Information Sciences 686 (2025) 121363

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Training feedforward neural networks with Bayesian hyper-heuristics ☆

A.N. Schreuder [a,*], A.S. Bosman [a], A.P. Engelbrecht [b,c], C.W. Cleghorn [d]

[a] University of Pretoria, Pretoria, Gauteng, South Africa
[b] Stellenbosch University, Stellenbosch, Western Cape, South Africa
[c] Center for Applied Mathematics and Bioinformatics, Gulf University for Science and Technology, Mishref, Kuwait
[d] University of the Witwatersrand, Johannesburg, Gauteng, South Africa

ARTICLE INFO

Keywords:
Hyper-heuristics
Meta-learning
Feedforward neural networks
Supervised learning
Bayesian statistics

ABSTRACT

The process of training *feedforward neural networks* (FFNNs) can benefit from an automated process where the best heuristic to train the network is sought out automatically by means of a high-level probabilistic-based heuristic. This research introduces a novel population-based *Bayesian hyper-heuristic* (BHH) that is used to train *feedforward neural networks* (FFNNs). The performance of the BHH is compared to that of ten popular low-level heuristics, each with different search behaviours. The chosen heuristic pool consists of classic gradient-based heuristics as well as *meta-heuristics* (MHs). The empirical process is executed on fourteen datasets consisting of classification and regression problems with varying characteristics. The BHH is shown to be able to train FFNNs well and provide an automated method for finding the best heuristic to train the FFNNs at various stages of the training process.

1. Introduction

A popular field of focus for studying *artificial neural networks* (ANNs) is the process by which these models are trained. ANNs are trained by optimisation algorithms known as heuristics. Many different heuristics have been developed and used to train ANNs [1]. Each heuristic has different search behaviours, characteristics, strengths and weaknesses. It is necessary to find the best heuristic to train ANNs in order to yield optimal results. This process is often non-trivial and time-consuming. Selection of the best heuristic to train ANNs is often problem specific [2].

A recent suggestion related to the field of *meta-learning* is to dynamically select and/or adjust the heuristic used throughout the training process. This approach focuses on the hybridisation of learning paradigms. One such form of hybridisation of learning paradigms is that of hybridisation of different *heuristics* as they are applied to some optimisation problem [3]. These methods are referred to as *hyper-heuristics* (HHs) and focus on finding the best heuristic in *heuristic space* to solve a specific problem.

In the general context of optimisation, many different types of HHs have been implemented and applied to many different problems [3]. However, research on the application of HHs in the context of ANN training is scarce. Nel [4] provides some of the first research in this field, applying a HH to *feedforward neural network* (FFNN) training.

☆ It is recommended that the article be viewed/printed in colour.

* Corresponding author.
E-mail addresses: an.schreuder@up.ac.za (A.N. Schreuder), annar@cs.up.ac.za (A.S. Bosman), engel@sun.ac.za (A.P. Engelbrecht), christopher.cleghorn@wits.ac.za (C.W. Cleghorn).

https://doi.org/10.1016/j.ins.2024.121363
Received 9 February 2023; Received in revised form 15 August 2024; Accepted 15 August 2024

Available online 21 August 2024

This research takes a particular interest in developing a population-based, selection HH that makes use of probability theory and Bayesian statistical concepts to guide the heuristic selection process. This paper presents a novel *Bayesian hyper-heuristic* (BHH), a new high-level heuristic that utilises a statistical approach, referred to as *Bayesian analysis*, which combines prior information with new evidence to the parameters of a heuristic selection probability distribution.

The general concept of the BHH is summarised as follows: the BHH implements a high-level heuristic selection mechanism that learns to select the best heuristic from a pool of low-level heuristics. These low-level heuristics are applied to a population of entities, each implementing a candidate solution to a FFNN. The intent of the BHH is to optimise both the underlying FFNN and the FFNN training process. The BHH does so by learning the probability that a given heuristic will perform well at a given stage in the FFNN training process. These probabilities are then used as heuristic selection probabilities in the next step of the training process.

The update of selection probabilities is guided by Bayesian analysis, where prior probabilities are updated with new evidence to form posterior probabilities. The performance of each heuristic-entity combination is logged, which captures the efficacy of each heuristic applied to their respective entity, at different stages of the training process. The logged performance data acts as new evidence for the Bayesian analysis process. The BHH also incorporates a novel proxied heuristic update step, where various components of the update step for low-level heuristics are sourced from other heuristics in the pool of heuristics, when the direct application of a heuristic is not feasible/possible.

The selection mechanism implemented by the BHH is different from the *multialgorithm, genetically adaptive multiobjective* (AMALGAM) and *bi-objective hyperheuristic training algorithm* (BOHTA) methods used by Nel [4], as well as the *hyper-heuristic Bayesian optimisation algorithm* (HHBOA) proposed by Oliva and Martins [5]. The key differences include that the BHH does not follow an evolutionary approach to the selected low-level heuristics. As such the population does not generate offspring, but rather reuses entities in the population. Furthermore, the BHH implements a discrete credit assignment mechanism, not making use of pareto fronts as in the AMALGAM and BOHTA methods.

Although this research takes a particular interest in training FFNNs, the BHH is not limited to these types of models and can generally be applied to any ANN as long as the ANN can be trained by all of the low-level heuristics. Experimentation with other ANN architectures besides FFNNs is left for future research.

The remainder of this article is structured as follows: Section 2 provides background information on ANNs. Section 3 provides details on various types of heuristics that have been used to train FFNNs. Section 4 presents background information on HHs and meta-learning. Section 5 presents background information on probability theory. Section 6 presents the developed BHH. Section 7 presents a detailed description of the empirical process and the setup of each experiment. Section 8 provides and discusses the results of the empirical study. Section 9 summarises the research that is done along with a brief overview of the findings.

2. Artificial neural networks

This research focuses on a particular type of ANN, referred to as *feedforward neural networks* (FFNNs). FFNNs were the first and simplest type of ANNs developed [6] and implement an architecture consisting of input, hidden and output layers by arranging them in sequential order. Furthermore, FFNNs implement fully connected topologies, where each *artificial neuron* (AN) in one layer is connected to all the ANs in the next, without any cycles. In FFNNs, information moves forward, in one direction, from the input nodes, through the hidden nodes and finally to the output nodes.

Training is the process whereby the weights of the FFNN are systematically changed with the aim of improving the *performance* of the FFNN. Finding the optimal weights that produce the best performance on a given task is an optimisation problem. The optimisation algorithm used to find the optimal weights is referred to as a *heuristic*. Heuristics search for possible solutions in the solution-space and make use of information from the search space to guide to process.

During the training process, the FFNN is exposed to data while trying to produce some target outcome. The degree to which the produced outcome differs from the target outcome is referred to as *loss*. Since training of FFNNs is an optimisation problem, the goal of the training process is to minimise the loss. The loss is calculated using an error function.

3. Heuristics

A heuristic refers to an algorithmic search technique that serves as a guide to a search process where good solutions to an optimisation problem are being sought out. Many different techniques have been used to train FFNNs [7]. At the time of writing, the majority of work that is published on the training of FFNNs involves the use of gradient-based techniques [4].

Gradient-based heuristics are optimisation techniques that make use of derivatives obtained from evaluating the ANN error function. In the context of supervised learning, loss functions produce a scalar value that represents the error between the output of the ANN and the desired output. When using *gradient descent* (GD) to train ANNs, the gradients of the loss function is used to adjust the weights of the ANN in order to minimise the error [8].

There are many variants of gradient-based heuristics. However, they all fundamentally apply the same generic GD framework that propagates the error signal backwards through the ANN. This algorithm is known as *backpropagation* (BP).

The simplest type of GD algorithm is referred to as *stochastic gradient descent* (SGD), which implements a gradient-based weight update step for each training pattern. In the context of this research, the implementation of SGD refers to the mini-batch training implementation of GD, where a small batch of training patterns are fed to the FFNN at once and the error function is aggregated across all training patterns.

Alternative variants have been proposed that lead to better control over the convergence characteristics of SGD. This research focuses on a number of these variants that include *momentum* (Momentum) [9], *Nesterov accelerated gradients* (NAG) [10], *adaptive gradients* (Adagrad) [11], Adadelta [12], *root mean squared error propagation* (RMSProp) [13] and *adaptive moment estimation* (Adam) [7].

Gradient-based heuristics are sensitive to the problem that they are applied to, with hyper-parameter selection often dominating the research focus [14]. Blum and Roli [15] mention that since the 1980s, a new kind of approximate algorithm has emerged which tries to combine basic heuristic methods in higher level frameworks aimed at efficiently and effectively exploring a search space. These methods are referred to as MHs.

The biggest difference between MHs and gradient-based heuristics is that MHs make use of meta-information obtained as a result of evaluating the FFNN during training and is not limited to information about the search space [15]. This also means that MHs do not necessarily require the error function to be differentiable. Blum and Roli [15] provide advantages of MHs that include the following: they are easy to implement, they are problem independent and do not require problem-specific knowledge, and they are generally designed to find global optima, while gradient-based approaches can get stuck in local optima more often. Similar to gradient-based heuristics, a number of different meta-heuristics have been used to successfully train FFNNs [1,16]. This research takes a particular interest in population-based MHs that have been used to train FFNNs. These include *particle swarm optimisation* (PSO) [17], *differential evolution* (DE) [18], and *genetic algorithms* (GAs) [19].

4. Hyper-heuristics

Burke et al. [20] define HHs as search methods or learning mechanism for selecting or generating heuristics to solve computational search problems. Burke et al. [21] mention that a HH is a high-level heuristic approach that, given a particular problem instance and a number of low-level heuristics, can select and apply an appropriate low-level heuristic at each decision point. HHs implement a form of *meta-learning* that is concerned with the selection of the best heuristic from a pool of heuristics to solve a given problem. It can be said that HHs are concerned with finding the best heuristic in *heuristic space*, while the underlying low-level heuristics find solutions in the feasible *search/solution space*.

Burke et al. [20] propose a classification scheme used to classify HHs. According to the proposed classification scheme, HHs are classified in two categories. These include the *source of feedback* used during learning and the nature of the *heuristic search space*. For the category that involves the source of feedback, HHs can be classified as either *no learning*, *online learning* or *offline learning*. For the category that involves the nature of the *heuristic search space*, HHs can be classified as either *heuristic selection* or *heuristic generation*. Further distinction is made between *construction* of heuristics and *perturbation* of heuristics.

In the general context of optimisation, many different types of HHs have been implemented and applied to many different problems. Some notable examples include [20,22,23]. Research on the application of HHs in the context of FFNN training is still scarce. Nel [4] provides some of the first research in this field, applying BOHTA, a novel adaptation of an evolutionary-based HH, known as the AMALGAM HH [24], to FFNN training. Furthermore, Oliva and Martins [5] provide HHBOA, the first use of Bayesian optimisation in a HH context. The method proposed by Oliva and Martins [5] uses a Bayesian selection operator to evolve combinations of low-level heuristics while looking for good problem solutions to a benchmark of optimisation functions, but does not apply a HH to the training of FFNNs.

This research takes a particular interest in a population-based, selection approach for HHs, with the particular intent of training FFNNs. In the context of population-based HHs, an entity pool exists that represents a pool of candidate solutions to the given problem. Each entity in the entity pool is assigned its own low-level heuristic from the heuristic pool. The selection of the best heuristic to apply to a candidate solution is based on the performance of the heuristic relative to that particular candidate solution at a particular point in the search process. Selection methods often make use of probabilistic approaches.

5. Probability

Bayesian statistics describe the probability of an event in terms of some belief, based on previous knowledge of the event and the conditions under which the event happened [25]. Bayes' theorem expresses how a degree of belief, expressed as a probability, should rationally change to account for the availability of related evidence.

One of the many applications of Bayes' theorem is to do statistical inference. Like FFNNs, Bayesian models need to be *trained*, a process known as *Bayesian analysis*. Bayesian analysis is the process by which prior beliefs are updated as a result of observing new data/evidence.

Bayesian analysis utilises the concept of conjugate priors. Wackerly et al. [26] state that conjugate priors are prior probability distributions that result in posterior distributions that are of the same functional form as the prior, but with different parameter values. The conjugate prior to a Bernoulli probability distribution is the Beta probability distribution and the conjugate prior to a categorical and multinomial probability distribution is the Dirichlet probability distribution [26].

6. Bayesian hyper-heuristics

This section presents the novel BHH and is structured as follows: Section 6.1 presents an overview of the BHH, Sections 6.2 - 6.7 provide details on the most important components of the BHH, Section 6.8 describes the optimisation step implemented by the BHH and Section 6.9 discusses the hyper-parameters used by the BHH.

6.1. Overview

According to the classification scheme for HHs by Burke et al. [20], the BHH is a population-based, meta-hyper-heuristic that utilises selection and perturbation of low-level heuristics in an online learning fashion.

The BHH implements a high-level heuristic selection mechanism that learns to select the best heuristic from a pool of low-level heuristics. These low-level heuristics are applied to a population of entities, each implementing a candidate solution to a FFNN. The intent of the BHH is to optimise both the underlying FFNN and the FFNN training process. The BHH does so by learning the probability that a given heuristic will perform well at a given stage in the FFNN training process. These probabilities are then used as heuristic selection probabilities in the next step of the training process.

The update of selection probabilities is guided by Bayesian analysis, where prior probabilities are updated with new evidence to form posterior probabilities. The performance of each heuristic-entity combination is logged, which captures the efficacy of each heuristic applied to their respective entity, at different stages of the training process. The logged performance data acts as new evidence for the Bayesian analysis process. The BHH also incorporates a novel proxied heuristic update step, where various components of the update step for low-level heuristics are sourced from other heuristics in the pool of heuristics, when the direct application of a heuristic is not feasible/possible.

Fig. 1 provides an illustration of the high-level architecture of the BHH. Algorithm 1 provides the high level pseudo-code implementation of the BHH. Discussions follow on the most important components of the BHH.

Algorithm 1 The pseudo-code for the implementation of the *Bayesian hyper-heuristic* (BHH).

```
step ← 0
select initial heuristics
initialise population and entities
evaluate entities' initial position
update population state
while stopping condition not met do
    for all entities in entity pool do
        if selected heuristic is gradient-based then
            get gradients
        end if
        apply low-level heuristic and proxy operations
        update population state
        log performance metrics to performance log
        if step < burn-in window size then
            select heuristic
        else
            if step % reanalysis interval = 0 then
                apply Bayesian analysis
            end if
            if step % reselection interval = 0 then
                select heuristic
            end if
            if step < replay window size then
                prune performance log
            end if
        end if
    end for
    step ← step + 1
end while
```

6.2. Heuristic pool

Generally speaking, the heuristic pool is a collection of low-level heuristics under consideration by the BHH. The heuristic pool contains the set of low-level heuristics that, together with their performance information, make up the heuristic space. Importantly, the heuristic pool must consist of a diverse set of low-level heuristics with varying capabilities. This research takes an interest in including both gradient-based heuristics as well as MHs in the heuristic pool. This approach is referred to as a *multi-method* approach.

6.3. Proxies

Heuristics often maintain a set of parameters that are used to control the behaviour of the heuristic. These parameters are refered to as heuristic *state*. The concept of proxies arises from the sparsity of state as maintained by different heuristics. Since heuristics maintain (possibly) different states, there is an uncertainty of state transition when switching between heuristics. A solution to state indifference is to *proxy* heuristic state update operations. State is then maintained in two parts: primary and proxied state. Primary state refers to the state that is originally maintained by a heuristic. Proxied state refers to the state that is not directly maintained by the heuristic, but can be updated by outsourcing the required state update operation to another heuristic. The BHH thus incorporates a mapping of proxied state update operations as given in the example in Table 1.

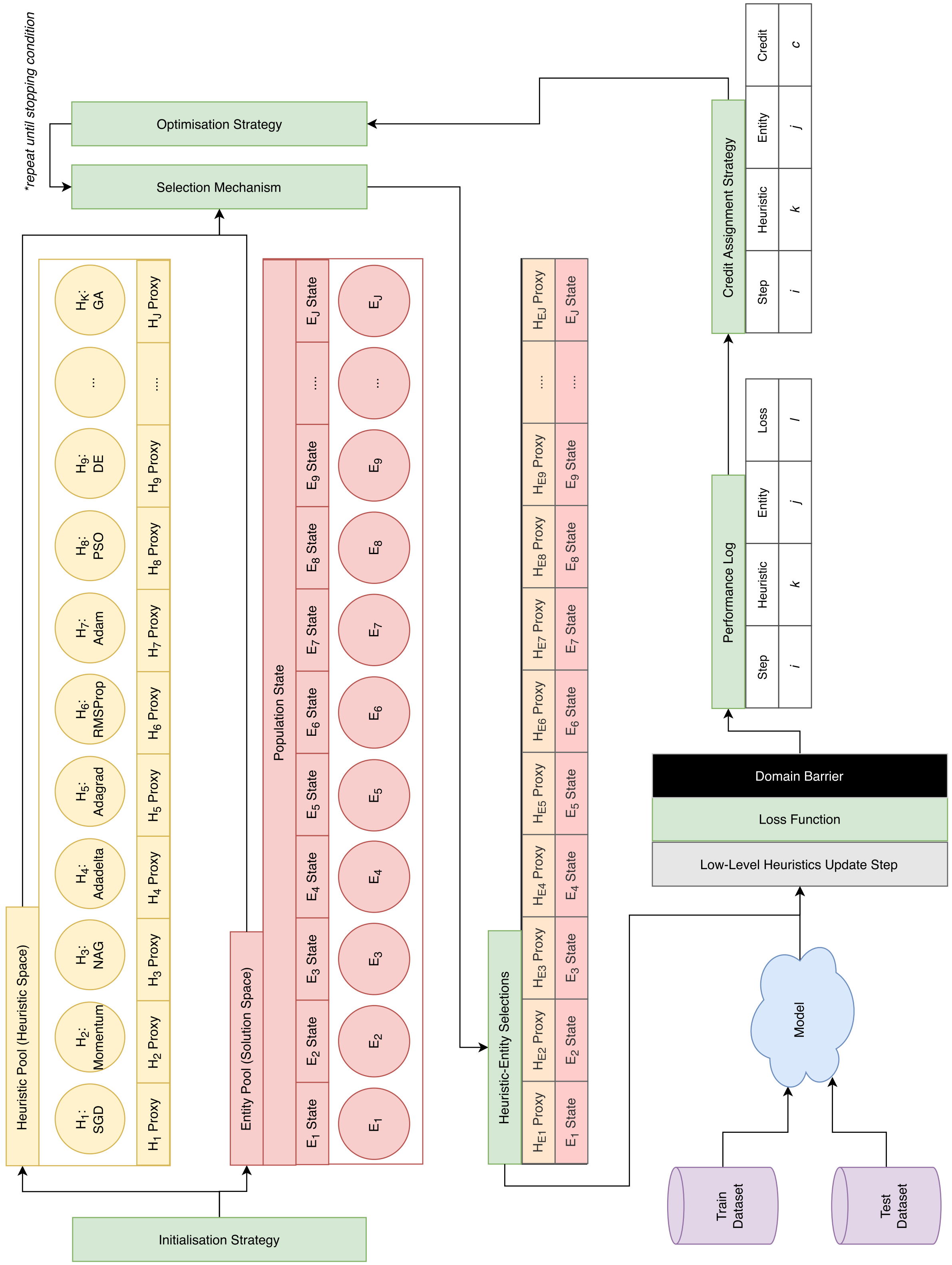

Fig. 1. An illustration of the architecture and high level components of the *Bayesian hyper-heuristic* (BHH).

Table 1
An example of a mapping of proxied state update operation maintained by the BHH.

		State Parameter		
		1	2	3
Heuristic	A	n/a	B	n/a
	B	n/a	n/a	A
	C	n/a	B	A

From the example given in Table 1, when heuristic A is selected, it will outsource state update operations from heuristic B for state parameter 2. Heuristic B will outsource from heuristic A for state parameter 3. Finally, heuristic C will outsource from heuristic A and B for state parameters 2 and 3 respectively. In this way, all heuristics maintain all the state parameters.

6.4. Entity pool

The entity pool refers to a collection or *population* of *entities* that each represent a candidate solution to the underlying FFNN being trained. The BHH selects from the heuristic pool a low-level heuristic to be applied to an individual entity. The outcome of this selection process is a mapping table that tracks which heuristic has been selected for which entity. These heuristic-entity combinations are applied to the underlying FFNN. The BHH tracks the performance of each of these combinations throughout the training process in a performance log.

Entities represent candidate solutions to the model's trainable parameters (weights) and other heuristic-specific state parameters. These state parameters are referred to as *local* state. Entities are treated as physical *particles* in a hyper-dimensional search environment. Entities model concepts from physics. For example, the candidate solution is represented as the entity's position. The velocity and acceleration is then analogous to the gradient and momentum of the entity respectfully [27]. Examples of entity state parameters, as derived from various low-level heuristics, include entity position, velocity, gradient, position delta, first and second moments of the gradient, the loss, personal best positions, losses, and so on. The entity state parameters are updated by the associated heuristic.

The population state refers to a collection of parameters that are shared between the entities in the population. Population state is also referred to as *global* state and represents the population's memory. The population state generally contains state parameters that are of importance to multiple heuristics, and usually tracks the state of the population and not the state of individual heuristics. Some examples of population state that can arise from different heuristics include the population of entities themselves, the global best entity found so far, the overall best loss achieved thus far, and so on.

6.5. Performance log

Heuristic selection probability is calculated based on heuristic-entity performance over time. Evidence of heuristic-entity performance is thus required for the BHH to learn. Historical heuristic-entity performance outcomes are stored in a performance log. The performance log tracks information such as the current step, selected heuristic, associated entity, the loss achieved and so on. Since the performance log can become very big, only a sliding window of the performance history is maintained at each step in the learning process. The sliding window is also referred to as a *replay* window/buffer.

6.6. Credit assignment strategy

The credit assignment strategy is a mechanism that assigns a discrete credit indicator to heuristics that perform well, based on their performance metrics such as loss. The credit assignment strategy implements a component of the "move acceptance" process as proposed by Özcan et al. [28] and addresses the credit assignment problem as discussed by Burke et al. [20]. A good credit assignment strategy will correctly allocate credit to the appropriate heuristic-entity combination. This research implements the following credit assignment strategies to choose from: *ibest* (iteration best), *pbest* (personal best), *gbest* (global best), *rbest* (replay window best), and *symmetric*, where credit is assigned to all entity-heuristic combinations, regardless of their performance.

6.7. Selection mechanism

The BHH implements a probabilistic predictive model based on the fundamentals of the Naïve Bayes algorithm. The BHH thus distinguishes between the following events: $\boldsymbol{H}$, the event of observing *heuristics*, $\boldsymbol{E}$, the event of observing *entities*, and $\boldsymbol{C}$, the event of observing *credit assignments* that indicate that the credit assignment *performance criteria* are met. By Bayes' theorem, the selection mechanism implemented by the BHH is given as

$$P(\boldsymbol{H}|\boldsymbol{E},\boldsymbol{C};\boldsymbol{\theta},\boldsymbol{\phi},\boldsymbol{\psi}) \propto P(\boldsymbol{E}|\boldsymbol{H};\boldsymbol{\phi})P(\boldsymbol{C}|\boldsymbol{H};\boldsymbol{\psi})P(\boldsymbol{H}|\boldsymbol{\theta}) \tag{1}$$

The predictive model thus models the *proportional* probability of the event (selection of) heuristic $\boldsymbol{H}$, given allocation to entity $\boldsymbol{E}$ and credit requirement $\boldsymbol{C}$, parameterised by sampled $\boldsymbol{\theta} \sim Dir(\boldsymbol{\alpha}; K)$, $\boldsymbol{\phi} \sim Dir(\boldsymbol{\beta}; K)^J$ and $\boldsymbol{\psi} \sim Beta(\gamma_1, \gamma_0)$. In the aforementioned, K is the heuristic pool size and J is the entity pool size. The parameters $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, γ_1 and γ_0 are referred to as concentration parameters. The

Table 2
The BHH baseline configuration as it is used in the empirical study.

heuristic pool	population	burn in	credit	reselection	replay	reanalysis	normalise	discounted rewards
all	5	0	ibest	10	10	10	false	false

concentration parameters are used to parameterise the prior probability distributions. Appendix A provides mathematical derivations of the predictive model.

6.8. Optimisation step

The intent of the BHH is to gather evidence that can be used to update prior beliefs about which heuristics perform well during training. These beliefs are represented by the concentration parameters $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, γ_1 and γ_0. A change in prior beliefs is represented by a change in these concentration parameters. Specifically, it can be said that the optimisation process implemented by the BHH updates *pseudo counts* of events that are observed in the performance logs. These pseudo counts track the occurrence of a heuristic, an entity, and resulting performance of these two elements. Through the credit assignment strategy, these pseudo counts are biased towards entity-heuristic combinations that meet performance requirements and yield credit allocations.

Generally, there are two different techniques that are used to train Naïve Bayes classifiers. The frequentist approach implements *maximum likelihood estimation* (MLE) and the Bayesian approach implements *maximum a posteriori estimation* (MAP).

6.8.1. Maximum a posteriori estimation

MAP is an approach to optimise the values for $\hat{\theta}_k$, $\hat{\phi}_{j,k}$ and $\hat{\psi}_k$ by optimising the parameters of their probability distributions. This process is referred to as *Bayesian analysis*. Bayesian analysis makes use of the *posterior* probability distribution. The concentration update operations yielded by MAP, are given as follows:

$$\alpha_k(t+1) = N_k + \alpha_k(t) \tag{2}$$

$$\beta_{j,k}(t+1) = N_{j,k} + \beta_{j,k}(t) \tag{3}$$

$$\gamma_{1,k}(t+1) = N_{1,k} + \gamma_{1,k}(t) \tag{4}$$

$$\gamma_{2,k}(t+1) = N_{0,k} + \gamma_{2,k}(t) \tag{5}$$

where N_k is a summary variable denoting the count of occurrences of heuristic k, N_j is a summary variable denoting the count of occurrences of entity j, $N_{j,k}$ is a summary variable denoting the count of occurrences of heuristic k for entity j, $N_{1,k}$ and $N_{0,k}$ are summary variables denoting the count of occurrences where heuristic k meets performance requirements and where heuristic k does not not meet performance requirements.

It can be said that the BHH implements a Gaussian process [29]. Since the reselection of heuristics happens at regular intervals, the outcome of a selection in one iteration may influence the outcome of another in the next iteration, making the implementation of the BHH a *hidden Markov model* (HMM) [30].

6.9. Hyper-parameters

The following hyper-parameters are implemented by the BHH: the *heuristic pool* configures the type of heuristics included in the heuristic pool, the *population size* specifies the number of entities in the entity pool, the *credit assignment strategy* specifies which credit assignment strategy to use, the *reselection interval* determines the frequency of heuristic reselection, the *replay window size* determines the maximum size of the performance log, the *reanalysis interval* determines the frequency at which Bayesian analysis is applied, the *burn in window size* determines the size of an initial window where experience is simply gathered without reanalysis, and finally, the *discounted rewards* and *normalisation* flags toggle scaling modifiers on values assigned by the credit assignment strategies, backwards in the performance log.

7. Methodology

This section provides the details of the implementation of the empirical process. At a high level the experimental procedure consist of a comparison between the BHH and standalone low-level heuristics. A number of datasets, models and heuristics are specified. Throughout the empirical process, a BHH baseline configuration is used.

7.1. BHH baseline

The BHH baseline is a name given to a specific configuration of the BHH which has been found to provide a reasonable baseline performance. The baseline configuration is used as the cornerstone configuration from which all other heuristics and their configurations are evaluated. The BHH baseline configuration is given in Table 2. In Table 2, the heuristic pool configuration, *all*, refers to a configuration where the heuristic pool contains all the low-level heuristics, including all gradient-based heuristics and MHs.

Table 3
Low-level heuristics and their hyper-parameter configurations.

heuristic	configuration	value	citation
sgd	learning rate	0.1 (0.01)	[10]
momentum	learning rate	0.1 (0.01)	[10]
	momentum	0.9	
nag	learning rate	0.1 (0.01)	[10]
	momentum	0.9	
adagrad	learning rate	0.1 (0.01)	[11]
	epsilon	1E-07	
rmsprop	learning rate	0.1 (0.01)	[13]
	rho	0.95	
	epsilon	1E-07	
adadelta	learning rate	1.0 (0.95)	[12]
	rho	0.95	
	epsilon	1E-07	
adam	learning rate	0.1 (0.01)	[7]
	beta1	0.9	
	beta2	0.999	
	epsilon	1E-07	
pso	population size	10	[32]
	learning rate	1.0 (0.9)	
	inertia weight (w)	0.729844	
	cognitive control (c1)	1.49618	
	social control (c2)	1.49618	
	velocity clip min	-1.0	
	velocity clip max	1.0	
de	population size	10	[33]
	selection strategy	best	
	xo strategy	exp	
	recombination probability	0.9 (0.1)	
	beta	2.0 (0.1)	
ga	population size	10	[34]
	selection strategy	rand	
	xo strategy	bin	
	mutation rate	0.2 (0.05)	

A population size of 5 is chosen as it empirically and consistently provides good results. The population size has a lower bound of 4, which is the highest, minimum number of entities required by any low-level heuristic used in this research. Furthermore, ablation studies [31] have shown that a smaller population size is generally better, since the Bayesian analysis process implemented by the BHH has less, but more concentrated performance information from which it has to learn.

7.2. BHH vs. low-level heuristics

For the standalone heuristics experimental group, a number of low-level heuristics are used along with their specified hyper-parameters. Each of these standalone low-level heuristics is compared to that of the BHH baseline configuration, across all datasets. The intent of the standalone heuristics experimental group is to determine if the BHH baseline configuration can generalise to multiple problems in comparison to individual low-level heuristics.

Additional to the BHH baseline configuration, two more BHH configurations are included. These include BHH configurations where the heuristic pool only makes use of gradient-based heuristics, and a configuration where the heuristic pool only makes use of MHs. The intent behind the inclusion of these configurations is to determine the effectiveness of multi-method approaches in the heuristic pool applied to training FFNNs.

7.3. Heuristics

Table 3 contains a list of all the standalone, low-level heuristics that are used as well as their hyper-parameter configurations. In some cases, parameters are changed dynamically throughout the training process using a *decay schedule*. Note from Table 3 that values that are configured to make use of a decay schedule are presented with the initial value first and the decay rate in brackets next to it.

The mapping of proxied heuristic state update operations implemented by the BHH in the empirical process is given in Fig. 2. In Fig. 2, cells containing **x** indicate that the associated heuristic implements that particular state parameter explicitly, and cells containing **o** indicate that the state parameter is implicitly implemented as part of the BHH algorithm.

		Entity State							Population State			
	notes	Represents the candidate solution	Is analogous to the negative gradient	Is analogous to velocity			Goes hand in hand with gradient and thus also velocity	Goes hand in hand with gradient^2 and thus also acceleration				
heuristic	**hyper-parameter**	**position**	**velocity**	**gradient**	**sum of gradients squared**	**expected position delta variance**	**expected gradient mean (hp1)**	**expected gradient variance (hp2)**	**pbest**	**ibest**	**rbest**	**gbest**
sgd	learning rate	x	x	x	-	-	-	-	-	-	-	x
momentum	learning rate, momentum (maps to hp1)	x	x	x	-	-	x	-	-	-	-	x
nag		x	x	x	-	-	x	-	-	-	-	-
adagrad	learning rate, epsilon	x	x	x	x	-	-	-	-	-	-	x
rmsprop	learning rate, rho (maps to hp2), momentum (maps to hp1), epsilon	x	x	x	-	-	-	x	-	-	-	x
adadelta	rho (maps to hp2), epsilon	x	x	x	-	x	-	x	-	-	-	x
adam	learning rate, momentum (maps to hp1), rho (maps to hp2)	x	x	x	-	-	x	x	-	-	-	x
PSO	W, C1, C2	x	x	o	-	-	-	-	x	o	-	x
DE		x	o	o	-	-	-	-	-	o	-	x
GA	mutation rate	x	o	o	-	-	-	-	-	-	-	x
BHH	burn in, replay window size, population size, reselection, reanalysis window size, normalisation, discounted rewards	x	x	x	x	x	x	x	x	x	x	x

Fig. 2. Mapping of proxied heuristic state update operations as implemented by the BHH.

Table 4
Classification datasets.

dataset	output	types	attributes	classes	instances	batch	steps	citation
iris	multivariate	real	4	3	150	16	10	[36]
car	multivariate	categorical	6	4	1728	128	14	[37]
abalone	multivariate	categorical, integer, real	8	28	4177	256	17	[38]
wine quality	multivariate	real	12	11	4898	256	20	[39]
mushroom	multivariate	categorical	22	2	8214	512	17	[40]
bank	multivariate	real	17	2	45211	512	89	[41]
diabetic	multivariate	integer	55	3	100000	1024	98	[42]

Table 5
Regression datasets.

dataset	output	types	attributes	instances	batch	steps	citation
fish toxicity	multivariate	real	7	908	64	15	[43]
housing	univariate	real	13	506	32	16	[44]
forest fires	multivariate	real	13	517	32	17	[45]
student performance	multivariate	integer	33	649	32	21	[46]
parkinsons	multivariate	integer, real	26	5875	256	23	[47]
air quality	multivariate, time series	real	15	9358	256	37	[48]
bike	univariate	integer, real	16	17389	256	68	[49]

7.4. Datasets

In the context of training FFNNs, the underlying models are trained across a number of datasets. All the datasets used in the empirical process originate from the UCI Machine Learning Repository [35]. Datasets are grouped by problem type and include seven classification and seven regression datasets. The details around the datasets used can be found in Tables 4 and 5. Each dataset is split into a training set comprising 80% of the data, and a test set comprising 20% of the data.

A number of classification datasets contain an unbalanced representation of classes. This work does not apply mechanisms to cater for class balancing, in order to eliminate as many variables and factors in the empirical process as possible.

Table 6
Model configurations.

dataset	inputs	hidden	output	biases	parameters	topology	l1 activation	l2 activation
fish toxicity	6	3	1	yes	25	dense	LReLU ($\alpha = 0.3$)	sigmoid
iris	4	5	3	yes	43	dense	LReLU ($\alpha = 0.3$)	softmax
air quality	12	8	1	yes	113	dense	LReLU ($\alpha = 0.3$)	sigmoid
housing	13	8	1	yes	121	dense	LReLU ($\alpha = 0.3$)	sigmoid
wine quality	13	10	7	yes	217	dense	LReLU ($\alpha = 0.3$)	softmax
parkinsons	21	10	1	yes	231	dense	LReLU ($\alpha = 0.3$)	sigmoid
car	21	10	4	yes	264	dense	LReLU ($\alpha = 0.3$)	softmax
forest fires	43	16	1	yes	721	dense	LReLU ($\alpha = 0.3$)	sigmoid
abalone	10	36	28	yes	1432	dense	LReLU ($\alpha = 0.3$)	softmax
bank	51	32	1	yes	1697	dense	LReLU ($\alpha = 0.3$)	softmax
bike	61	32	1	yes	2017	dense	LReLU ($\alpha = 0.3$)	sigmoid
student performance	99	32	1	yes	3233	dense	LReLU ($\alpha = 0.3$)	sigmoid
adult	108	64	1	yes	7041	dense	LReLU ($\alpha = 0.3$)	softmax
mushroom	117	64	1	yes	7617	dense	LReLU ($\alpha = 0.3$)	softmax
diabetic	2369	32	3	yes	75939	dense	LReLU ($\alpha = 0.3$)	softmax

7.5. Models

All models trained in the empirical process follow implementations of shallow FFNNs with only one hidden layer. The number of hidden units used were determined empirically. Weights are initialised by means of *Glorot uniform sampling*. The models and their configuration, as it is used for each dataset, are given in Table 6.

7.6. Performance measures

Binary cross entropy (BinXE) is used for classification problems with two classes and *sparse categorical cross entropy* (SparseCatXE) is used for classification problems with more than two classes. For the classification problems, accuracy is also measured. For regression problems, the *mean squared error* (MSE) is used as a performance metric. After training has completed, the *average rank*, based on test loss, for all configurations, is calculated at each mini-batch step.

7.7. Statistical analysis

Each experiment and configuration is trained for a maximum of 30 epochs and is repeated over 30 independent runs, for each of the datasets. No early-stopping mechanism is used. Statistical analysis is executed on the results from the test datasets. An average rank is calculated across all 30 runs, for both experimental groups and configurations, at each step, for every epoch of training.

The Shapiro-Wilk test for normality ($\alpha = 0.001$) is used to determine if the results are normally distributed. The Levene's test for equality of variance ($\alpha = 0.001$) is used. For experiments with three or more configurations, the ANOVA statistical test ($\alpha = 0.001$) is used. The Kruskal-Wallis ranked non-parametric test for statistical significance ($\alpha = 0.001$) is used for cases where data is not normally distributed. Finally, a post-hoc Tukey honest significant difference test ($\alpha = 0.001$) is used from which significant ranking is retrieved. Descriptive and critical difference plots are then retrieved from these results to provide visual aid.

7.8. Implementation

All implementations are done from first principles in Python 3.9 using Tensorflow 2.7 and Tensorflow Probability 0.15.0. The source code and data for this research is provided and made public at https://github.com/arneschreuder/masters.

8. Results

This section provides the results of the empirical process that has been conducted. Detailed discussions follow on the outcomes of each experiment.

8.1. Overview

This section provides a brief discussion on the general outcome of the empirical process as a whole and identifies some key aspects to be kept in mind when interpreting the results of the experiments.

Firstly, the BHH applies a form of online learning. As such, the BHH applies the learning mechanism during training in a single run of the training process. The training process is not repeated iteratively as is the case with some HHs.

Most of the training progress is observed to occur within the first five epochs. As a result, the BHH should apply most learning at the early stages of the training process. After five epochs, the training of most of the underlying FFNNs converges and little performance gain is observed after that point. Since this empirical process does not apply early stopping of the training process, the BHH will continue to explore the heuristic space beyond the five epoch mark.

Table 7
Empirical results showing normalised average rank and statistics for the top six low-level heuristics and three heuristic pool variants of the BHH baseline configuration, across multiple datasets, for all independent runs and epochs.

	BHH vs. Low-Level Heuristics - Average Rank (Part A)					
dataset	**adagrad**	**adam**	**rmsprop**	**bhh_gd**	**nag**	**bhh_all**
abalone	2,2215 (±1,591)	2,3989 (±1,887)	4,6172 (±2,65)	4,7032 (±2,108)	4,2731 (±1,542)	5,9376 (±2,399)
air_quality	3,6409 (±2,259)	5,4312 (±2,62)	3,4452 (±2,57)	5,0817 (±2,762)	3,8194 (±2,229)	6,686 (±3,061)
bank	2,5495 (±1,598)	2,0796 (±1,587)	3,4645 (±2,209)	4,8828 (±1,702)	4,2871 (±1,732)	6,2419 (±2,157)
bike	1,7204 (±1,384)	3,6925 (±4,004)	6,2624 (±4,58)	3,8441 (±1,398)	6,4516 (±1,02)	4,2151 (±1,361)
car	4,7634 (±0,938)	1,6226 (±1,405)	2,3269 (±1,409)	3,3473 (±1,35)	6,0785 (±0,799)	3,5624 (±1,315)
diabetic	2,7796 (±1,659)	7,1484 (±2,227)	6,7376 (±2,577)	5,2269 (±2,186)	1,8118 (±1,413)	9,3968 (±3,022)
fish_toxicity	4,2645 (±2,614)	3,6022 (±2,445)	3,5946 (±2,329)	5,4118 (±2,665)	5,8914 (±2,629)	5,829 (±2,856)
forest_fires	5,1559 (±2,922)	4,2688 (±2,984)	5,0355 (±3,143)	4,6935 (±2,759)	5,6882 (±2,215)	5,4839 (±3,107)
housing	3,4484 (±2,025)	3,3344 (±1,819)	3,6946 (±2,166)	4,4742 (±2,312)	4,6839 (±2,658)	4,3763 (±2,438)
iris	6,3946 (±1,6)	3,5839 (±2,511)	2,6968 (±1,912)	4,7473 (±2,275)	3,5548 (±2,125)	5,2204 (±3,041)
mushroom	4,4656 (±1,053)	2,1344 (±1,883)	2,4656 (±1,359)	3,4484 (±1,602)	6,3323 (±0,891)	3,6688 (±2,469)
parkinsons	2,4677 (±1,497)	2,2333 (±1,742)	3,5656 (±2,492)	4,572 (±1,934)	7,5355 (±1,44)	4,3839 (±1,861)
student_performance	2,5634 (±1,912)	11,3978 (±2,178)	12,4312 (±1,34)	5,6624 (±3,57)	3,1935 (±2,12)	5,8634 (±3,159)
wine_quality	3,2806 (±1,931)	2,1118 (±1,666)	3,6301 (±1,731)	4,7882 (±2,105)	4,1505 (±1,916)	5,1925 (±1,951)
avg rank	3,5512 (±2,25)	3,9314 (±3,423)	4,5691 (±3,517)	4,6346 (±2,364)	4,8394 (±2,384)	5,4327 (±2,9)
normalised avg rank	1	2	3	4	5	6

The BHH does not implement a type of move-acceptance strategy where the application of a heuristic to an entity is only accepted if it leads to a better solution. In some cases, the BHH then selects heuristics that yield sub-optimal results, but is shown to mostly return to optimal solutions over a number of steps.

Given the stochastic nature of the heuristic selection mechanism, sufficient samples of the performance of each heuristics-entity combination in the performance log are required for the BHH to learn. This requirement is further strengthened by the Bayesian nature of the probabilistic model implemented by the BHH. The probabilistic model implements *probability distributions of heuristic selection probabilities* and as such, insufficient samples in the performance log could render a form of random search.

By default, the BHH baseline configuration has a reanalysis interval of 10, and a replay window size of 10, which is a small window to learn from. Despite the small reanalysis interval and the small replay window size, it should be observed that the BHH exploits small performance biases and finds small performance gains throughout.

8.2. BHH vs. low-level heuristics

Tables 7 and 8 provide the empirical results in ranked format. The performance rank is calculated as the average rank produced by each heuristic, across all datasets, for all independent runs and all epochs. The average rank across all epochs produces a view on the performance of the heuristics as it relates to the entire training process. Finally, a normalised average rank is provided for the overall performance of all heuristics at the bottom of the table. The normalised average rank is calculated as a discrete normalisation of the average rank achieved across all datasets, for all independent runs and epochs.

Tables 7 and 8 show that the *bhh_gd* configuration produced the best results of the BHH variants and managed to perform well, producing generally good results across all datasets. The *bhh_gd* configuration managed to produce results that are comparable to the top three heuristics for each dataset, while the *bhh_all* and *bhh_mh* produced average results compared to all the heuristics.

The normalised average ranks provided in Tables 7 and 8 show that the *bhh_gd* configuration ranked fourth, while the *bhh_al* and *bhh_mh* configurations ranked sixth and eighth amongst all thirteen heuristic implementations respectively. These results show that the BHH generally performs well, but is not able to outperform the best heuristic for each dataset.

Tables 9 and 10 provide the empirical results in accuracy format for the classification datasets. The accuracy is measured at the maximum epoch (30).

Fig. 3 provides an illustration of the overall critical difference plots that illustrate the statistically significant differences in ranked performance for each heuristic as it relates to all datasets, across all independent runs and epochs. Although the outcomes of the *bhh_al* and *bhh_mh* configurations seem to produce average performance results, it should be noted that the performance difference between all heuristics is very small. Furthermore, the best configuration of the BHH, namely the *bhh_gd* configuration, is statistically outperformed overall by only Adagrad and Adam, yielding statistically comparable results to RMSProp and NAG. It should be noted that the standalone low-level heuristics already produce good results in general across all datasets. In this particular case, producing better performance outcomes can be hard to achieve. However, as mentioned previously, the BHH provides a generalisation capability across all datasets that is advantageous to the BHH.

Another observation that can be made is that the gradient-based heuristics generally performed much better than the MHs on all datasets. State of the art methods for training FFNNs, such as Adam, utilise gradient-based approaches that have been proven to work well on many occasions [7]. Exploration of the heuristic space leads the BHH to consider other heuristics during the training process, which could possibly result in worse performances at times. A suggestion to improve on these results is to include a move-acceptance strategy where heuristic progressions are discarded if they fail to produce better results.

Table 8
Empirical results showing normalised average rank and statistics for the bottom seven low-level heuristics and three heuristic pool variants of the BHH baseline configuration, across multiple datasets, for all independent runs and epochs.

	BHH vs. Low-Level Heuristics - Average Rank (Part B)						
dataset	adadelta	bhh_mh	ga	pso	sgd	momentum	de
abalone	5,3129 (±1,478)	8,1882 (±1,195)	11,1108 (±1,102)	11,2559 (±1,826)	8,628 (±1,019)	9,8151 (±1,16)	12,5376 (±1,329)
air_quality	5,2441 (±3,162)	6,357 (±2,303)	7,8204 (±2,265)	9,9151 (±2,288)	10,6613 (±1,606)	11,7559 (±1,473)	11,1419 (±2,236)
bank	5,672 (±1,241)	9,7495 (±1,048)	10,9817 (±1,216)	11,8376 (±1,464)	8,2774 (±1,03)	8,4774 (±1,068)	12,4989 (±1,224)
bike	5,3602 (±1,155)	7,4108 (±1,008)	9,2269 (±1,183)	9,3086 (±1,761)	10,3355 (±1,419)	10,7086 (±1,423)	12,4634 (±1,465)
car	7,7344 (±1,746)	8,8505 (±1,413)	10,7763 (±1,471)	8,3613 (±1,622)	10,2452 (±1,447)	10,9226 (±1,349)	12,4086 (±1,492)
diabetic	2,6753 (±1,629)	8,557 (±1,17)	11,2011 (±1,413)	10,7022 (±1,067)	5,9215 (±1,542)	6,3355 (±1,612)	12,5065 (±1,242)
fish_toxicity	7,914 (±3,429)	6,3849 (±2,944)	6,7043 (±2,82)	7,5731 (±2,982)	11,5785 (±1,459)	12,2301 (±1,382)	10,0215 (±2,358)
forest_fires	6,5161 (±3,082)	5,4591 (±2,668)	7,3667 (±2,37)	6,4796 (±3,354)	10,8129 (±1,207)	11,7065 (±1,325)	12,3333 (±1,923)
housing	7,5903 (±2,748)	7,5441 (±1,736)	7,8839 (±2,099)	9,9409 (±2,317)	11,4075 (±1,528)	11,2731 (±1,506)	11,3484 (±2,096)
iris	11,3527 (±1,779)	6,6075 (±2,555)	8,2473 (±1,765)	8,2731 (±4,384)	10,3796 (±1,294)	11,0548 (±1,409)	8,8871 (±3,251)
mushroom	6,5538 (±1,071)	9,0452 (±1,093)	11,5731 (±1,193)	7,872 (±0,92)	9,7527 (±1,083)	10,9785 (±1,108)	12,7097 (±1,478)
parkinsons	6,472 (±2,423)	8,3161 (±1,644)	7,7968 (±1,719)	8,4892 (±1,901)	11,7516 (±1,155)	12,5419 (±1,351)	10,8742 (±1,317)
student_performance	3,4194 (±2,006)	6,9333 (±2,44)	7,1032 (±1,989)	11,0624 (±1,067)	6,6366 (±2,023)	6,7011 (±2,242)	8,0323 (±1,935)
wine_quality	6,0011 (±2,404)	9,5935 (±1,494)	10,3387 (±1,62)	11,1602 (±1,773)	8,6344 (±1,18)	9,5269 (±1,341)	12,5903 (±1,352)
avg rank	6,2727 (±3,004)	7,7855 (±2,271)	9,1522 (±2,48)	9,4451 (±2,75)	9,6445 (±2,214)	10,2877 (±2,346)	11,4538 (±2,354)
normalised avg rank	7	8	9	10	11	12	13

Table 9
Empirical results showing average accuracy at the max epoch (30) along with statistics for the top six low-level heuristics and three heuristic pool variants of the BHH baseline configuration, across multiple datasets, for all independent runs and epochs.

	BHH vs. Low-Level Heuristics - Average accuracy at 30 epochs (Part A)					
dataset	adagrad	adam	rmsprop	bhh_gd	nag	bhh_all
abalone	27.66% (+-0.014)	27.17% (+-0.013)	27.74% (+-0.016)	26.17% (+-0.014)	27.14% (+-0.014)	25.52% (+-0.021)
bank	90.15% (+-0.003)	90.33% (+-0.003)	90.25% (+-0.003)	89.88% (+-0.003)	89.94% (+-0.003)	89.84% (+-0.003)
car	92.24% (+-0.013)	96.64% (+-0.011)	96.13% (+-0.018)	93.13% (+-0.013)	89.39% (+-0.019)	93.59% (+-0.017)
diabetic	58.14% (+-0.004)	57.23% (+-0.004)	57.85% (+-0.004)	57.76% (+-0.006)	58.53% (+-0.003)	55.42% (+-0.024)
iris	92% (+-0.046)	94.89% (+-0.042)	96.33% (+-0.035)	92.44% (+-0.077)	96.44% (+-0.037)	93.11% (+-0.056)
mushroom	99.97% (+-0)	99.98% (+-0.001)	100% (+-0)	99.98% (+-0)	99.84% (+-0.001)	99.96% (+-0.001)
wine_quality	55.06% (+-0.014)	55.62% (+-0.013)	54.86% (+-0.011)	53.35% (+-0.017)	54.64% (+-0.015)	53.57% (+-0.015)

Table 10
Empirical results showing average accuracy at the max epoch (30) along with statistics for the bottom seven low-level heuristics and three heuristic pool variants of the BHH baseline configuration, across multiple datasets, for all independent runs and epochs.

	BHH vs. Low-Level Heuristics - Average accuracy at 30 epochs (Part B)						
dataset	adadelta	bhh_mh	ga	pso	sgd	momentum	de
abalone	26.76% (+-0.013)	23.32% (+-0.018)	20.52% (+-0.019)	18.31% (+-0.029)	24.42% (+-0.017)	24.29% (+-0.015)	11.81% (+-0.044)
bank	89.85% (+-0.003)	89.44% (+-0.004)	88.79% (+-0.004)	88.56% (+-0.006)	88.86% (+-0.004)	89.01% (+-0.003)	88.3% (+-0.004)
car	88.25% (+-0.018)	80.74% (+-0.034)	71.52% (+-0.022)	81.16% (+-0.035)	69.78% (+-0.027)	70.86% (+-0.019)	69.98% (+-0.034)
diabetic	58.36% (+-0.003)	56.53% (+-0.006)	53.82% (+-0.003)	54.22% (+-0.011)	57.45% (+-0.004)	57.43% (+-0.002)	51.21% (+-0.059)
iris	83.22% (+-0.063)	92.56% (+-0.062)	88.67% (+-0.091)	93.67% (+-0.042)	80.56% (+-0.098)	80.89% (+-0.084)	84.89% (+-0.099)
mushroom	99.95% (+-0.001)	97.77% (+-0.007)	76.78% (+-0.112)	98.67% (+-0.009)	91.08% (+-0.008)	89.11% (+-0.01)	52.58% (+-0.028)
wine_quality	54.66% (+-0.012)	51.52% (+-0.014)	49.26% (+-0.017)	48.21% (+-0.038)	52.14% (+-0.014)	51.38% (+-0.014)	42.1% (+-0.042)

Section 6.9 presented the BHH hyper-parameters. A number of ablation studies were done in the interest of finding good hyper-parameters for the baseline BHH [31]. A summary of these ablation studies on different hyper-parameter configurations is given in Table 11 along with a brief conclusion of the outcomes.

A general finding in the ablations studies revealed the inter-dependencies between the hyper-parameters. For example, there is an inter-dependency between the *reselection*, *replay*, and *reanalysis* hyper-parameters. This can be seen in the ablation studies that focused on the *normalise* and *discounted rewards* hyper-parameters. The default *replay window size* (10) hyper-parameter, results in a scenario where the logged performance data is too little for the *normalise* and *discounted rewards* hyper-parameters to make a statistically significant impact.

Furthermore, the ablation studies on the *replay* and *reanalysis* hyper-parameters, which resulted in problem dependent outcomes, illustrates that a dynamic, online-learning approach is required, since some problems benefit from maintaining many records in the performance log, while others do not.

	BHH vs. Low-Level Heuristics - Average accuracy at 30 epochs (Part B)						
dataset	adadelta	bhh_mh	ga	pso	sgd	momentum	de
abalone	26.76% (+-0.013)	23.32% (+-0.018)	20.52% (+-0.019)	18.31% (+-0.029)	24.42% (+-0.017)	24.29% (+-0.015)	11.81% (+-0.044)
bank	89.85% (+-0.003)	89.44% (+-0.004)	88.79% (+-0.004)	88.56% (+-0.006)	88.86% (+-0.004)	89.01% (+-0.003)	88.3% (+-0.004)
car	88.25% (+-0.018)	80.74% (+-0.034)	71.52% (+-0.022)	81.16% (+-0.035)	69.78% (+-0.027)	70.86% (+-0.019)	69.98% (+-0.034)
diabetic	58.36% (+-0.003)	56.53% (+-0.006)	53.82% (+-0.003)	54.22% (+-0.011)	57.45% (+-0.004)	57.43% (+-0.002)	51.21% (+-0.059)
iris	83.22% (+-0.063)	92.56% (+-0.062)	88.67% (+-0.091)	93.67% (+-0.042)	80.56% (+-0.098)	80.89% (+-0.084)	84.89% (+-0.099)
mushroom	99.95% (+-0.001)	97.77% (+-0.007)	76.78% (+-0.112)	98.67% (+-0.009)	91.08% (+-0.008)	89.11% (+-0.01)	52.58% (+-0.028)
wine_quality	54.66% (+-0.012)	51.52% (+-0.014)	49.26% (+-0.017)	48.21% (+-0.038)	52.14% (+-0.014)	51.38% (+-0.014)	42.1% (+-0.042)

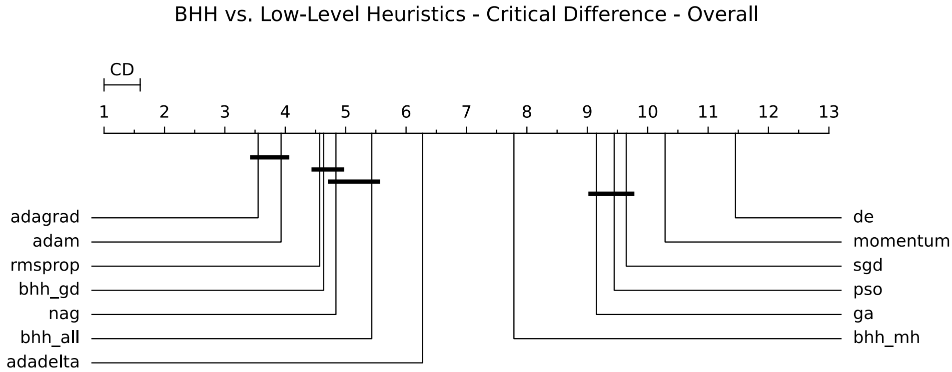

Fig. 3. Critical difference plots for the average ranks of all low-level heuristics compared to three heuristic pool variants of the baseline BHH, across all datasets, runs and epochs.

Table 11
BHH variants and their configuration. The heuristic pool configuration *gd* refers to the heuristic pool configuration where only gradient-based heuristics are included, and *mh* refers to the heuristic pool configuration where only MHs are included.

hyper-parameter	considered	conclusion
heuristic pool	all, gd, mh	gd performed best
population	5, 10, 15, 20, 25	smaller is better
credit	ibest, pbest, rbest, gbest, symmetric	problem dependent
reselection	1, 5, 10, 15, 20	larger is better
replay	1, 5, 10, 15, 20	problem dependent
reanalysis	1, 5, 10, 15, 20	problem dependent
burn in	0, 5, 10, 15, 20	lower is better
normalise	false, true	no significant difference
discounted rewards	false, true	no significant difference

9. Conclusion

The research done in this study stems from the difficult and tedious process of selecting the best heuristic for training FFNNs. The research presented in this article identified the possibility of using a different approach, referred to as HHs, to automate the heuristic selection process.

This research set out to develop a novel high-level heuristic that utilises probability theory in an online learning setting to drive the automatic heuristic selection process.

For the experimental group that compares the BHH baseline with a number of low-level heuristics, it was found that the *bhh_gd* configuration, which contains only gradient-based heuristics in the heuristic pool, performed the best out of the BHH variants, achieving an overall rank of fourth amongst thirteen heuristics that were implemented and executed on fourteen datasets. The *bhh_gd* configuration produced performance results close to that of the best low-level heuristics and was statistically outperformed only by the top two low-level heuristics. The *bhh_all* configuration, which contains only gradient-based heuristics and MHs in the heuristic pool, achieved an overall rank of sixth, and the *bhh_mh* configuration, which contains only MHs in the heuristic pool, achieved an overall rank of eighth.

Although the *bhh_gd* configuration produced performance results comparable to the best low-level heuristics, the *bhh_all* and *bhh_mh* configurations produced average results. It was found that, in general, gradient-based heuristics produced the best results, as such, it is understandable that the *bhh_gd* yielded the best performance outcomes between the different BHH variants that were implemented.

Although the BHH variants were not able to produce better results than the top low-level heuristics, the BHH variants still effectively trained the underlying FFNNs and produced good training outcomes overall. It was shown that the *bhh_gd* configuration produced the lowest variance in rank between datasets out of all of the heuristics implemented, giving the BHH the ability to generalise well to other problems.

Finally, it was shown that the BHH provides a mechanism whereby prior expert knowledge can be injected, before training starts. Future research can exploit this knowledge and provide a significant bias towards heuristics that are known to perform well on particular problem types. Future research can also investigate the scalability and effectiveness of the BHH on other model architectures such as *deep neural networks* (DNNs). Furthermore, the selection mechanism of the BHH can be extended to not just select heuristics from a heuristic pool, but also different model architectures from a model architecture pool.

CRediT authorship contribution statement

A.N. Schreuder: Writing – original draft. **A.S. Bosman:** Writing – review & editing. **A.P. Engelbrecht:** Writing – review & editing. **C.W. Cleghorn:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Appendix A. Naïve Bayes

This section aims to dissect the probabilistic model that is presented in Equation (1). The BHH implements a form of Naïve Bayes classifier, and thus independence between events can be assumed. The following derived *probability mass functions* (PMFs) are provided as fundamental building blocks to show the mechanism by which the BHH learns.

The independence between events for class label $\boldsymbol{H}$, simply yields the PMF of the Multinomial distribution as presented below:

$$\begin{aligned} P(\boldsymbol{H}|\boldsymbol{\theta}) &\propto \prod_{i=1}^{I}\prod_{k=1}^{K} P(h_{i,k}|\theta_k) \\ &\propto \prod_{i=1}^{I}\prod_{k=1}^{K} \theta_k^{\mathbb{1}_1(h_{i,k})} \\ &\propto \prod_{k=1}^{K} \theta_k^{\sum_{i=1}^{I} \mathbb{1}_1(h_{i,k})} \\ &\propto \prod_{k=1}^{K} \theta_k^{N_k} \end{aligned} \tag{A.1}$$

where N_k is a summary variable such that $N_k = \sum_{i=i}^{I} \mathbb{1}_1(h_{i,k})$, denoting the count of occurrences of the event h_i taking on class k in I independent, identical runs.

The independence between events $\boldsymbol{E}$, given class label $\boldsymbol{H}$, is denoted by the likelihood of $\boldsymbol{E}$, conditional to the occurrence of heuristic k and model parameter $\boldsymbol{\phi}$ as follows:

$$\begin{aligned} P(\boldsymbol{E}|\boldsymbol{H};\boldsymbol{\phi}) &\propto \prod_{i=1}^{I}\prod_{j=1}^{J}\prod_{k=1}^{K} P(e_{i,j,k}|h_{i,k};\phi_{j,k}) \\ &\propto \prod_{i=1}^{I}\prod_{j=1}^{J}\prod_{k=1}^{K} \phi_{j,k}^{\mathbb{1}_1(e_{i,j,k})\mathbb{1}_1(h_{i,k})} \\ &\propto \prod_{j=1}^{J}\prod_{k=1}^{K} \phi_{j,k}^{\sum_{i}^{I}\left[\mathbb{1}_1(e_{i,j,k})\mathbb{1}_1(h_{i,k})\right]} \\ &\propto \prod_{j=1}^{J}\prod_{k=1}^{K} \phi_{j,k}^{N_{j,k}} \end{aligned} \tag{A.2}$$

where $N_{j,k}$ is a summary variable such that $N_{j,k} = \sum_{i=i}^{I} \mathbb{1}_1(e_{i,j,k})\mathbb{1}_1(h_{i,k})$, denoting the count of occurrences of the events e_i taking on class j and h_i taking on class k, i.e. the count of occurrences of both entity j and heuristic k occurring together in I independent, identical runs.

Finally, the independence between events for the performance criteria $\boldsymbol{C}$, given class label $\boldsymbol{H}$, is denoted by the likelihood of $\boldsymbol{C}$, conditional to the occurrence of heuristic k and model parameter $\boldsymbol{\psi}$ as given below:

$$\begin{aligned}
P(\boldsymbol{C}|\boldsymbol{H};\boldsymbol{\psi}) &\propto \prod_{i=1}^{I}\prod_{k=1}^{K} P(c_{i,k}|h_{i,k};\psi_k) \\
&\propto \prod_{i=1}^{I}\prod_{k=1}^{K} \psi_k^{\mathbb{1}_1(c_{i,k})\mathbb{1}_1(h_{i,k})}(1-\psi_k)^{\mathbb{1}_0(c_{i,k})\mathbb{1}_1(h_{i,k})} \\
&\propto \prod_{k=1}^{K} \psi_k^{\sum_{i=1}^{I}\mathbb{1}_1(c_{i,k})\mathbb{1}_1(h_{i,k})}(1-\psi_k)^{\sum_{i=1}^{I}\mathbb{1}_0(c_{i,k})\mathbb{1}_1(h_{i,k})} \\
&\propto \prod_{k=1}^{K} \psi_k^{N_{1,k}}(1-\psi_k)^{N_{0,k}} \\
&\propto \prod_{k=1}^{K} \psi_k^{N_{1,k}}(1-\psi_k)^{(N_k-N_{1,k})}
\end{aligned} \tag{A.3}$$

where N_k is the same summary variable as described for Equation (A.1). $N_{1,k}$ is a summary variable such that $N_{1,k} = \sum_{i=1}^{I} \mathbb{1}_1(c_{i,k})\mathbb{1}_1(h_{i,k})$, denoting the count of occurrences of the events c_i taking on a success (i.e. $c_i = 1$) and h_i taking on class k, i.e. the count of occurrences of both succeeding in the performance criteria and heuristic k occurring together in I independent, identical runs. Similarly, $N_{0,k} = N_k - N_{1,k}$ denotes the count of occurrences of the events c_i taking on a failure (i.e. $c_i = 0$) and h_i taking on class k.

Equations (A.1), (A.2) and (A.3) can be substituted into the proportional evaluation of the predictive model as given in Equation (1), resulting in

$$\begin{aligned}
P(\boldsymbol{H}|\boldsymbol{E},\boldsymbol{C};\theta,\boldsymbol{\phi},\boldsymbol{\psi}) &\propto P(\boldsymbol{E}|\boldsymbol{H};\boldsymbol{\phi})P(\boldsymbol{C}|\boldsymbol{H};\boldsymbol{\psi})P(\boldsymbol{H}|\theta) \\
&\propto \left[\prod_{j=1}^{J}\prod_{k=1}^{K}\phi_{j,k}^{N_{j,k}}\right]\left[\prod_{k=1}^{K}\psi_k^{N_{1,k}}(1-\psi_k)^{(N_k-N_{1,k})}\right]\left[\prod_{k=1}^{K}\theta_k^{N_k}\right]
\end{aligned} \tag{A.4}$$

Consider the practical implementation of the predictive model as shown in Equation (A.4). Computationally, the equation presented in Equation (A.4) will underflow on a real computer if the resulting probabilities are very small.

A.1. Numerical stability

When Equation (A.4) is evaluated, the numerical stability is shown to underflow if the resulting probabilities from its parts are very small. Multiplication of multiple fractional parameters leads to an even smaller fractional number. Probabilities might be very low at some points during training. A solution to the aforementioned problem is to apply the *log-sum-exp* trick. The transformation of Equation (A.4) using the log-sum-exp trick is given as

$$LSE(P(h_k|e_j,c_1;\boldsymbol{\theta},\boldsymbol{\phi},\boldsymbol{\psi})) = \ln(\exp(\phi_{j,k}) + \exp(\psi_k) + \exp(\theta_k)) \tag{A.5}$$

References

[1] A.S. Rakitianskaia, A.P. Engelbrecht, Training feedforward neural networks with dynamic particle swarm optimisation, Swarm Intell. 6 (2012) 233–270.
[2] J.A. Allen, S. Minton, Selecting the right heuristic algorithm: runtime performance predictors, in: Conference of the Canadian Society for Computational Studies of Intelligence, Springer, 1996, pp. 41–53.
[3] E.K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, R. Qu, Hyper-heuristics: a survey of the state of the art, J. Oper. Res. Soc. 64 (2013) 1695–1724.
[4] G.S. Nel, A hyperheuristic approach towards the training of artificial neural networks, Ph.D. thesis, University of Stellenbosch, 2021.
[5] D. Oliva, M.S. Martins, A Bayesian based hyper-heuristic approach for global optimization, in: 2019 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2019, pp. 1766–1773.
[6] J. Schmidhuber, Deep learning in neural networks: an overview, Neural Netw. 61 (2015) 85–117.
[7] D. Kingma, J. Ba, Adam: a method for stochastic optimization, in: International Conference on Learning Representations, 2014.
[8] A. Engelbrecht, Computational Intelligence: An Introduction, Wiley, 2007.
[9] N. Qian, On the momentum term in gradient descent learning algorithms, Neural Netw. 12 (1999) 145–151.
[10] I. Sutskever, J. Martens, G. Dahl, G. Hinton, On the importance of initialization and momentum in deep learning, in: International Conference on Machine Learning, PMLR, 2013, pp. 1139–1147.
[11] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, J. Mach. Learn. Res. 12 (2011).
[12] M.D. Zeiler, Adadelta: an adaptive learning rate method, arXiv preprint, arXiv:1212.5701, 2012.
[13] G. Hinton, N. Srivastava, K. Swersky, Neural Networks for Machine Learning Lecture 6a Overview of Mini-Batch Gradient Descent, 2012, p. 2, Cited on 14.
[14] Y. Bengio, Gradient-based optimization of hyperparameters, Neural Comput. 12 (2000) 1889–1900.

[15] C. Blum, A. Roli, Metaheuristics in combinatorial optimization: overview and conceptual comparison, ACM Comput. Surv. 35 (2003) 268–308.
[16] A.B. Van Wyk, An Analysis of Overfitting in Particle Swarm Optimised Neural Networks, Master's thesis, University of Pretoria, 2014.
[17] S.A. Abdulkarim, A.P. Engelbrecht, Time series forecasting with feedforward neural networks trained using particle swarm optimizers for dynamic environments, Neural Comput. Appl. 33 (2021) 2667–2683.
[18] Y. Xue, Y. Tong, F. Neri, An ensemble of differential evolution and Adam for training feed-forward neural networks, Inf. Sci. 608 (2022) 453–471.
[19] K. Khan, A. Sahai, A comparison of ba, ga, pso, bp and lm for training feed forward neural networks in e-learning context, Int. J. Intell. Syst. Appl. 4 (2012) 23.
[20] E.K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, J.R. Woodward, A classification of hyper-heuristic approaches, in: Handbook of Metaheuristics, Springer, 2010, pp. 449–468.
[21] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, S. Schulenburg, Hyper-heuristics: an emerging direction in modern search technology, in: Handbook of Metaheuristics, Springer, 2003, pp. 457–474.
[22] J. Grobler, A.P. Engelbrecht, G. Kendall, V. Yadavalli, Investigating the use of local search for improving meta-hyper-heuristic performance, in: 2012 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2012, pp. 1–8.
[23] S.A. Van der Stockt, A.P. Engelbrecht, Analysis of selection hyper-heuristics for population-based metaheuristics in real-valued dynamic optimization, Swarm Evol. Comput. (2018).
[24] J.A. Vrugt, B.A. Robinson, Improved evolutionary optimization from genetically adaptive multimethod search, Proc. Natl. Acad. Sci. USA 104 (2007) 708–711.
[25] B.K. Hackenberger, Bayes or not Bayes, is this the question?, Croat. Med. J. 60 (2019) 50.
[26] D. Wackerly, W. Mendenhall, R.L. Scheaffer, Mathematical statistics with applications, in: Cengage Learning, 2014.
[27] R. Eberhart, J. Kennedy, Particle swarm optimization, in: Proceedings of the IEEE International Conference on Neural Networks, vol. 4, 1995, pp. 1942–1948.
[28] E. Özcan, B. Bilgin, E.E. Korkmaz, Hill climbers and mutational heuristics in hyperheuristics, in: Parallel Problem Solving from Nature-PPSN IX, Springer, 2006, pp. 202–211.
[29] J. Görtler, R. Kehlbeck, O. Deussen, A visual exploration of Gaussian processes, Distill 4 (2019) e17.
[30] L. Rabiner, B. Juang, An introduction to hidden Markov models, IEEE ASSP Mag. 3 (1986) 4–16.
[31] A. Schreuder, Training Feedforward Neural Networks with Bayesian Hyper-Heuristics, Master's dissertation, University of Pretoria, Department of Computer Science, Pretoria, South Africa, 2022.
[32] F. Van den Bergh, A.P. Engelbrecht, A convergence proof for the particle swarm optimiser, Fundam. Inform. 105 (2010) 341–374.
[33] E. Mezura-Montes, J. Velázquez-Reyes, C.A. Coello Coello, A comparative study of differential evolution variants for global optimization, in: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, 2006, pp. 485–492.
[34] A. Lambora, K. Gupta, K. Chopra, Genetic algorithm - a literature review, in: 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), 2019, pp. 380–384.
[35] D. Dua, C. Graff, Uci machine learning repository, http://archive.ics.uci.edu/ml, 2017.
[36] R.A. Fisher, The use of multiple measurements in taxonomic problems, Ann. Eugen. 7 (1936) 179–188.
[37] M. Bohanec, V. Rajkovic, Knowledge acquisition and explanation for multi-attribute decision making, in: 8th Intl Workshop on Expert Systems and Their Applications, Citeseer, 1988, pp. 59–78.
[38] S.G. Waugh, Extending and benchmarking Cascade-Correlation: extensions to the Cascade-Correlation architecture and benchmarking of feed-forward supervised artificial neural networks, Ph.D. thesis, University of Tasmania, 1995.
[39] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, J. Reis, Modeling wine preferences by data mining from physicochemical properties, Decis. Support Syst. 47 (2009) 547–553.
[40] J.C. Schlimmer, Concept acquisition through representational adjustment, Ph.D. thesis, University of California, Irvine, 1987.
[41] S. Moro, P. Cortez, P. Rita, A data-driven approach to predict the success of bank telemarketing, Decis. Support Syst. 62 (2014) 22–31.
[42] B. Strack, J.P. DeShazo, C. Gennings, J.L. Olmo, S. Ventura, K.J. Cios, J.N. Clore, Impact of HbA1C measurement on hospital readmission rates: analysis of 70,000 clinical database patient records, BioMed Res. Int. 2014 (2014).
[43] M. Cassotti, D. Ballabio, R. Todeschini, V. Consonni, A similarity-based qsar model for predicting acute toxicity towards the fathead minnow (pimephales promelas), SAR QSAR Environ. Res. 26 (2015) 217–243.
[44] D. Harrison Jr., D.L. Rubinfeld, Hedonic housing prices and the demand for clean air, J. Environ. Econ. Manag. 5 (1978) 81–102.
[45] P. Cortez, A. Morais, A data mining approach to predict forest fires using meteorological data, in: New Trends in Artificial Intelligence, Proceedings of the 13th EPIA, APPIA, Guimarães, Portugal, 2007, pp. 512–523.
[46] P. Cortez, A.M.G. Silva, Using data mining to predict secondary school student performance, in: Proceedings of 5th Future Business Technology Conference (FUBUTEC), EUROSIS-ETI, 2008, pp. 5–12.
[47] A. Tsanas, M. Little, P. McSharry, L. Ramig, Accurate telemonitoring of Parkinson's disease progression by non-invasive speech tests, IEEE Trans. Biomed. Eng. 57 (4) (2009) 884–893.
[48] S. De Vito, E. Massera, M. Piga, L. Martinotto, G. Di Francia, On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario, Sens. Actuators B, Chem. 129 (2008) 750–757.
[49] H. Fanaee-T, J. Gama, Event labeling combining ensemble detectors and background knowledge, Prog. Artif. Intell. 2 (2014) 113–127.