Yegnanarayana B 2009 *Artificial Neural Networks* (PHI Learning Pvt. Ltd)
[Yin08] Yin H 2008 Learning nonlinear principal manifolds by self-organising maps *Principal Manifolds for Data Visualization and Dimension Reduction* (Springer) pp 68–95
[YNDT18] Yamashita R, Nishio M, Kinh Gian Do R and Togashi K 2018 Convolutional neural networks: an overview and application in radiology *Insights into Imaging* **9** 611–29
[ZFO17] Zhong J, Feng L and Ong Y-S 2017 Gene expression programming: a survey *IEEE Comput. Intell. Mag.* **12** 54–72
[ZJQ+22] Zou M, Jiang W-G, Qin Q-H, Liu Y-C and Li M-L 2022 Optimized XGBoost model with small dataset for predicting relative density of Ti-6AI-4V parts manufactured by selective laser melting *Materials* **15** 5298
[ZRK20] Zhang Y, Ren H and Khailany B 2020 Grannite: graph neural network inference for transferable power estimation *2020 57th ACM/IEEE Design Automation Conf. (DAC)* (IEEE) pp 1–6
[ZSC+16] Zhang Z, Song Y, Cui H, Wu J, Schwartz F and Qi H 2016 Topological analysis and Gaussian decision tree: effective representation and classification of biosignals of small sample size *IEEE Trans. Biomed. Eng.* **64** 2288–99
[ZZ19] Zhang D and Zhang D 2019 Wavelet transform *Fundamentals of image data mining: Analysis, Features, Classification and Retrieval* pp 35–44
[ZZC+21] Zhang Y, Zhu R, Chen Z, Gao J and Xia D 2021 Evaluating and selecting features via information theoretic lower bounds of feature inner correlations for high-dimensional data *Eur. J. Oper. Res.* **290** 235–47
[ZZK+20] Zhong Z, Zheng L, Kang G, Li S and Yang Y 2020 Random erasing data augmentation *Proc. AAAI Conf. on Artificial Intelligence* vol 34 pp 13001–8