Noname manuscript No.
(will be inserted by the editor)

Distributed Learning Automata-Based Scheme for Classification Using Novel Pursuit Scheme

Morten Goodwin, Anis Yazidi

the date of receipt and acceptance should be inserted later

Abstract Learning Automata (LA) is a popular decision making mechanism to "determine the optimal action out of a set of allowable actions" [1]. The distinguishing characteristic of automata-based learning is that the search for the optimising parameter vector is conducted in the space of probability distributions defined over the parameter space, rather than in the parameter space itself [2]. Recently, Goodwin and Yazidi pioneered the use of Ant Colony Optimisation (ACO) for solving classification problems [3]. In this paper, we propose a novel classifier based on the theory of LA. The classification problem is formulated as a deterministic optimization problem involving a team of LA that operate collectively to optimize an objective function.

Although many LA algorithms have been devised in the literature, those LA schemes are not able to solve deterministic optimization problems as they suppose that the environment is stochastic. In this paper, we develop a novel pursuit LA which can be seen as the counterpart of the family of pursuit LA developed for stochastic environments [4]. While classical pursuit LA are able to pursue the action with the highest reward estimate, our pursuit LA rather pursues the collection of actions that yield the highest performance. The theoretical analysis of the pursuit scheme does not follow classical LA proofs and can pave the way towards more schemes where LA can be applied to solve deterministic optimization problems.

When applied to classification, the essence of our scheme is to search for a separator in the feature space by imposing a LA based random walk in a

Morten Goodwin
Centre for Artificial Intelligence Research
University of Agder
E-mail: morten.goodwin@uia.no

Anis Yazidi
Dept. of Computer Science
Oslo Metropolitan University
E-mail: anis.yazidi@oslomet.no

grid system. To each node in the gird we attach an LA, whose actions are the choice of the edges forming the separator. The walk is self-enclosing, i.e., a new random walk is started whenever the walker returns to starting node forming a closed classification path yielding a multiedged polygon. In our approach, the different LA attached at the different nodes search for a polygon that best encircles and separates each class. Based on the obtained polygons, we perform classification by labelling items encircled by a polygon as part of a class using ray casting function.

Seen from a methodological perspective, PolyPursuit-LA has appealing properties compared to SVM. In fact, unlike PolyPursuit-LA, the SVM performance is dependent on the right choice of kernel function (e.g. Linear Kernel, Gaussian Kernel)— which is considered a "black art". PolyPursuit-LA can find arbitrarily complex separators in the feature space.

Experimental results from both synthetic and real-life data show that our scheme is able to perfectly separate both simple and complex patterns outperforming existing classifiers, including polynomial and linear SVM, without the need of any "kernel trick". We believe that the results are impressive given the simplicity of PolyPursuit-LA compared to other approaches such as SVM.

Keywords Classification, Learning Automata, Polygons.

1 Introduction

Learning Automata (LA) have been used in systems that have incomplete knowledge about the Environment in which they operate [1,5–11]. The learning mechanism attempts to learn from a *stochastic Teacher* which models the Environment. In his pioneering work, Tsetlin [12] attempted to use LA to model biological learning. In general, a random action is selected based on a probability vector, and these action probabilities are updated based on the observation of the Environment's response, after which the procedure is repeated.

The term "Learning Automata" was first publicised and rendered popular in the survey paper by Narendra and Thathachar. The goal of LA is to "determine the optimal action out of a set of allowable actions" [1]. The distinguishing characteristic of automata-based learning is that the search for the optimising parameter vector is conducted in the space of probability distributions defined over the parameter space, rather than in the parameter space itself [2].

With regard to applications, the entire field of LA and stochastic learning has had a myriad of applications [5–7, 9, 10], which (apart from the many applications listed in these books) include solutions for problems in network and communications [13–16], network call admission, traffic control, quality of service routing, [17–19], distributed scheduling [20], training hidden Markov models [21], neural network adaptation [22], intelligent vehicle control [23], and even fairly theoretical problems such as graph partitioning [24].

In addition to these fairly generic applications, with a little insight, LA can be used to assist in solving (by, indeed, learning the associated parameters) the stochastic resonance problem [25], the stochastic sampling problem in computer graphics [26], the problem of determining roads in aerial images by using geometric-stochastic models [27], and various location problems [28]. Similar learning solutions can also be used to analyse the stochastic properties of the random waypoint mobility model in wireless communication networks [29], to achieve spatial point pattern analysis codes for GISs [30], to digitally simulate wind field velocities [31], to interrogate the experimental measurements of global dynamics in magneto-mechanical oscillators [32], and to analyse spatial point patterns [33]. LA-based schemes have already been utilised to learn the best parameters for neural networks [22], optimising QoS routing [19], and bus arbitration [14] – to mention a few other applications.

Although many LA algorithms have been devised in the literature, those LA schemes are not able to solve deterministic optimization problems as they suppose that the environment is stochastic. In this paper, we develop a novel pursuit LA which can be seen as the counterpart of the family of pursuit LA developed for stochastic environments [4]. While classical pursuit LA are able to pursue the action with the highest reward estimate, our pursuit LA rather pursues the collection of actions that yield the highest performance. The theoretical analysis of the pursuit scheme does not follow classical LA proofs and can pave the way towards more schemes where LA can be applied to solve deterministic optimization problems.

Supervised Learning is one of the most central tasks in Machine Learning and Pattern Recognition. However, the latter task becomes intrinsically challenging whenever the data to be classified is not easily separable in the feature space. A myriad of classification algorithms have been proposed in the literature with a variety of behaviours and limitations [34–36]. Examples of these algorithms include Neural Networks, SVM and Decision trees.

A broad class of classification algorithms, including SVM and perception relies upon defining a mathematical function with weights as unknown that efficiently can separate two or more classes of data where the unknown weights are learned for the training data. These functions are called kernels in the case of SVM and linear functions in case of perception.

However, the main difficulty is to choose the nature of this function (linear or non-linear). Often, the "best" hyperplane to separate classes does not follow the function's mathematical properties. For example, the "best" line can be a polygon encircling certain data points, which is not a function and therefore cannot straightforward be outputted by SVM or similar classifiers.

Different types of kernels were proposed in SVM so that deciding the "shape" of the separator would not be limited to linear separators. However, the accuracy of the SVM is dependent on making the correct choice of the kernel function which is not an easy task given the unlimited number of available kernels.

Figure 1 shows an example of labeled data where it is not possible to perfectly separate the data with one function simply because any line that perfectly separates the data has multiple $y-$values of a certain number of the $x-$values — which defies the definition of mathematical functions. SVM deals with this by projecting the data in high dimensional space using the "kernel trick" in which the data can be easily separable.

This paper introduces PolyPursuit-LA, a novel classification scheme operating in two dimensions using LA and that not involving a "kernel trick" whenever the data is not easily separable. The presented approach deals with classification problem in two-dimensional Euclidean by building "separator" with many-sided polygons. The polygons are extrapolated from reinforced random walkers showing a preference towards encapsulating of all items from one class and excluding from the encapsulation any items from other classes. Thus, emerging polygons encapsulate each class in such a manner that they can be used as classifiers. The classification takes place by resorting to ray casting unknown items that identifies which polygon(s), if any, an item is contained in. Each item is labeled depending on which polygons the item is part of. It is worth mentioning that recently, Goodwin and Yazidi pioneered the use of ant colony optimisation (ACO) for solving classification problems [3]. In this paper, we propose a novel classifier in two-dimensional feature spaces based on the theory of Learning Automata (LA). There are some fundamentals. As Dorigo and Gianni Di Caro point out: "the main difference lies in the fact that in ACO the environment signals (i.e., the ants) are stochastically biased, by means of their probabilistic transition rule, to direct the learning process towards the most interesting regions of the search space. That is, the whole environment plays a key, active role to learn good state-action pairs" [37]. Our devised pursuit LA is a contribution in itself in the field of LA as it is the first LA that operates in deterministic environment. Many Pursuit LA algorithms have been already devised for stochastic environment where the LA pursues the action with highest average reward [4,38]. In our case, we operate under a deterministic environment and thus the idea of average reward is not existent. We should emphasize that the current theoretical analysis presented in this article is general and can be provide a fundamental for development for more schemes where LA can be applied to solve deterministic optimization problems.

1.1 Outline

The paper is organised as follows. Section 2 introduces the problem that we attempt to solve.

Section 3 reviews relevant state-of-the-art approaches in the area of classifiers. Section 4 provides a brief introduction to the theory of LA which is the basis for our approach reckoned as PolyPursuit-LA. Section 5 continues by introducing our solution: PolyPursuit-LA as a method for creating polygons for classification using two classes and corresponding results. We then ex-

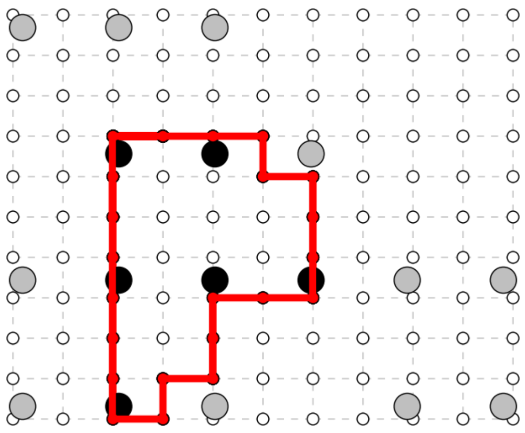

Fig. 1 Example of simple two class classification scenario with the classes Black (T_1) and Gray (T_2)

tends the two-class approach to a multi-class classification problem. Section 6 provide some comprehensive experimental results. In Section 7, we draw conclusions and provide insights into further areas for study.

2 Problem formulation

Classification of unknown items based on labeled data is a supervised learning problem. In line with common practice, the problem is divided into two phases, namely (1) training and (2) classification:

1. **Training phase:** The aim of this phase is to create polygons that encircles classes of items so that the polygons separate the training classes from each other.
2. **Classification phase:** In this phase, we use the polygons as a basis to determine to which class a new unknown item to be classified belongs. This is achieved by finding the polygon(s) of which it is a part.

   Further, this paper presents two distinct variants of PolyPursuit-LA:

1. LA polygon classification for two-class classification problems.
2. LA polygon classification for multi-class classification problems.

2.0.1 The training phase

This training phase can be formulated as a combinatorial optimisation problem. The training data, T, consists of multiple classes. The data is mapped to a two dimensional Euclidean space as follows. A grid-like bidirectional planar graph $G(V, E)$ with vertices $i \in V$ and edges $(i, j) \in E$ is created where

$i, j \in V$. All vertices have $x-$ and $y-$ coordinates and corresponding edges so that an edge (i, j) represents the possibility to move from vertex i to j. The vertices in the graph are defined so that the first vertex, 1 always has lower $x-$ and $y-$values than all the training data. Similarly, the last vertex, N, has $x-$ and $y-$values larger than the training data. Hence, all the training data $t_i \in T$ lies somewhere between vertices 1 and N, $1 < t_i < N \forall_{t_i \in T}$.

Two-class classification problem An example is shown in Figure 1. In this example T consists of 17 items, 6 in the black class T_1 and 11 in the gray class T_2. The grid $G(V, E)$ is created so that all items is located in the grid.

To deduct, the main purpose of the training phase is to find a polygon, s, that encircles and separates the training classes.[1] Using the example form Figure 1 the task is to find an s that encircles the first training data T_1, but not T_2 — a polygon that efficiently separates T_1 from T_2.

A polygon s is therefore a list of vertices and edges so that the first vertex in s is equal to the last vertex in s, and all vertices are connected together with corresponding edges. All polygons that are possible to extract from the graph $G(V, E)$ are contained in the search space $\mathbf{S}$.

When operating with two classes, say T_1 and T_2, there is only a need for one polygon to perfectly separate the data.

Whether or not a training element t_i is inside a polygon $s \in \mathbf{S}$ is defined as the following:

$$\begin{aligned} h(t_i, s) = & \quad 1 \textit{ if } t_i \textit{ is inside of } s \\ h(t_i, s) = & \quad 0 \textit{ otherwise} \end{aligned} \tag{1}$$

Ideally, all items in class one (e.g. T_1) should lie within the polygon, while all items in other classes should fall outside the polygon. Any item, t_i from class T_1 that is correctly located within the polygon s will yield $h(t_i, s) = 1$, similarly, any item, t_j not part of class T_1 and correctly lying outside of the polygon s will yield $1 - h(t_j, s) = 1$. For all other items, $h(., s)$ will produce 0. Further let $f(s)$ be a function that combines $h(t_i, s)$ for all $t_i \in T$ so that an ideal polygon that encapsulates all items in T_1 and no other items will yield an $f(s) = 1$. An incorrect polygon, one that incorrectly encapsulates all other items than T_2 and none in T_1, will yield an $f(s) = 0$.[2]

The overall aim of the training phase can therefore be stated as attempting to find a polygon s^* for each class, consisting of vertices and edges, that minimises $f(s^*)$. Thus, formally speaking, we aim to find an $s^* \in \mathbf{S}$ so that $f(s^*) \leq f(s) \in \mathbf{S}$. Keeping this aim in mind, we use LA as explained in section 4.

Multi-class classification problem In the case of classification with more than two classes, one polygon is not sufficient to separate all classes. There is a need to separate each class from the others. For example, let us suppose there

[1] In the case of multi-class classification, more than one polygon is required.

[2] $f(s)$ is formally defined in section 5.

are three classes: T_1,T_2, and T_3. In simple terms, we need a classifier that identifies an item as belonging to T_1, one to T_2, and one to T_3. This is done by finding one polygon that separates T_1 from the rest, and so on.

The output from the training phase is therefore a list of classifiers rather than one single s^*. Following the same example with three classes, we have one classifier that decides whether an item is part of T_1, $s^*_{T_1}$, and one that decides whether an item is part of T_2, $s^*_{T_2}$. If it is neither part of T_1 nor T_2, it naturally belongs to T_3. Hence, the number of classifier is one less than the number of classes.

For N classes, we get the following $N-1$ classifiers:

$$\mathbf{s}^*_{\mathbf{all}} = \{s^*_{T_1}, s^*_{T_2}, \ldots, s^*_{T_{N-1}}\} \tag{2}$$

2.0.2 The classification phase

The classification phase resorts to the polygons from the training phase. The classification task is to find to which class a new item with unknown label, t_k, belongs. The problem formulation depends on whether we are dealing with (1) two-class classification problems or (2) multi-class classification problems.

Two-class classification problem Since the training phase produces one polygon, s^*, the two-class classification problem is reduced to simply determining whether a new item is located within or outside s^*. The problem can be stated as follows: given the polygon s^* and a new item with unknown label, t_k, which class does t_k belong to? Using the update function from equation 1, given two classes T_1 and T_2 and the polygon s^* we can define the following decision rules:

$$\begin{aligned} &t_k \textit{ is of class } T_1 \textit{ if } h(t_k, s^*) = 1 \\ &t_k \textit{ is of class } T_2 \textit{ if } h(t_k, s^*) = 0 \end{aligned} \tag{3}$$

Multi-class classification problem The classification phase uses the set of polygons, $\mathbf{s}^*_{\mathbf{all}}$ (see equation 2), from the training phase. The task is to classify an unlabeled item t_k. The following decision rules are used in the case of multi-class classification:

$$\begin{array}{ll} t_k \textit{ is of class } T_1 & \textit{if } h(t_k, s^*_{T_1}) = 1 \\ t_k \textit{ is of class } T_2 & \textit{if } h(t_k, s^*_{T_2}) = 1 \\ \ldots & . \\ t_k \textit{ is of class } T_{N-1} & \textit{if } h(t_k, s^*_{T_{N-1}}) = 1 \\ t_k \textit{ is of class } T_N & \textit{otherwise} \end{array} \tag{4}$$

In simple terms, the above classification rules mean simply that if the item to be classified is part of the first polygon $s^*_{T_1}$, it should be classified as the label corresponding to the first polygon, T_1. Otherwise, if it is part of $s^*_{T_2}$, it should be classified as T_2, and so on. However, if the item is not part of any of the polygons of the T_{N-1} classes, it will be labelled as the class T_N.

3 Related Work

3.1 Distributed LA on a Graph

Misra and Oommen pioneered the concept of LA on a graph using pursuit LA [13,39,40] for solving the stochastic shortest path problem. Li [41] used a type of S Learning Automata [42] to find the shortest path in a graph. Beigy and Meybodi [43] provided the first proof in the literature that demonstrates the convergence of distributed LA on a graph for a reward inaction LA. Concerning applications of distributed LA on a graph in the field of computer communications, we refer the reader to the work of Torkestani and collaborators [44–46].

3.2 Some related LA work for classification and function optimisation

In order to put our work in the right perspective, we will briefly discuss different classification schemes relevant to this work from the field of LA theory. A similar work to ours is due to Thathachar and Sastry [47] where the authors use a team of LA in order to find the optimal discriminant function in a feature space. The discriminant functions are parametrised, and a parameter is attached to each that is to be learned. Subsequently, Santharam et al. [48] proposed using continuous LA in order to deal with the disadvantages of discretisation, thus allowing an infinite number of actions. For an excellent review on the application of LA to the field of Pattern Recognition we refer the reader to [49] . In [50], Zahiri devised an LA based classifier that operates using hypercubes in a recursive manner. We believe that the latter idea can be used to extend our current solution: PolyPursuit-LA for handling multi-dimensional classification problems. In [51], the authors have proposed LA optimisation methods for multimodal functions. Through experimental settings, the performance of these algorithms were shown to outperform genetic algorithms. In [52], the authors propose genetic LA for optimising functions. Similarly, the work [53] proposed genetic algorithms for classifiers.

3.3 ACO for classification

Swarm intelligence denotes a set of nature-inspired paradigms that have received a lot of attention in computer science due to their simplicity and adaptability [54]. ACO figures among the most popular swarm intelligence algorithms due to its ability to solve many optimisation problems. ACO involves artificial ants operating a reinforced random walk over a graph. The ants release pheromones in favorable paths which subsequent ant members follow creating a reinforcement learning based behaviour. The colony of ants will thus concentrate its walk on the most favorable paths and in consequence iteratively optimise the solution [55].

ACO was first applied to find shortest path from a source to a sink in bidirectional graphs. ACO has later found increased popularity due to its low complexity and ability to work in dynamic environments. The flexibility of ACO is apparent as it has been shown to be used in a wide variety of problems [54] such as solving NP hard problems [56], traffic optimisation [57], evacuation path computation in emergency situations [58, 59], classification with neural networks, [60] Bayesian classifiers [61], rule based classification [62] and routing in communication networks.

Finding the shortest path in a bidirectional graph with vertices and edges $G(V, E)$ using ACO in its simplest form works as follows. Artificial ants move from vertex to vertex. An ant that finds a route s from the source v_s to the sink v_t will release pheromones $\tau_{i,j}$ corresponding to all edges $e_{i,j} \in s$.

A considerable amount of work for ACO classification tasks was reported in the literature. The existing approaches fall into two main categories: (1) Rule based extractors, and (2) Hybrid approaches involving ACO that attempt to improve and enhance the quality of existing classifiers.

The rule based classifiers [62–64] construct graphs by letting the ants walk with preference towards common occurring examples so that strong pheromone trails are used as rules in the classifier, or a set of IF-THEN rules [65]. The most notable rule based ACO classifiers are probably the AntMiner series [66] including: AntMiner [63, 67], AntMiner2 [63], AntMiner3 [62], AntMiner+ [64] and new variantssuch as MAnt-Miner [68] and TACO-miner [69]. All the aforementioned AntMiner variants rely on the idea of letting the ant walk "on" examples so that the pheromone trails can yield usable rules. Many other rule-based schemes exist, including rule extractor for web page classification [70].

The hybrid approaches use ACO to improve the performance of legacy classifiers, for example in feed forward neural networks [71–73]. For example, in the latter case, this is achieved by letting the ants minimize a function consisting of a set of decision variables corresponding to the neuron parameter weights. Furthermore, there is a multitude of hybrid ACO variants for Bayesian networks [74–78], multi-net classifiers [61], fuzzy rule-based systems [79], and rule pruning [80].

Classification problems usually involve usually finding classification boundaries in feature spaces. Among the early and most popular classifier figures the perception algorithm. There are other classes of algorithms that are worth investigating as an alternative method to LA for solving our classification problem which include nature-inspired algorithms such as Butterfly [81], Multilayer Perceptron which we compared to in the experiments, Convolutional Neural Networks [82], Particle Swarm [83], and commercial softwares such as IBM Bluemix [84].

Perception works based on "error-driven learning" where it iteratively learns a linear separator model by adjusting the weights of the model whenever it has misclassified an item from the training data.

However, the major limitation of perception algorithm is the fact that it only finds a linear decision boundary which works well for linearly separable data but fails to handle the case of non-linearly separable data. In order to deal with the limitation of linear classifier, non-linear SVM variants were proposed. SVM tries to circumvent over-fitting by choosing the maximal margin hyperplane where the margin is the shortest distance between the decision boundary and any of the data points. Despite the well recognised performance of SVM in the machine learning community, the task of choosing the right type of kernel, for example, linear, polynomial, Gaussian is considered to be a black art!

A powerful concept in SVM is the "kernel trick" where the input data are mapped to higher-dimensional feature space in which the data items can be separable.

4 Learning Automata

In the field of Automata Theory, an automaton [5–7,9,10] is defined as a quintuple composed of a set of states, a set of outputs or actions, an input, a function that maps the current state and input to the next state, and a function that maps a current state (and input) into the current output.

Definition 1: A LA is defined by a quintuple $\langle A, B, Q, F(.,.), G(.)\rangle$, where:

1. $A = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is the set of outputs or actions that the LA must choose from, and $\alpha(t)$ is the action chosen by the automaton at any instant t.
2. $B = \{\beta_1, \beta_2, \ldots, \beta_m\}$ is the set of inputs to the automaton. $\beta(t)$ is the input at any instant t. The set B can be finite or infinite. The most common LA input is $B = \{0, 1\}$, where $\beta = 0$ represents reward, and $\beta = 1$ represents penalty.
3. $Q = \{q_1, q_2, \ldots, q_s\}$ is the set of finite states, where $Q(t)$ denotes the state of the automaton at any instant t.
4. $F(.,.) : Q \times B \mapsto Q$ is a mapping in terms of the state and input at the instant t, such that, $q(t+1) = F[q(t), \beta(t)]$. It is called a *transition function*, i.e., a function that determines the state of the automaton at any subsequent time instant $t + 1$. This mapping can either be deterministic or stochastic.
5. $G(.)$: is a mapping $G : Q \mapsto A$, and is called the *output function*. G determines the action taken by the automaton if it is in a given state as: $\alpha(t) = G[q(t)]$. With no loss of generality, G is deterministic.

If the sets Q, B and A are all finite, the automaton is said to be *finite*.

The Environment, E, typically, refers to the medium in which the automaton functions. The Environment possesses all the external factors that affect the actions of the automaton. Mathematically, an Environment can be abstracted by a triple $\langle A, C, B\rangle$. A, C, and B are defined as follows:

1. $A = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is the set of actions.

2. $B = \{\beta_1, \beta_2, \ldots, \beta_m\}$ is the is the output set of the Environment. Again, we consider the case when $m = 2$, i.e., with $\beta = 0$ representing a "Reward", and $\beta = 1$ representing a "Penalty".
3. $C = \{c_1, c_2, \ldots, c_r\}$ is a set of penalty probabilities, where element $c_i \in C$ corresponds to an input action α_i.

The process of learning is based on a learning loop involving the two entities: the Random Environment (RE), and the LA, as described in Figure 2. In the learning process, the LA continuously interacts with the Environment to process responses to its various actions (i.e., its choices). Finally, through sufficient interactions, the LA attempts to learn the optimal action offered by the RE. The actual process of learning is represented as a set of interactions between the RE and the LA.

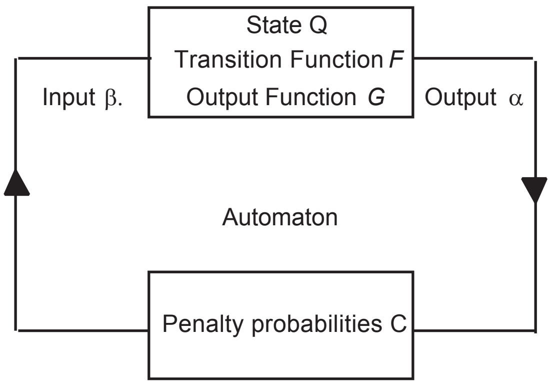

Fig. 2 Feedback Loop of LA.

The automaton is offered a set of actions, and it is constrained to choosing one of them. When an action is chosen, the Environment gives out a response $\beta(t)$ at a time "t". The automaton is either penalized or rewarded with an unknown probability c_i or $1 - c_i$, respectively. On the basis of the response $\beta(t)$, the state of the automaton $\phi(t)$ is updated and a new action is chosen at (t+1). The penalty probability c_i satisfies:

$$c_i = Pr[\beta(t) = 1 | \alpha(t) = \alpha_i] \qquad (i = 1, 2, \ldots, R).$$

We now provide a few important definitions used in the field. $P(t)$ is referred to as the action probability vector, where, $P(t) = [p_1(t), p_2(t), \ldots, p_r(t)]^T$, in which each element of the vector.

$$p_i(t) = Pr[\alpha(t) = \alpha_i],\ i = 1, \ldots, r,\ \text{such that} \sum_{i=1}^{r} p_i(t) = 1\ \ \forall t. \tag{5}$$

Given an action probability vector, $P(t)$ at time t, the *average penalty* is:

$$\begin{aligned} M(t) &= E[\beta(t)|P(t)] = Pr[\beta(t) = 1|P(t)] \\ &= \sum_{i=1}^{r} Pr[\beta(t) = 1|\alpha(t) = \alpha_i]\ Pr[\alpha(t) = \alpha_i] \\ &= \sum_{i=1}^{r} c_i p_i(t). \end{aligned} \tag{6}$$

The average penalty for the "pure-chance" automaton is given by:

$$M_0 = \frac{1}{r}\sum_{i=1}^{r} c_i. \tag{7}$$

As $t \mapsto \infty$, if the average penalty $M(t) < M_0$, at least asymptotically, the automaton is generally considered to be better than the pure-chance automaton. $E[M(t)]$ is given by:

$$E[M(t)] = E\{E[\beta(t)|P(t)]\} = E[\beta(t)]. \tag{8}$$

A LA that performs better than by pure-chance is said to be *expedient*.

Definition 2: A LA is considered *expedient* if:

$$\lim_{t\mapsto\infty} E[M(t)] < M_0.$$

Definition 3: A LA is said to be *absolutely expedient* if $E[M(t+1)|P(t)] < M(t)$, implying that $E[M(t+1)] < E[M(t)]$.

Definition 4: A LA is considered *optimal* if $\lim_{t\mapsto\infty} E[M(t)] = c_l$,where $c_l = \min_i\{c_i\}$.

It should be noted that no optimal LA exist. Marginally sub-optimal performance, also termed above as ϵ-optimal performance, is what LA researchers attempt to attain. **Definition 5:** A LA is considered *ϵ-optimal* if:

$$\lim_{n\mapsto\infty} E[M(t)] < c_l + \epsilon, \tag{9}$$

where $\epsilon > 0$, and can be arbitrarily small, by a suitable choice of some parameter of the LA.

5 PolyPursuit-LA

This section presents our approach to two-class classification by introducing PolyPursuit-LA. For the training phase, it maps the classification problem to a combinatorial optimisation problem over the set of all different polygons in a grid system and by formally specifying an appropriate cost function that encircles one class. Thus, PolyPursuit-LA trains the classifier by defining a

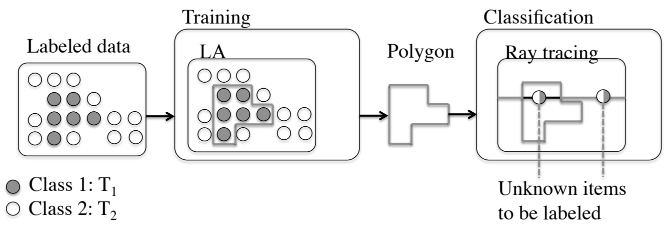

Fig. 3 Overview of approach applied to a simple two-class classification problem.

polygon s. Subsequently, it uses s with ray casting in order to discover if an item is part of the s.

Figure 3 presents an overview of the approach in the case of a simple two-class classification problem. The data is separated using a team of distributed LA yielding a polygon. Next, the polygon is used in the classification with ray casting. In this example, the first item to be labeled will be classified as a T_1 ("Class 1") since it is shown to lie inside the polygon, while the second item will be classified as T_2 ("Class 2") since it lies outside the polygon.

In order to use a team of distributed LA for encircling points into polygons, we resort to a cost function that measures the quality of PolyPursuit-LA solution. In order to discover whether or not a point lies within a polygon we use ray casting.

5.1 Distributed LA

At each epoch, a polygon is chosen randomly according to a distribution over a set of possible paths. The polygon represents a self enclosing path where the source coincides with the destination. The observed performance (classification accuracy) is used to reinforce the polygon by increasing the probability of choosing it again. Since the paths yielding low performance receive weak reinforcement signals, they are chosen less frequently. Thus the scheme can adaptively focus more resources on paths that yield high performance.

Given a grid modeled as a graph $G = (V, E)$, where $V = \{1, ..., m\}$ is the set of nodes in the graph, and E is the set of directed links in the graph. We attach an LA to each node in the graph. The action of each LA attached to a node is the choice of the next hop (neighbor node). Let $N(i)$ be the set of the neighbors of a node i.

The automaton's state probability vector at the node i at time t is $P_i(t) = [p_{(i,j_1)}(t), p_{(i,j_2)}(t) \dots p_{(i,j_{|N(i)|})}(t)]$. where $|N(i)|$ is the number of node i.

Thus, $p_{(i,j)}(t)$ is the probability at time instant t to select the node j as next hop when being in node i.

The normalized feedback function (or reward strength) is given by $f(s(t))$, where $s(t)$ is the path taken at instant t. The function $f(.)$ will be specified in the next section. Informally speaking $f(.)$ measures the fitness of the solution taking values from $[0, 1]$ where 0 is the lowest possible reward while 1 is the highest reward.

Let $s^*(t)$, the optimal path found so far, i.e, the path with highest accuracy obtained up to time instant (t).

The idea of pursuit here is to reward the LA whose actions (edges) lie along $s^*(t)$.

We consider the LA update equations at node i where $i \in s^*(t)$. For all neighbors j, i.e., $j \in N(i)$, the update is given by:

$$p_{(i,j)}(t+1) = (1-\theta_t)\delta_j + \theta_t p_{(i,j)}(t) \tag{10}$$

$$\delta_j = \begin{cases} 1 & \text{if } j \in s^*(t) \\ 0 & \text{else} \end{cases} \tag{11}$$

θ_t is the update parameter and depends on time.

The informed reader would observe that the above update scheme corresponds to the linear-reward inaction LA update.

In fact if $j \notin s^*(t)$ then $p_{(i,j)}(t+1)$ is reduced by multiplying by θ_t which is less than 1

$$p_{(i,j)}(t+1) = \theta_t p_{(i,j)}(t) \tag{12}$$

However if $j \in s^*(t)$ then $p_{(i,j)}(t+1)$ is increased by

$$\begin{aligned} p_{(i,j)}(t+1) - p_{(i,j)}(t) &= [(1-\theta_t) + \theta_t p_{(i,j)}(t)] - p_{(i,j)}(t) & (13)\\ &= (1-\theta_t) + p_{(i,j)}(t)(\theta_t - 1) & (14)\\ &= (1-\theta_t)(1 - p_{(i,j)}(t)) \geq 0 & (15)\end{aligned}$$

In Theorem 1, we will consider the conditions by which the algorithm can converge when the update parameter depends on time. Further, we will give convergence results for the case of fixed θ, i.e, independent of t.

Please note that, initially:
$p_{(i,j)}(0) = \frac{1}{|N(i)|}$, for $j \in N(i)$.

With the updating formula (Equation 10), we can show that the probability distribution Formula converges to the distribution that satisfies the following property if the optimal polygon is unique.

For $i \in s^*$

$$p_{ij} = \begin{cases} 1 & \text{if } j \in s^* \\ 0 & \text{else} \end{cases} \tag{16}$$

The update scheme is called pursuit LA and has rules that obey LRI. The idea is to always reward the transitions probabilities along the best solution obtained so far.

Intuition behind PolyPursuit-LA Thathachar and Sastry [85] pioneered the idea of pursuit LA. The action with the highest reward estimate is "pursued". The latter work has fueled a great deal of interest in pursuit LA involving different variants [4,86,87]. A common feature for all these pursuit algorithms is to estimate the reward probability of each action and pursue the action with the highest reward. In our current work, the environment is rather deterministic. Therefore, we opt to pursue the action of the LA along the path that leads to the highest accuracy.

We will now state some theoretical results that catalogue the properties of the PolyPursuit-LA for both the time varying update parameter and the fixed update parameter.

Theorem 1 *The optimal solution is generated with probability* 1 *only if the update parameter θ_t obeys the following condition:*

$$\sum_{t=1}^{\infty}\prod_{k=1}^{t-1}\theta_k = \infty \tag{17}$$

Proof The proof follows similar arguments as in [88]. Using recurrence, we can obtain a lower bound on $p_{(i,j)}(t)$:

$$p_{(i,j)}(t) \geq \prod_{k=1}^{t-1}\theta_k p_{(i,j)}(0) \tag{18}$$

For all arcs (i,j), by initialization, $p_{(i,j)}(0) = 1/|N(i)|$, where $N(i)$ is the set of neighbors of node i. Let $p_{min}(0)$, the smallest value for all arcs (i,j), it is clear by way of construction of the grid $p_{min}(0) = 1/4$ since the max of neighbors is 4 for nodes in the grid not close to the border.

Let $A_t = \{s(t) \neq s^*\}$ the event that at iteration t, the candidate solution does not contain the optimal solution.

Let B_T the event that optimal solution is not found up to instant T.

$$P(B_T) = \prod_{t=1}^{T} P(A_t) \tag{19}$$

$$P(B_T) \leq \prod_{t=1}^{T}(1 - \prod_{k=1}^{t-1}(\theta_k p_{min}(0))^{|s^*|}) \tag{20}$$

By resorting to $(1-u) \leq exp(-u)$ we obtain

$$P(B_\infty) \leq \prod_{t=1}^{\infty}(1 - \prod_{k=1}^{t-1}\theta_k) \tag{21}$$

$$\leq \prod_{t=1}^{\infty} exp(- \prod_{k=1}^{t-1}(\theta_k p_{min}(0))^{|s^*|}) \tag{22}$$

$$= exp(- \sum_{t=1}^{\infty}\prod_{k=1}^{t-1}\theta_k^{|s^*|} p_{min}(0)^{|s^*|}) \tag{23}$$

However, from our assumption

$$\sum_{t=1}^{\infty}\prod_{k=1}^{t-1}\theta_k = \infty$$

Since we have

$$P(B_\infty) \leq 0 \tag{24}$$

then

$$P(B_\infty) = P(s^*\text{never obtained}) = 0 \tag{25}$$

Examples of smoothing sequences which eventually generate the optimal solution with probability 1 (that is, which satisfy the sufficient condition of Theorem 1) includes
$\theta_t = 1 - 1/(t+1)^\beta$ for $\beta > 1$.
and
$\theta_t = 1 - \frac{1}{(t+1)log(t+1)^\beta}$ for $\beta > 1$.
Let t^* the first time instant when the optimal solution is found, the optimal arcs are always reinforced. For $t^* + r$, for (i^*, j) such that $i^* \in s^*$ and $j \notin s^*$, we have:
Using recurrence, we can verify:

$$p_{(i^*,j)}(t^* + r) = \prod_{k=t^*}^{t^*+r-1}\theta_k p_{(i^*,j^*)}(t) \tag{26}$$

Easy to see from the assumption that $\prod_{k=0}^{\infty}\theta_k = 0$, by considering the log of the expression described in assumption on θ_k.

$$\lim_{r\to\infty} p_{(i^*,j)}(t^* + r) = 0 \tag{27}$$

By considering summation to 1 of probability of going from node i^*, and for j^* belonging to the optimal path.

$$\lim_{r\to\infty} p_{(i^*,j^*)}(t^* + r) = 1 \tag{28}$$

5.2 Constant Update Parameter

In Theorem 1, we give the convergence result of the PolyPursuit-LA for the case of fixed parameter θ that is independent of time.

Theorem 2 *The optimal solution is generated with probability* 1 *only if the update parameter* $\theta \to 1$.

Proof Using recurrence, we know that:

$$p_{(i,j)}(t) > \theta^{t-1} p_{(i,j)}(0) \tag{29}$$

Thus,

$$P(A_t) \leq 1 - (\theta^{t-1} p_{min}(0))^{|s^*|} \tag{30}$$

Therefore,

$$P(B_T) \leq \prod_{t=1}^{T} \left(1 - (\theta^{t-1} p_{min}(0))^{|s^*|}\right) \tag{31}$$

Thus, using again $(1-u) \leq exp(-u)$ we obtain

$$P(s^* \text{never obtained}) = Prob(B_\infty) \leq \prod_{t=1}^{\infty} exp(-\theta^{(t-1)|s^*|} p_{min}(0)^{|s^*|}) \tag{32}$$

$$\leq exp(-p_{min}(0)^{|s^*|} \sum_{t=1}^{\infty} \theta^{(t-1)|s^*|}) \tag{33}$$

$$= exp(-p_{min}(0)^{|s^*|} \sum_{t=0}^{\infty} \theta^{t|s^*|}) \tag{34}$$

Let us define $h(\alpha) = \sum_{t=0}^{\infty} \theta^{t|s^*|}$.

$$h(\alpha) = \sum_{t=0}^{\infty} \theta^{t|s^*|} \tag{35}$$

$$= 1/(1 - \theta^{|s^*|}) \tag{36}$$

$$\lim_{T\to\infty} P(B_T) = P(s^* \text{ never obtained}) \tag{37}$$

$$\leq exp(-p_{min}(0)^{|s^*|} h(\theta)) \tag{38}$$

Since we know that $\lim_{\theta\to 1} h(\theta) = \infty$, then $\lim_{T\to\infty} P(B_T)$ can be made arbitrarily close to zero, if θ approaches 1.

Hence the theorem is proven. Now, let us characterize the LA probabilities at convergence.

Let t^* the first time instant when the optimal solution is found, the optimal arcs are always reinforced. For $t^* + r$, for $(i^*, j^*) \in s^*$, we have:

Using recurrence , we can obtain verify

$$p_{(i^*,j^*)}(t+r) = \theta^r p_{(i^*,j^*)}(t) + (1-\theta)\sum_{i=0}^{r-1}\theta^{r-i-1} \tag{39}$$

We remark

$$\lim_{r\to\infty}\sum_{i=0}^{r-1}\theta^{r-i-1} = 1/(1-\theta) \tag{40}$$

Therefore,

$$\lim_{r\to\infty} p_{(i^*,j^*)}(t+r) = (1-\theta)\times 1/(1-\theta) = 1 \tag{41}$$

5.3 Training phase

The classifier is trained using a guided walk with the team of distributed LA optimising for the score function $f(s)$ in order to create a polygon. Since actions lying over the best solution so far will get reinforced, the actions of the team of LA will converge towards a polygon that is a good separator. This polygon is the key to the classification. Hence, the pheromones on the path are deposited directly in accordance with a score function over the length of the path.

Note that the classifier, implicitly, performs optimisation according to two properties of the data: the score function $f(s)$.

The classifier can therefore be considered as a multi-edged polygon with only vertical or horizontal edges.

In order to ease the understanding of the training process, we shall give an example. Suppose we have a four by four grid containing 16 nodes as in Figure 4. To each node in the grid, we attach an LA. For instance to node number 1, we attach LA whose action probability vector is $P_1(t) = [p_{(1,2)}(t), p_{(1,5)}(t)]$. Similarly for example for nodes 6, we attach an LA whose action probability vector is $P_6(t) = [p_{(6,2)}(t), p_{(6,7)}(t), p_{(6,10)}(t), p_{(6,5)}(t)]$. We suppose that the search for separator starts at node 1. For the sake of clarity we use the notation LA_i to denote the LA attached to node i. We suppose that at time instant t, LA_1 chooses the edge $(1,2)$. Then LA_2 gets activated and we suppose it chooses $(2,6)$. We suppose that the process continues where the following edges get chosen sequentially $(6,7)$, $(7,11)$,$(11,10)$, $(10,9)$, $(9,5)$ and $(5,1)$. Clearly, the resulting polygon has an accuracy of 100% since it perfectly separates the red and blue classes. In the subsequent time instants, we shall pursue the actions that lie along this optimal polygon since it has the highest accuracy: namely action $(1,2)$ corresponding to LA_1 sees its probability increasing while the other action $(1,5)$ sees its probability decreasing. Similarly, we pursue action $(2,1)$ corresponding to LA_2 and the rest of the actions along the optimal polygon until action $(5,1)$ corresponding to LA_5.

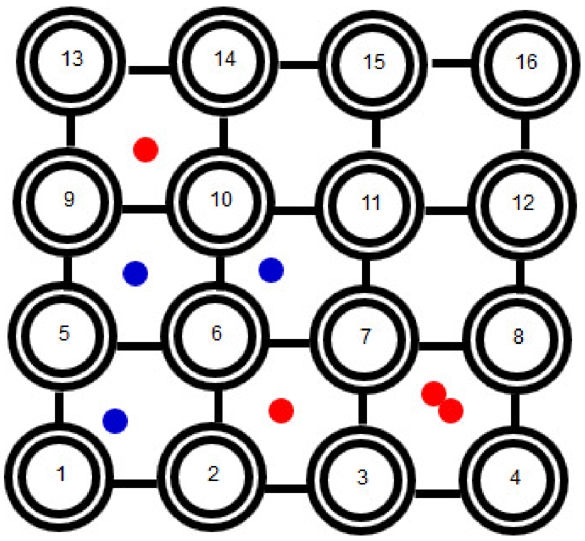

Fig. 4 Illustration of the PolyPursuit-LA

5.4 Boostraping the source node

A detail worth mentioning is the way by which we choose the source node of the polygon. The performance of the scheme is dependent on the right choice of the source vertex for the polygon. Therefore, in order to deal with this disadvantage we allow the scheme to readjust it choice of the source vertice. Whenever a polygon gives a better performance compared to previous iterations, we choose a random node among the nodes part of the best known polygon as the source node. Please note that when probabilities have converged, our experience is that as long as the source node is part of the best polygon, the choice of source node is of little importance.

More advanced methods can be used and verified empirically. However, we found that this simple strategy resulted in good performance.

6 Experiments

The experiments are carried out as traditional supervised learning approaches in two phases: training and classification. The behaviour of the algorithm can best be explained by examining how it behaves on the training data. Because of this factor, the figures depict a visualisation of the polygon on the training data — yielding a good overview of the algorithm behaviour.

The data is generated by various functions intended to show the performance of PolyPursuit-LA using various data. The data generation is explained in each chapter. In each experiment 2000 data points are generated [3],

[3] The number of samples has negligible effect on the performance of the scheme.

of which half are randomly selected for training and the rest used for classification. Further, the data always contains two classes; the blue T_1 class and the red T_2 class.

The granularity of the grids are always chosen as 10x10 and the $\theta = 0.99$.

6.1 Simple environments

We present a simple experimental settings as proof of concept of PolyPursuit-LA. This section empirically shows that the approach works in a simple environment with two easily separable sets of data. The data is composed of two blocks of data: T_1 and T_2. Figure 1 shows the LA convergence after the training phase in this environment. The LA have built a rectangular polygon encircling all items in T_1, but none of the items in T_2. Since this is a polygon that perfectly separates the classes, it yields $f(s) = 1$. The polygon solution in this example is quite straight forward.

In this simple proof of concept, PolyPursuit-LA gave an accuracy of 100%.

6.2 Gaussian

Figure 5 depicts the classification polygon found by the distributed LA for data generated from two different Gaussian distributions. This experiment is interesting because, in contrast to the proof of concept, no classifier will be able to give a perfect result and is therefore a good test for PolyPursuit-LA.

Regarding the classification, the PolyPursuit-LA classifier gave an accuracy of 0.832. For comparison purposes, linear and polynomial SVM produce the same data accuracies, 0.837 and 0.842 respectively.

These high numbers indicate that PolyPursuit-LA is able to perform in line with SVM even when data overlap — without loss of precision.

6.2.1 Semi-circles

Figure 6 shows the scheme in a more complex scenario with semi-circles (or half moons) where there are no clear separation boundaries. It is an interesting experiment because there exists no linear or polynomial solution that can result in a perfect classifier without mapping to multiple dimensions or depending on a kernel trick.

Despite the added complexity, the PolyPursuit-LA approach works perfectly and surrounds the data from the blue class without including the read. In fact, in the classification phase it gives an accuracy, precision and recall of 1. For comparison purposes, linear and polynomial SVM produce accuracies of 0.912 and 0.997 on the same data.

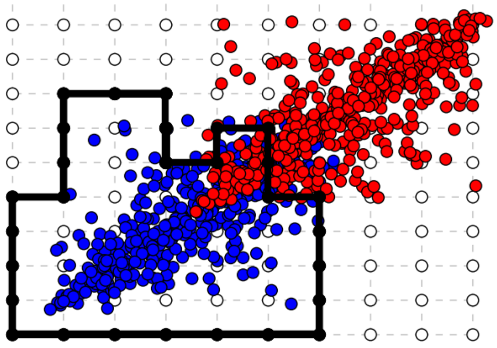

Fig. 5 Gaussian with 0% noise

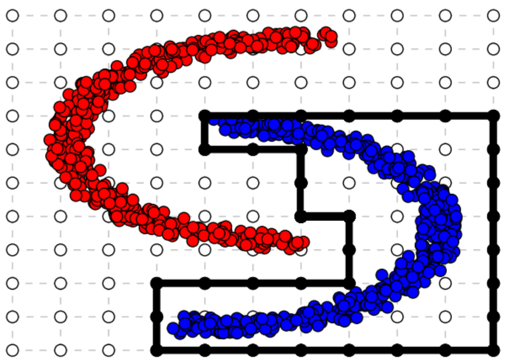

Fig. 6 Half-moon

6.2.2 circles

Figure 7 illustrates the case of non-linear classification boundaries in the form of a circle.

The circular data are interesting to test because no mathematical function can separate the two classes in two dimensions. Any line separating the two classes will need to have multiple y-values for several of the x-values.

Despite the added complexity, the PolyPursuit-LA approach works perfectly by surrounding the data from the blue class without including the red.

To repeat, the accuracy, precision and recall for PolyPursuit-LA is 1, while for linear and polynomial SVM gives accuracies of 0.538 and 0.892 respectively. For SVM to come up to the performance of PolyPursuit-LA, we need to rely on a RBF kernel.

In Figure 7, we increase the noise from 5% up to 50%. Noise means simply that some points of one class are overlapping with the other class making impossible to separate between these overlapping points. When the noise is 50%, half of the data is in the wrong class making the classification an impossible task.

Despite the added noise, the scheme perform well. We would expect an approximate 2.5% loss in accuracy because of the 5% noise. Our empirical results results confirms this giving an accuracy of 0.949. For comparison purposes, SVM had no chance performance by adding noise.

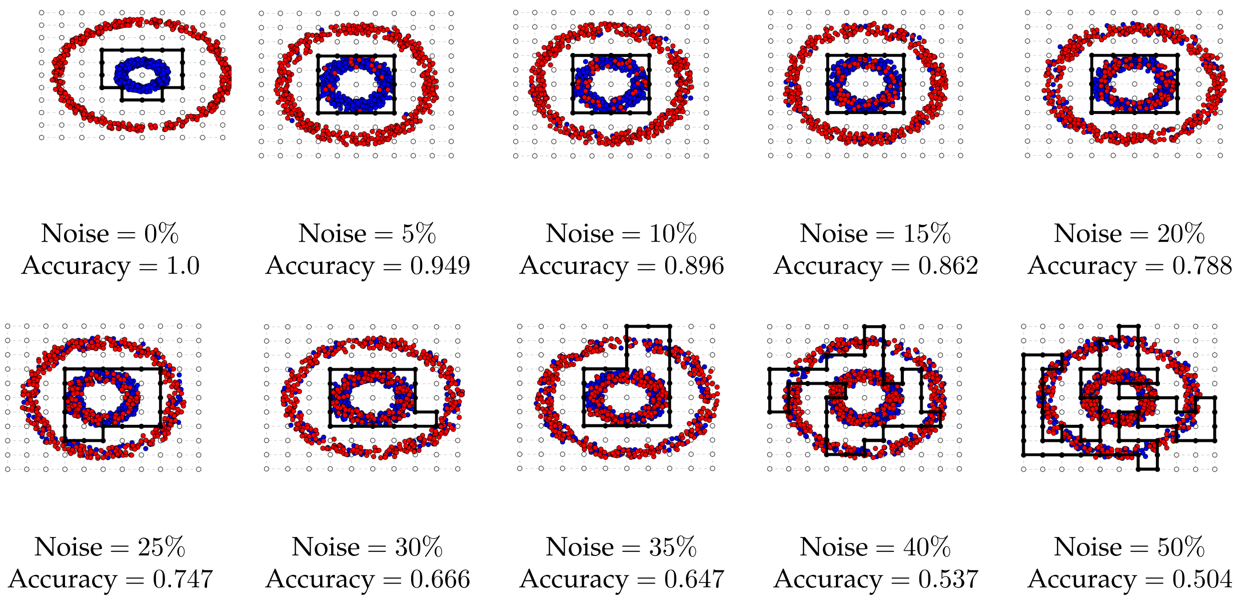

Fig. 7 Example of best known polygon of circular classes and corresponding classification accuracy with increased noise

6.2.3 Gaussian blobs

Figure 8 depicts the case of pairwise Gaussian distributions with the distance between the μ varying from 200 down to 0. The standard deviation is the same for all distributions, $\sigma = 34.1$. This means that when the distance between the μ from both distributions is 200, there is no overlap between the

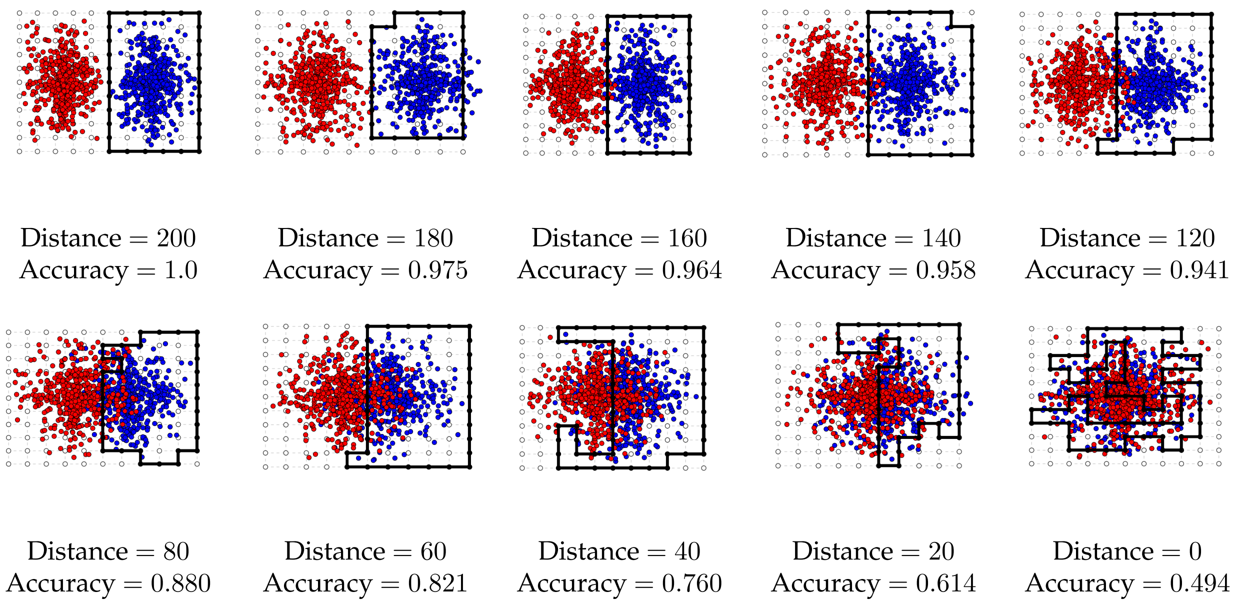

Fig. 8 Example of best known polygon of Gaussian distributed blob classes and corresponding classification accuracy where the distance between the distributions μ becomes increasingly closer together,

distributions. As the distance is decreasing, more and more data is overlapping between the distributions until a distance of 0 where the two μs are the same.

These are interesting results because they explain the behaviour of PolyPursuit-LA when the data increasingly overlap, and in turn becoming increasingly difficult to separate.

When the distance is 200, the data is not at all overlapping yielding an accuracy of 1.0. By moving the two μs closer together, with a distance of 180 the blobs are barely overlapping and PolyPursuit-LA yields in the classification phase an accuracy of 0.975.

With a distance of 120, the data overlaps slightly more. However, PolyPursuit-LA shows barely any drop in performance. It is still able to accurately separate the data and yields in the classification phase an accuracy of 0.941.

When the distance is 0 in Figure 8, the μ from the two Gaussian distributions are the same and the data is completely overlapping and in turn indistinguishable from each other. In this scenario, PolyPursuit-LA reaches an accuracy of 0.494, similar to random guessing.

This indicates that PolyPursuit-LA is able to accurately classify data, even when the data is overlapping and hard to distinguish. Only when the two classes are completely overlapping will PolyPursuit-LA come to short.

6.2.4 Real datasets

We have applied PolyPursuit-LA for two real datasets, Iris and Wine. Figure 9 and Figure 10 show the performance of PolyPursuit-LA respectively over time for the Iris data set and the Wine data set. Please note that the synthetic data sets contain 2000 samples while the Wine quality data set contains 6499. From both figures we observe that PolyPursuit-L coverges quite fast in around 4000 iterations.

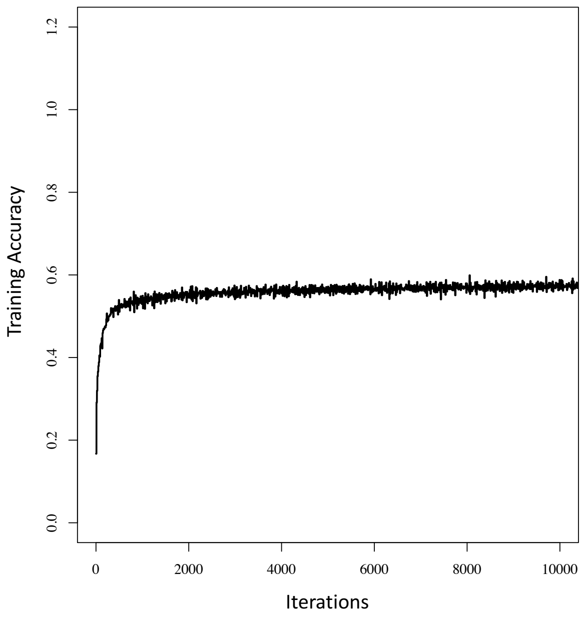

Fig. 9 Training accuracy PolyLA- Pursuit for the Iris data set.

Table 1 summarizes all the results for all the experiences where we compare to Linear SVM, Gaussian SVM, CALA Random Forest, Nearest Neighbour and Multilayer Perceptron. We performed experimental comparison and highlighted the differences and the approach proposed in [89], which is denoted here Continuous Action LA (CALA). There are two major differences between our work and [89]. First, the referred work focuses on linear discriminant. Our approach can find arbitrarily discriminants in the form of polygon that are not necessarily linear. In this sense, from a design perspective the work involving CALA focuses on finding the optimal parameters for the discriminant while our results involve mapping the problem to a search on grid and finding an ″optimal path″. Second, the environment in referred work is stochastic, in a sense, the true label of the classes is noisy. The introduction of a noise model is an argument for using CALA since typical LA operate in

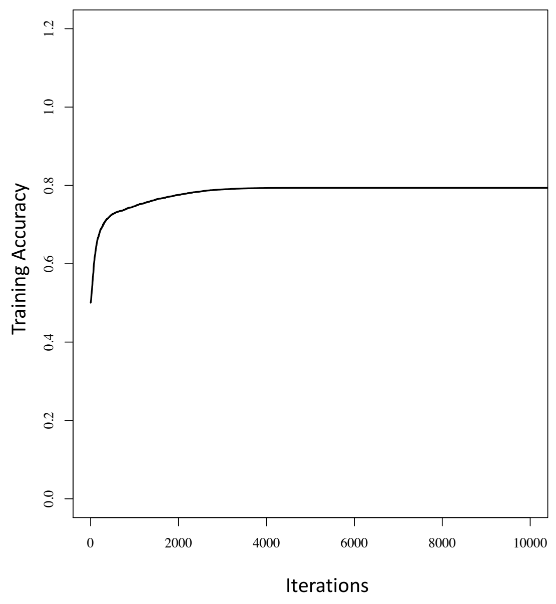

Fig. 10 Training accuracy PolyLA- Pursuit for the Wine data set

random environments. In our problem, the environment is deterministic as the identity of the label is not noisy. As aforementioned, we emphasize that PolyLA-Pursuit is designed for deterministic environment and therefore we believe that it is more suited for the classification problems where the identity of the labels are deterministic. From Table 1 we observe that PolyLA-Pursuit is the only scheme that performs consistently well in almost all the scenarios. In fact, the PolyLA-Pursuit outperforms all the other schemes in the circular case almost for all the noise levels. This supremacy is clear if for example we consider the circular noise 5% where the PolyLA-Pursuit achieves 0.949 accuracy while the second most performant algorithm, which is in this case Multilayer Perceptron, achieves only 0.81 accuracy. Based on this observation, it seems that PolyLA-Pursuit is better at handling non-linear discriminant functions in two-dimensional space than the rest of the compared algorithms. For the Gaussian blob scenarios, the PolyLA-Pursuit and the Random first algorithm perform almost equally well. For example, for Gaussian blob distance 140, the PolyLA-Pursuit yields 0.978 accuracy while the second best algorithm, which is in this case Random Forest, achieves 0.975 accuracy. The comparison results are promising and demonstrate the potential of PolyLA-Pursuit.

Table 2 shows the comparison in accuracy between PolyLA-Pursuit and a recently published ACO variant [3]. We see that PolyACO+ [3] and PolyLA-

Scenario	PolyLA-Pursuit	Linear SVM	Gaussian SVM	CALA	Random Forest	Nearest Neighbour	Multilayer Perceptron
Proof of concept	1.0 (1.0/1.0)	1.0(1.0/1.0)	1.0(1.0/1.0)	0.459(0.918/0.478)	1.0(1.0/1.0)	1.0(1.0/1.0)	1.0(1.0/1.0)
Gaussian distributions	0.832 (0.813/0.862)	0.851(0.836/0.872)	0.784(0.769/0.810)	0.562(1.0/0.533)	0.825(0.823/0.827)	0.844(0.842/0.846)	0.81(0.800/0.826)
Semi-circles	1.0 (1.0/1.0)	0.897(0.899/0.894)	0.997(1.0/0.994)	0.5(1.0/0.5)	0.951(0.932/0.972)	1.0(1.0/1.0)	0.907(1.0/0.814)
Circular 0% noise	1.0 (1.0/1.0)	0.611(0.579/0.808)	0.983(1.0/0.966)	0.5(1.0/0.5)	0.93(0.882/0.992)	1.0(1.0/1.0)	0.481(0.490/0.962)
Circular 5% noise	0.949 (0.948/0.950)	0.59(0.563/0.798)	0.909(0.914/0.902)	0.5(1.0/0.5)	0.522(0.522/0.51)	0.525(0.5256/0.51)	0.81(1.0/0.62)
Circular 10% noise	0.896 (0.882/0.914)	0.541(0.538/0.568)	0.827(0.839/0.808)	0.5(1.0/0.5)	0.885(0.896/0.87)	0.886(0.881/0.892)	0.469(0.484/0.938)
Circular 15% noise	0.862 (0.859/0.866)	0.513(0.511/0.572)	0.820(0.837/0.794)	0.5(1.0/0.5)	0.844(0.848/0.838)	0.837(0.841/0.83)	0.42(0.456/0.84)
Circular 20% noise	0.788 (0.785/0.792)	0.513(0.512/0.522)	0.713(0.726/0.684)	0.5(1.0/0.5)	0.785(0.748/0.858)	0.731(0.725/0.744)	0.919(1.0/0.838)
Circular 25% noise	0.747 (0.743/0.754)	0.518(0.517/0.546)	0.661(0.660/0.664)	0.5(1.0/0.5)	0.701(0.692/0.724)	0.686(0.6905/0.674)	0.427(0.460/0.854)
Circular 30% noise	0.666 (0.649/0.72)	0.481(0.481/0.498)	0.621(0.637/0.56)	0.5(1.0/0.5)	0.67(0.672/0.664)	0.594(0.600/0.562)	0.395(0.441/0.79)
Circular 35% noise	0.647 (0.645/0.652)	0.489(0.489/0.494)	0.593(0.605/0.534)	0.5(1.0/0.5)	0.627(0.617/0.666)	0.555(0.557/0.534)	0.359(0.417/0.718)
Circular 40% noise	0.537 (0.530/0.932)	0.518(0.517/0.524)	0.512(0.513/0.466)	0.5(1.0/0.5)	0.565(0.546/0.77)	0.542(0.542/0.536)	0.342(0.406/0.684)
Circular 50% noise	0.504 (0.504/0.438)	0.493(0.493/0.508)	0.5(0.5/0.518)	0.5(1.0/0.5)	0.503(0.502/0.65)	0.496(0.496/0.518)	0.199(0.284/0.398)
Gaussian blob distance 200	1.0 (1.0/1.0)	1.0(1.0/1.0)	0.805(1.0/0.61)	0.495(0.974/0.497)	1.0(1.0/1.0)	1.0(1.0/1.0)	0.5(0.5/1.0)
Gaussian blob distance 180	0.975 (0.997/0.952)	0.997(0.996/0.998)	0.805(0.719/1.0)	0.514(0.996/0.507)	0.996(0.996/0.996)	0.994(0.992/0.996)	0.498(0.498/0.996)
Gaussian blob distance 160	0.964 (0.979/0.948)	0.987(0.989/0.984)	0.79(0.996/0.582)	0.504(1.0/0.502)	0.995(0.994/0.996)	0.982(0.985/0.978)	0.49(0.494/0.98)
Gaussian blob distance 140	0.978 (0.979/0.976)	0.974(0.968/0.98)	0.820(0.740/0.984)	0.512(1.0/0.506)	0.975(0.962/0.988)	0.958(0.952/0.964)	0.464(0.481/0.928)
Gaussian blob distance 120	0.941 (0.949/0.932)	0.957(0.948/0.966)	0.774(0.696/0.97)	0.508(1.0/0.504)	0.949(0.953/0.944)	0.949(0.949/0.948)	0.464(0.4813/0.928)
Gaussian blob distance 80	0.880 (0.875/0.886)	0.877(0.880/0.872)	0.721(0.822/0.564)	0.505(1.0/0.502)	0.897(0.8997/0.894)	0.867(0.889/0.838)	0.448(0.472/0.896)
Gaussian blob distance 60	0.821 (0.811/0.836)	0.822(0.827/0.814)	0.643(0.605/0.820)	0.508(1.0/0.504)	0.822(0.825/0.816)	0.795(0.790/0.802)	0.895(1.0/0.79)
Gaussian blob distance 40	0.760 (0.757/0.764)	0.751(0.748/0.756)	0.590(0.637/0.418)	0.52(1.0/0.510)	0.754(0.764/0.734)	0.699(0.707/0.678)	0.336(0.401/0.672)
Gaussian blob distance 20	0.614 (0.608/0.638)	0.648(0.640/0.676)	0.558(0.545/0.696)	0.509(1.0/0.504)	0.636(0.623/0.686)	0.572(0.574/0.554)	0.681(1.0/0.362)
Gaussian blob distance 0	0.494 (0.494/0.514)	0.493(0.491/0.398)	0.496(0.497/0.682)	0.507(1.0/0.503)	0.49(0.488/0.418)	0.521(0.521/0.51)	0.419(0.455/0.838)
Iris	0.674 (0.627/0.888)	0.98(1.0/0.96)	0.98(1.0/0.96)	0.5(1.0/0.5)	0.96(0.925/1.0)	1.0(1.0/1.0)	0.5(0.5/1.0)
Wine	0.674 (0.567/0.590)	0.687(0.589/0.577)	0.673(0.609/0.384)	0.378(1.0/0.378)	0.662(0.584/0.375)	0.664(0.560/0.522)	0.351(0.360/0.927)

Table 1 Summary of Accuracy of PolyPursuit-LA Performance. Precision and Recall in parenthesis.

Scenario	PolyLA-Pursuit	PolyACO+
Proof of concept	1.0	1.0
Gaussian distributions	0.832	0.852
Semi-circles	1.0	1.0
Circular 0% noise	1.0	1.0
Circular 5% noise	0.949	0.948
Iris	0.674	0.920
Wine	0.674	0.993

Table 2 Summary of Accuracy of PolyPursuit-LA Performance compared to PolyACO+.

Pursuit have comparable performance while PolyACO+ seems to outperform in the real-life data sets.

7 Conclusion

In this paper, we propose a novel distributed pursuit LA in for deterministic environments in contrast to the legacy LA that only deal with stochastic environment. We apply the devised LA for solving a classification problem. The essence of our scheme is to search for a separator in the feature space by imposing a learning automata based random walks in a grid system. To each node in the gird we attach an LA, whose actions are the choice of the edges forming the separator. The walk is self-enclosing, i.e, a new random walk is started whenever the walker returns to starting node forming a closed classification path yielding a multi-edged polygon. In our approach, the different LA attached at the different nodes search for a polygon that best encircles and separates each class. Based on the obtained polygons, we perform classification by labelling items encircled by a polygon as part of a class using ray casting function. Indeed, PolyPursuit-LA has appealing properties compared to SVM. In fact, unlike PolyPursuit-LA, the SVM performance is dependent on the right choice of the kernel function (e.g. Linear Kernel, Gaussian Kernel)—which is considered a “black art”. PolyPursuit-LA can find arbitrarily complex separators in feature space.

As a future work, we want to explore other applications of the pursuit LA in the future specially for solving combinatorial problems.

References

1. M. Agache, B. J. Oommen, Generalized pursuit learning schemes: New families of continuous and discretized learning automata, IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics 32 (6) (2002) 738–749.
2. M. A. L. Thathachar, P. S. Sastry, Varieties of learning automata: An overview, IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics 32 (6) (2002) 711–722.
3. M. Goodwin, A. Yazidi, Ant colony optimisation-based classification using two-dimensional polygons, in: International Conference on Swarm Intelligence, Springer, 2016, pp. 53–64.
4. M. Agache, B. J. Oommen, Generalized pursuit learning schemes: New families of continuous and discretized learning automata, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 32 (6) (2002) 738–749.
5. S. Lakshmivarahan, Learning Algorithms Theory and Applications, Springer-Verlag, 1981.
6. K. Najim, A. S. Poznyak, Learning Automata: Theory and Applications, Pergamon Press, Oxford, 1994.
7. K. S. Narendra, M. A. L. Thathachar, Learning Automata: An Introduction, Prentice-Hall, Inc., 1989.
8. M. S. Obaidat, G. I. Papadimitriou, A. S. Pomportsis, Learning automata: Theory, paradigms, and applications, IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics 32 (6) (2002) 706–709.
9. A. S. Poznyak, K. Najim, Learning Automata and Stochastic Optimization, Springer-Verlag, Berlin, 1997.
10. M. A. L. Thathachar, P. S. Sastry, Networks of Learning Automata: Techniques for Online Stochastic Optimization, Kluwer Academic, Boston, 2003.
11. J. Zhang, C. Wang, D. Zang, M. Zhou, Incorporation of optimal computing budget allocation for ordinal optimization into learning automata, IEEE Transactions on Automation Science and Engineering 13 (2) (2016) 1008–1017.
12. M. L. Tsetlin, Automaton Theory and the Modeling of Biological Systems, Academic Press, New York, 1973.
13. S. Misra, B. J. Oommen, GPSPA: A new adaptive algorithm for maintaining shortest path routing trees in stochastic networks, International Journal of Communication Systems 17 (2004) 963–984.
14. M. S. Obaidat, G. I. Papadimitriou, A. S. Pomportsis, H. S. Laskaridis, Learning automata-based bus arbitration for shared-edium ATM switches, IEEE Transactions on Systems, Man, and Cybernetics: Part B 32 (2002) 815–820.
15. B. J. Oommen, T. D. Roberts, Continuous learning automata solutions to the capacity assignment problem, IEEE Transactions on Computers C-49 (2000) 608–620.
16. G. I. Papadimitriou, A. S. Pomportsis, Learning-automata-based TDMA protocols for broadcast communication systems with bursty traffic, IEEE Communication Letters (2000) 107–109.
17. A. F. Atlassis, N. H. Loukas, A. V. Vasilakos, The use of learning algorithms in ATM networks call admission control problem: A methodology, Computer Networks 34 (2000) 341–353.
18. A. F. Atlassis, A. V. Vasilakos, The use of reinforcement learning algorithms in traffic control of high speed networks, Advances in Computational Intelligence and Learning (2002) 353–369.

19. A. V. Vasilakos, M. P. Saltouros, A. F. Atlassis, W. Pedrycz, Optimizing QoS routing in hierarchical ATM networks using computational intelligence techniques, IEEE Transactions on Systems, Man and Cybernetics: Part C 33 (2003) 297–312.
20. F. Seredynski, Distributed scheduling using simple learning machines, European Journal of Operational Research 107 (1998) 401–413.
21. J. Kabudian, M. R. Meybodi, M. M. Homayounpour, Applying continuous action reinforcement learning automata (CARLA) to global training of hidden markov models, in: Proceedings of the International Conference on Information Technology: Coding and Computing , ITCC'04, Las Vegas, Nevada, 2004, pp. 638–642.
22. M. R. Meybodi, H. Beigy, New learning automata based algorithms for adaptation of backpropagation algorithm pararmeters, International Journal of Neural Systems 12 (2002) 45–67.
23. C. Unsal, P. Kachroo, J. S. Bay, Simulation study of multiple intelligent vehicle control using stochastic learning automata, Transactions of the Society for Computer Simulation International 14 (1997) 193–210.
24. B. J. Oommen, E. d. S. Croix, Graph partitioning using learning automata, IEEE Transactions on Computers C-45 (1995) 195–208.
25. J. J. Collins, C. C. Chow, T. T. Imhoff, Aperiodic stochastic resonance in excitable systems, Physical Review E 52 (1995) R3321–R3324.
26. R. L. Cook, Stochastic sampling in computer graphics, ACM Trans. Graph. 5 (1986) 51–72.
27. M. Barzohar, D. B. Cooper, Automatic finding of main roads in aerial images by using geometric-stochastic models and estimation, IEEE Transactions on Pattern Analysis and Machine Intelligence 7 (1996) 707–722.
28. M. L. Brandeau, S. S. Chiu, An overview of representative problems in location research, Management Science 35 (1989) 645–674.
29. C. Bettstetter, H. Hartenstein, X. Pérez-Costa, Stochastic properties of the random waypoint mobility model, Journal Wireless Networks 10 (2004) 555–567.
30. B. S. Rowlingson, P. J. Diggle, SPLANCS: Spatial Point Pattern Analysis Code in S-Plus, University of Lancaster, North West Regional Research Laboratory, 1991.
31. M. Paola, Digital simulation of wind field velocity, Journal of Wind Engineering and Industrial Aerodynamics 74-76 (1998) 91–109.
32. J. P. Cusumano, B. W. Kimble, A stochastic interrogation method for experimental measurements of global dynamics and basin evolution: Application to a two-well oscillator, Nonlinear Dynamics 8 (1995) 213–235.
33. A. Baddeley, R. Turner, Spatstat: An R package for analyzing spatial point patterns, Journal of Statistical Software 12 (2005) 1–42.
34. R. Caruana, A. Niculescu-Mizil, An empirical comparison of supervised learning algorithms, in: Proceedings of the 23rd international conference on Machine learning, ACM, 2006, pp. 161–168.
35. R. Caruana, N. Karampatziakis, A. Yessenalina, An empirical evaluation of supervised learning in high dimensions, in: Proceedings of the 25th international conference on Machine learning, ACM, 2008, pp. 96–103.
36. G. Madjarov, D. Kocev, D. Gjorgjevikj, S. Džeroski, An extensive experimental comparison of methods for multi-label learning, Pattern Recognition 45 (9) (2012) 3084–3104.
37. M. Dorigo, G. Di Caro, Ant colony optimization: a new meta-heuristic, in: Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406), Vol. 2, IEEE, 1999, pp. 1470–1477.
38. B. J. Oommen, M. Agache, Continuous and discretized pursuit learning schemes: Various algorithms and their comparison, IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics 31 (2001) 277–287.
39. S. Misra, B. J. Oommen, Dynamic algorithms for the shortest path routing problem: learning automata-based solutions, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 35 (6) (2005) 1179–1192.
40. S. Misra, B. J. Oommen, An efficient dynamic algorithm for maintaining all-pairs shortest paths in stochastic networks, IEEE Transactions on Computers 55 (6) (2006) 686–702.
41. H. Li, L. Mason, M. Rabbat, Distributed adaptive diverse routing for voice-over-ip in service overlay networks, IEEE Transactions on Network and Service Management 6 (3) (2009) 175–189.
42. L. Mason, An optimal learning algorithm for s-model environments, IEEE Transactions on Automatic Control 18 (5) (1973) 493–496.

43. H. Beigy, M. R. Meybodi, Utilizing distributed learning automata to solve stochastic shortest path problems, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 14 (05) (2006) 591–615.
44. J. A. Torkestani, M. R. Meybodi, An intelligent backbone formation algorithm for wireless ad hoc networks based on distributed learning automata, Computer Networks 54 (5) (2010) 826–843.
45. J. A. Torkestani, M. R. Meybodi, Finding minimum weight connected dominating set in stochastic graph based on learning automata, Information Sciences 200 (2012) 57–77.
46. J. A. Torkestani, M. R. Meybodi, A learning automata-based heuristic algorithm for solving the minimum spanning tree problem in stochastic graphs, The Journal of Supercomputing 59 (2) (2012) 1035–1054.
47. M. A. Thathachar, P. S. Sastry, Learning optimal discriminant functions through a cooperative game of automata, IEEE Transactions on Systems, Man and Cybernetics 17 (1) (1987) 73–85.
48. G. Santharam, P. Sastry, M. Thathachar, Continuous action set learning automata for stochastic optimization, Journal of the Franklin Institute 331 (5) (1994) 607–628.
49. P. Sastry, M. Thathachar, Learning automata algorithms for pattern classification, Sadhana 24 (4) (1999) 261–292.
50. S. Zahiri, Learning automata based classifier, Pattern Recognition Letters 29 (1) (2008) 40–48.
51. X. Zeng, Z. Liu, A learning automata based algorithm for optimization of continuous complex functions, Information Sciences 174 (3) (2005) 165–175.
52. M. Howell, T. Gordon, F. Brandao, Genetic learning automata for function optimization, IEEE Transactions on Systems, Man, and Cybernetics 32 (6) (2002) 804–815.
53. S. Bandyopadhyay, C. A. Murthy, S. K. Pal, Pattern classification with genetic algorithms, Pattern recognition letters 16 (8) (1995) 801–808.
54. T. Stützle, M. López-Ibáñez, M. Dorigo, A concise overview of applications of ant colony optimization, Wiley Encyclopedia of Operations Research and Management Science.
55. M. Dorigo, M. Birattari, T. Stutzle, Ant colony optimization, Computational Intelligence Magazine, IEEE 1 (4) (2006) 28–39.
56. W. Gutjahr, A graph-based ant system and its convergence, Future Generation Computer Systems 16 (8) (2000) 873–888.
57. L. D'Acierno, M. Gallo, B. Montella, An ant colony optimisation algorithm for solving the asymmetric traffic assignment problem, European Journal of Operational Research 217 (2) (2012) 459–469.
58. M. Goodwin, O.-C. Granmo, J. Radianti, P. Sarshar, S. Glimsdal, Ant colony optimisation for planning safe escape routes, in: Recent Trends in Applied Artificial Intelligence, Springer, 2013, pp. 53–62.
59. M. Goodwin, O.-C. Granmo, J. Radianti, Escape planning in realistic fire scenarios with ant colony optimisation, Applied Intelligence 42 (1) (2015) 24–35.
60. T. Desell, S. Clachar, J. Higgins, B. Wild, Evolving deep recurrent neural networks using ant colony optimization, in: Evolutionary Computation in Combinatorial Optimization, Springer, 2015, pp. 86–98.
61. K. M. Salama, A. A. Freitas, Ant colony algorithms for constructing bayesian multi-net classifiers, Intelligent Data Analysis 19 (2) (2015) 233–257.
62. B. Liu, H. Abbas, B. McKay, Classification rule discovery with ant colony optimization, in: IEEE/WIC International Conference on Intelligent Agent Technology, 2003. IAT 2003., IEEE, 2003, pp. 83–88.
63. R. S. Parpinelli, H. S. Lopes, A. Freitas, et al., Data mining with an ant colony optimization algorithm, IEEE Transactions on Evolutionary Computation 6 (4) (2002) 321–332.
64. D. Martens, M. De Backer, R. Haesen, J. Vanthienen, M. Snoeck, B. Baesens, Classification with ant colony optimization, IEEE Transactions on Evolutionary Computation 11 (5) (2007) 651–665.
65. F. E. Otero, A. A. Freitas, C. G. Johnson, cant-miner: an ant colony classification algorithm to cope with continuous attributes, in: Ant colony optimization and swarm intelligence, Springer, 2008, pp. 48–59.
66. I. C. Junior, Data mining with ant colony algorithms, in: Intelligent Computing Theories and Technology, Springer, 2013, pp. 30–38.

67. R. S. Parpinelli, H. S. Lopes, A. A. Freitas, An ant colony based system for data mining: applications to medical data, in: Proceedings of the genetic and evolutionary computation conference (GECCO-2001), Citeseer, 2001, pp. 791–797.
68. S. Hota, P. Satapathy, A. K. Jagadev, Modified ant colony optimization algorithm (mant-miner) for classification rule mining, in: Intelligent Computing, Communication and Devices, Springer, 2015, pp. 267–275.
69. L. Özbakir, A. Baykasoğlu, S. Kulluk, H. Yapıcı, Taco-miner: an ant colony based algorithm for rule extraction from trained neural networks, Expert Systems with Applications 36 (10) (2009) 12295–12305.
70. N. Holden, A. Freitas, Web page classification with an ant colony algorithm, in: Parallel Problem Solving from Nature-PPSN VIII, Springer, 2004, pp. 1092–1102.
71. K. Socha, C. Blum, An ant colony optimization algorithm for continuous optimization: application to feed-forward neural network training, Neural Computing and Applications 16 (3) (2007) 235–247.
72. K. Salama, A. M. Abdelbar, A novel ant colony algorithm for building neural network topologies, in: Swarm Intelligence, Springer, 2014, pp. 1–12.
73. K. M. Salama, A. M. Abdelbar, Learning neural network structures with ant colony algorithms, Swarm Intelligence (2015) 1–37.
74. L. M. De Campos, J. M. Fernandez-Luna, J. A. Gámez, J. M. Puerta, Ant colony optimization for learning bayesian networks, International Journal of Approximate Reasoning 31 (3) (2002) 291–311.
75. L. M. De Campos, J. Puerta, et al., Learning bayesian networks by ant colony optimisation: searching in two different spaces, Mathware & soft computing 9 (3) (2008) 251–268.
76. R. Daly, Q. Shen, S. Aitken, Learning bayesian networks: approaches and issues, The knowledge engineering review 26 (02) (2011) 99–157.
77. J. Jun-Zhong, H.-X. ZHANG, H. Ren-Bing, L. Chun-Nian, A bayesian network learning algorithm based on independence test and ant colony optimization, Acta Automatica Sinica 35 (3) (2009) 281–288.
78. R. Daly, Q. Shen, Learning bayesian network equivalence classes with ant colony optimization, arXiv preprint arXiv:1401.3464.
79. C.-F. Juang, P.-H. Chang, Designing fuzzy-rule-based systems using continuous ant-colony optimization, IEEE Transactions on Fuzzy Systems 18 (1) (2010) 138–149.
80. A. Chan, A. Freitas, A new classification-rule pruning procedure for an ant colony algorithm, in: Artificial Evolution, Springer, 2006, pp. 25–36.
81. S. Arora, S. Singh, An effective hybrid butterfly optimization algorithm with artificial bee colony for numerical optimization, changes 26 (2017) 27.
82. A. O. Restrepo Rodríguez, D. E. Casas Mateus, G. García, P. Alonso, C. E. Montenegro Marín, R. González Crespo, Hyperparameter optimization for image recognition over an ar-sandbox based on convolutional neural networks applying a previous phase of segmentation by color–space, Symmetry 10 (12) (2018) 743.
83. J. Meza, H. Espitia, C. Montenegro, R. G. Crespo, Statistical analysis of a multi-objective optimization algorithm based on a model of particles with vorticity behavior, Soft Computing 20 (9) (2016) 3521–3536.
84. M. Magdin, F. Prikler, Are instructed emotional states suitable for classification? demonstration of how they can significantly influence the classification result in an automated recognition system, IJIMAI 5 (4) (2019) 141–147.
85. M. A. L. Thathachar, P. S. Sastry, A new approach to designing reinforcement schemes for learning automata, IEEE Transactions on Systems, Man, and Cybernetics SMC-15.
86. X. Zhang, O.-C. Granmo, B. J. Oommen, On incorporating the paradigms of discretization and bayesian estimation to create a new family of pursuit learning automata, Applied intelligence 39 (4) (2013) 782–792.
87. B. J. Oommen, J. K. Lanctôt, Discretized pursuit learning automata, IEEE Transactions on Systems, Man, and Cybernetics SMC-20 (4) (1990) 931–938.
88. W. J. Gutjahr, Aco algorithms with guaranteed convergence to the optimal solution, Information processing letters 82 (3) (2002) 145–153.
89. M. A. L. Thathachar, P. S. Sastry, Learning optimal discriminant functions through a cooperative game of automata, IEEE Transactions on Systems, Man and Cybernetics 17 (1) (1987) 73–85.