Pattern Analysis and Applications (2020) 23:1235–1250
https://doi.org/10.1007/s10044-019-00848-6

THEORETICAL ADVANCES

Distributed Learning Automata-based S-learning scheme for classification

Morten Goodwin[2] · Anis Yazidi[1] · Tore Møller Jonassen[1]

Received: 12 January 2019 / Accepted: 10 September 2019 / Published online: 12 October 2019

Abstract
This paper proposes a novel classifier based on the theory of Learning Automata (LA), reckoned to as PolyLA. The essence of our scheme is to search for a separator in the feature space by imposing an LA-based random walk in a grid system. To each node in the grid, we attach an LA whose actions are the choices of the edges forming a separator. The walk is self-enclosing, and a new random walk is started whenever the walker returns to the starting node forming a closed classification path yielding a many-edged polygon. In our approach, the different LA attached to the different nodes search for a polygon that best encircles and separates each class. Based on the obtained polygons, we perform classification by labeling items encircled by a polygon as part of a class using a ray casting function. From a methodological perspective, PolyLA has appealing properties compared to SVM. In fact, unlike PolyLA, 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." PolyLA, on the other hand, can find arbitrarily complex separator in the feature space. We provide sound theoretical results that prove the optimality of the scheme. Furthermore, experimental results show that our scheme is able to perfectly separate both simple and complex patterns outperforming existing classifiers, such as polynomial and linear SVM, without the need to map the problem to many dimensions or to introduce a "kernel trick." We believe that the results are impressive, given the simplicity of PolyLA compared to other approaches such as SVM.

Keywords Classification · Learning Automata · Polygons · Distributed learning

1 Introduction

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 are not easily separable in the feature space. A myriad of classification algorithms have been proposed in the literature with a variety of behaviors and limitations [1–3]. Examples of these algorithms include neural networks, SVM and decision trees.

A broad class of classification algorithms such as SVM and perceptron relies upon defining a mathematical function with weights that can efficiently separate two or more classes of data. The weights are unknown and learned from the training data. These functions are either linear, polynomial, or for more complex patterns, kernels equivalent to mapping the data to a many-dimensional space where the classes are separable by a hyperplane.

However, the main difficulty is to choose the nature of the function or kernel. Often, the "best" hyperplane, or line in two dimensions that separates classes does not follow the mathematical properties of a function. The "best" separator can for example be a polygon encircling certain data points, which is not a function and therefore cannot straightforward be outputted by SVM or similar classifiers. The accuracy of the SVM is dependent on the right 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

The third author passed away on February 04, 2018, and the authors dedicate this manuscript to his memory.

✉ Anis Yazidi
Anis.Yazidi@oslomet.no

[1] Department of Computer Science, Oslo Metropolitan University, Oslo, Norway

[2] Department of Information and Communication Technology, University of Agder, Kristiansand, Norway

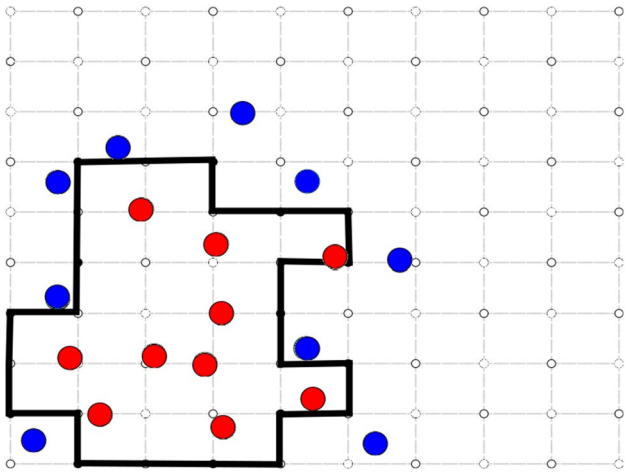

Fig. 1 Example of simple two-class classification scenario with the classes blue (T_1) and red (T_2) (color figure online)

simply because any line separating the data perfectly will have multiple y-values of some 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" where the data can be easily separable.

This paper introduces PolyLA, a novel classification scheme operating in two dimensions[1] using LA and that does not involve a "kernel trick" whenever the data are not easily separable. As in [4], PolyLA deals with the classification problem in a completely different manner from existing classifiers. Instead of relying upon mathematical functions for separating the classes, PolyLA surrounds the classes with polygons guided by reinforced random walk and ray casting. Some of the best known classification techniques, such as support vector machine (SVM) and perceptron-based classifiers, rely upon constructing mathematical functions having weights that efficiently separate two or more classes of data in the feature space. In two-dimensional spaces, the separation boundary might be nonlinear and thus the decision boundaries might be complex. SVM deals with this situation by either projecting the data on a higher-dimensional space or using a kernel trick, which provides a separator not limited to a linear or polynomial function. The adoption of a kernel is equivalent to transposing the data to many dimensions, but the accuracy depends on the right choice of the kernel functions as well as on several other parameters. The latter choice is usually performed through manual trial and error. The presented approach deals with classification problems in two-dimensional Euclidean feature space by building "separator" with many-sided polygons. The polygons are extrapolated from reinforced random walks with a preference toward encapsulating all items from one class and excluding from the encapsulation any items from other classes. In this manner, emerging polygons encapsulates each class in such a way that they can be used as classifiers. The classification takes place by resorting to ray casting of unknown items so that to identify if an item is contained in the polygon. Each item is labeled depending on whether or not it is inside the polygon.

1.1 Outline

The paper is organized as follows. Section 2 introduces the problem that we attempt to solve. Section 3 gives a brief introduction to the theory of LA which is fundamental for our approach named PolyLA. Section 4 reviews relevant state-of-the-art in the area of classifiers as well as related LA-based classifiers. Section 5 continues with introducing our solution: PolyLA as a method for creating polygons for classification with two classes and corresponding results. Section 6 shows empirical results for PolyLA and compares it with comparable algorithms, namely SVM. Finally, in Sect. 7, we draw final conclusions and give insights into future work.

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 encircle 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 which class a new unknown item to be classified belongs to. This is achieved by finding which polygon(s) it is part of.

Further, this paper presents two distinct variants of PolyLA:

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

[1] It is easy to generalize the current model to multi-dimensional by considering pairs of dimensions. In this article, we limit ourselves to the two-dimensional case as a proof of concept. Experiments for the multi-dimensional case can be provided if the requested by the referee.

2.1 The training phase

This training phase can be formulated as a combinatorial optimization problem. The training data, T, consist of multiple classes. The data are 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 $t_i \in T$ lie somewhere between vertices 1 and $N, 1 < t_i < N \forall_{t_i \in T}$.

2.1.1 Two-class classification problem

An example is shown in Fig. 1. In this example, T consists of 19 items, 9 in the blue class T_1 and 10 in the red class T_2. The grid $G(V, E)$ is created so that all items are 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. Using the example from Fig. 1, the task is to find an s that encircles the first training data T_1, but not T_2—a polygon that separates well 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. With two classes, there is only a need for one polygon to perfectly separate the data.

Whether a training element t_i is inside a polygon $s \in \mathbf{S}$ is defined formally as:

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

Ideally, all items in class one (e.g., T_1) should be within the polygon, while all items in the other classes should fall outside the polygon. Any item, t_i from class T_1 that is correctly within the polygon s will yield $h(t_i, s) = 1$, and, similarly, any item, t_j not part of class T_1 and is correctly outside of the polygon s will yield $1 - h(t_j, s) = 1$. For all other items, $h(., s)$ will give 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, that in a flawed way 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 to find a polygon $s*$ for each class, consisting of vertices and edges, that minimizes $f(s*)$. Thus, formally, we aim to find an $s* \in \mathbf{S}$ so that $f(s*) \leq f(s) \in \mathbf{S}$ using an LA-based random walk on the grid as explained in Sect. 3.

2.1.2 Multi-class classification problem

In the case of classification with more than two classes, one polygon is not sufficient to separate all classes. As an example, let us suppose there are three classes: T_1,T_2, and T_3. In simple term, 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:

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

2.2 The classification phase

The classification phase resorts to the polygons from the training phase. The classification task is to find which class a new item with unknown label, t_k, belongs to.

Since the training phase produces one polygon, $s*$, the problem is reduced to simply determining whether a new item is 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 Eq. 1, given two classes T_1 and T_2 and the polygon $s*$, we can define the following decision rules:

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

2.2.1 Multi-class classification problem

The classification phase uses the set of polygons, $\mathbf{s*_{all}}$ (see Eq. 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:

$$\begin{aligned} &t_k \text{ is of class } T_1 && \text{if } h(t_k, s*_{T_1}) = 1 \\ &t_k \text{ is of class } T_2 && \text{if } h(t_k, s*_{T_2}) = 1 \\ &\cdots && . \quad . \\ &t_k \text{ is of class } T_{N-1} && \text{if } h(t_k, s*_{T_{N-1}}) = 1 \\ &t_k \text{ is of class } T_N && \text{otherwise} \end{aligned} \tag{4}$$

[2] $f(s)$ is formally defined in Sect. 5 and Eq. 14.

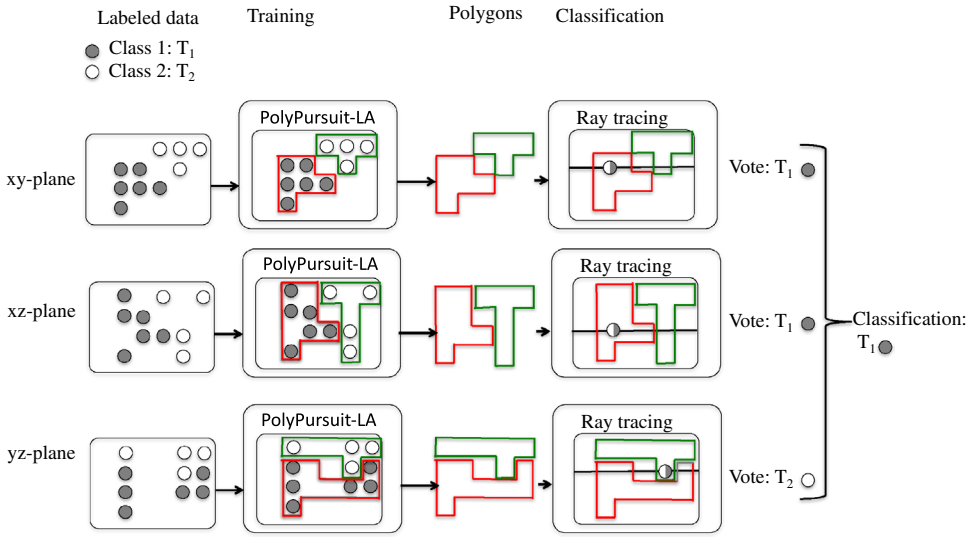

Fig. 2 Overview of training and classification for PolyLA with several features

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 labeled as the class T_N.

2.3 Multi-dimensional classification

It is possible to extend PolyLA to support multiple features by splitting a multi-feature classification problem into several two-dimensional sub-problems which are trained independently. The overall classification is a combination of the results from all sub-problems through a majority voting scheme. More precisely, the overall class prediction is derived by taking the most common class prediction from all the sub-problems, as illustrated in Fig. 2 using the majority vote rule.

In this sense, PolyLA constructs solutions in all the planes that the data set consists of and handles each plane individually. The number of possible planes depends on the number of features in the data set. For example, a three-dimensional feature space with axes x, y and z has three planes xy, xz and yz (see Fig. 2). More generally, the number of planes for n dimensional feature space is simply equal to the number of dimension pairs and is given by: $\binom{n}{2}$. Inevitably, the number of planes explodes as the number of features increases. However, feature selection and reduction methods could be used to deal with this problem.

3 Learning Automata

The fundamental tool which we shall use in most of our research involves Learning Automata (LA). LA have been used in systems that have incomplete knowledge about the Environment in which they operate [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 publicized 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" [5].

With regard to applications, the entire field of LA and stochastic learning has had a myriad of applications [6–8, 10, 11], 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]. Besides 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 analyze 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 analyze spatial point patterns [33]. LA-based schemes have already been utilized to learn the best parameters for neural networks [22], optimizing QoS routing [19], and bus arbitration [14]—to mention a few other applications.

In the field of Automata Theory, an automaton [6–8, 10, 11] 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 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 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.

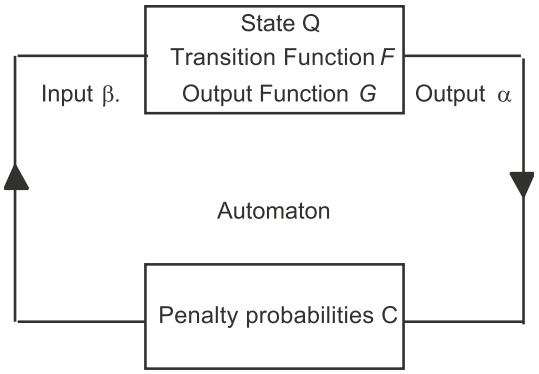

Fig. 3 Feedback loop of LA

The process of learning is based on a learning loop involving the two entities: the random environment (RE), and the LA, as described in Fig. 3. In the process of learning, 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.

The automaton is offered a set of actions, and it is constrained to choose 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:

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

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.

3.1 Classification of Learning Automata

3.1.1 Deterministic Learning Automata

An automaton is termed as a *deterministic automaton*, if both the transition function $F(.,.)$ and the output function $G(.)$ are deterministic. Thus, in a deterministic automaton, the subsequent state and action can be uniquely specified, provided the present state and input are given.

3.1.2 Stochastic Learning Automata

If, however, either the transition function $F(.,.)$, or the output function $G(.)$ is stochastic, the automaton is termed to be a *stochastic automaton*. In such an automaton, if the current state and input are specified, the subsequent states and actions cannot be specified uniquely. In such a case, $F(.,.)$ only provides the probabilities of reaching the various states from a given state.

In the first LA designs, the transition and the output functions were time invariant, and for this reason these LA were considered "Fixed Structure Stochastic Automata" (FSSA). Tsetlin Krylov, and Krinsky [12] presented notable examples of this type of automata.

Later, Vorontsova and Varshavskii introduced a class of stochastic automata known in the literature as Variable Structure Stochastic Automata (VSSA). In the definition of a VSSA, the LA are completely defined by a set of actions (one of which is the output of the automaton), a set of inputs (which is usually the response of the Environment) and a learning algorithm, T. The learning algorithm [8] operates on a vector (called *the Action Probability vector*).

Note that the algorithm $T : [0,1]^R \times \mathrm{A} \times \mathrm{B} \to [0,1]^R$ is an updating scheme where $\mathrm{A} = \{\alpha_1, \alpha_2, ..., \alpha_R\}$, $2 \leq \mathrm{R} < \infty$, is the set of output actions of the automaton, and B is the set of responses from the Environment. Thus, the updating is such that

$$P(t+1) = T(P(t), \alpha(t), \beta(t)),$$

where $P(t)$ is the action probability vector, $\alpha(t)$ is the action chosen at time t, and $\beta(t)$ is the response it has obtained.

If the mapping T is chosen in such a manner that the Markov process has absorbing states, the algorithm is referred to as an absorbing algorithm. Many families of VSSA that posses absorbing barriers have been reported [8]. Ergodic VSSA have also been investigated [8, 34]. These VSSA converge in distribution, and thus, the asymptotic distribution of the action probability vector has a value that is independent of the corresponding initial vector. While ergodic VSSA are suitable for non-stationary environments, absorbing VSSA are preferred in stationary environments.

4 Related work

4.1 Distributed LA on a graph

Misra and Oommen pioneered of the concept of concept of LA on a graph using pursuit LA [13, 35, 36] for solving the

stochastic shortest path problem. Li [37] used a type of *S* LA [38] to find the shortest path in a graph. Beigy and Meybodi [39] provided the first proof in the literature that shows the convergence of distributed LA on a graph for a reward inaction LA. For applications of distributed LA on a graph in the field of computer communications, we refer the reader to the work of Torkestani and his collaborators [40–42].

4.2 LA for classification and function optimization

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.

In general terms, the distinguishing characteristic of LA-based learning is that the search for the optimizing parameter vector is conducted in the space of probability distributions defined over the parameter space, rather than in the parameter space itself [43]. In machine learning, the most common method for building a classifier is to conduct a search over the parameter space using optimization techniques such as gradient descent, while the common and recurrent theme reported in the literature when building a classifier based on LA is to work in a probability space rather than a parameter space. The main advantage of working in a probability space is better resilience to noise. This resilience to noise was demonstrated in [43] where the true label of each data point in the training data is noisy in the sense that it is revealed by an Oracle according to a faulty model. It was demonstrated in some cases, LA performs better than other classical classification algorithms such as feedforward neural networks even with discretized parameter space, and thus a limited number of possible parameters which might reduce the accuracy of the scheme [43]. It is worth mentioning that Continuous Action LA (CALA), in contrast to classical LA, does not discretize the parameter space and rather operates on a continuous parameter space where the choices of the parameter are drawn from a time-varying sampling distribution that is adjusted based on ideas borrowed from the field of reinforcement learning [44].

In [44], another structure of LA algorithms used for classification is presented which possesses a multi-layer representation similar to neural networks. The actions of the first level LA are real-value parameters of the hyperlanes. The second level of LA is Boolean decisions regarding which hyperlanes to be included to create convex sets using an AND operation. The final layer of LA performs an OR operation on the outputs of the second layer units. Therefore, the discriminant is a Boolean expression consisting of linear inequalities [44]. Similar ideas were applied in order to learn the decisions trees classifiers using LA teams [45] where an individual LA can be used to learn the best split rule at a given node.

A closely related work to ours is due to Thathachar and Sastry [46] where the authors use a team of LA in order to find the optimal discriminant function in a feature space. The discriminant functions are parametrized, and an LA are attached to each parameter. The LA team is involved in a cooperative game with common payoff. The general theme is to classify the next pattern with the chosen discriminant function and to either reward or penalize the joint action of to the team depending on whether the classification agrees with the true label or not. Later, Santharam et al. [47] proposed to use continuous LA in order to deal with the disadvantages of discretization, 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 [44] . In [48], 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: PolyLA for handling multi-dimensional classification problems. In [49], the authors have proposed LA optimization methods for multimodal functions. Through experimental settings, the performance of these algorithms was shown to outperform genetic algorithms.

Some improvements of the latter algorithm were introduced in [50] to better remove and regenerate the hypercubes and to better update the LA probabilities which yielded better accuracy.

In [51], the authors introduce a combination of the LA and genetic algorithms for real-valued function optimization. The latter algorithm termed GLA bears similarity to the population-based incremental learning algorithm. The main task in Pattern Recognition is to output a class label from a feature vector given as input. In [52], LA was used where the actions of the LA are the possible classes. An LA gets rewarded or penalized in the training phase depending on the real class of the input. However, according to Barto and Anandan: “an action is optimal only in the context of certain feature vectors” [52]. This problem is known as associative learning where the aim is to learn to associate different inputs to different actions.

Moreover, LA was also used to learn the parameters of neural networks as an alternative of the classical gradient descent methods [53].

4.3 Swarm intelligence for classification

Swarm intelligence denotes a set of nature-inspired paradigms that have received a lot of attention in computer science due to its simplicity and adaptability [54]. Ant Colony Optimizaiton (ACO) figures among the most popular swarm intelligence algorithms due to its ability to solve

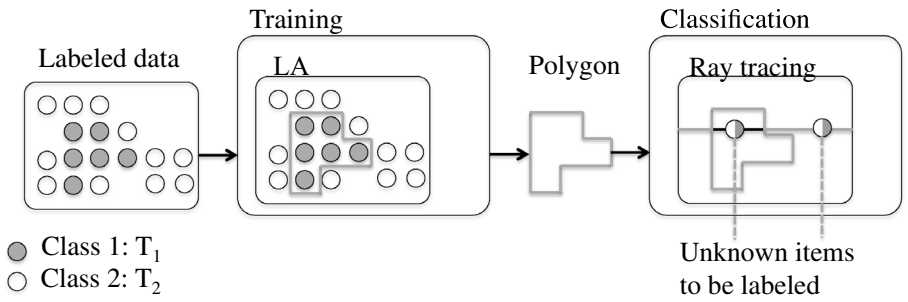

Fig. 4 Overview of approach applied to a simple classification problem

many optimization 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 behavior. The colony of ants will thus concentrate its walk on the most favorable paths and in consequence iteratively optimize the solution [55].

Recently, work on ACO for classification where the ants perform walks to separate classes has been published [4, 56–58].[3] The approach, named PolyACO, relies upon ants walking in two and many dimensions to circumvent and separate classes from each other, and in this way constructing decision boundaries not limited by linear or polynomial functions. Our current work is inspired by PolyACO [4] which pioneered the idea of using the reinforced random walk over a polygon for solving classification problem. There are two main differences between PolyACO, and between our approach PolyLA. First, PolyLA is less computationally intensive than PolyACO as the latter uses global updates while the former resorts to local updates. In fact, because of the evaporation effect of the trails, all the pheromones of all edges in the graph need to be updated at each iteration in PolyACO. In PolyLA, local updates are performed as only the LA probabilities of the edges of nodes along the chosen path are adjusted. Despite the simplicity of PolyLA, we shall show that it exhibits comparable performance to PolyACO in the experimental Sect. 6. The second difference lies in the fact that PolyLA uses negative feedback update by virtue of applying the theory of LA. The term negative feedback was reckoned by Di Caro and Dorigo in their seminal work [59] where they contrast LA and ACO approaches for distributed routing over a graph. In [59], Di Caro and Dorigo pointed out the difficulty of creating LA systems that perform well over graph problems due to stability problem. According to Di Caro and Dorigo [59], " it would be interesting to investigate the use of negative reinforcements, even if it can potentially lead to stability problems, as observed by people working on older automata systems." In simple words, negative feedback arises as each node involved in the chosen path performs local updates by reducing the choice probability of the non-walked edge while increasing the choice probability of the edge lying along the nodes of the chosen path at the given iteration. ACO only uses positive feedback as the edge along the walked path is reinforced via pheromones. In this paper, we provide theoretical results that show that LA converge to an optimal solution. The theoretical results are novel in the field of LA as this work is one of the few works that presents formal proofs for the convergence LA on a distributed graph while related LA works usually conjecture similar theorems [13, 35, 36].

4.4 Support vector machine

Classification problems usually involve finding classification boundaries in feature spaces. Among the early and most popular classifiers figures the perception algorithm.

Perception works based on "error-driven learning" where it iteratively learns a linear separator model by adjusting the weights of the model whenever it 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 nonlinearly separable data. In order to deal with the limitation of linear classifier, nonlinear SVM variants were proposed. SVM tries to circumvent over-fitting by choosing the maximal margin hyperplane where the margin is the smallest distance between the decision boundary and any of the data points.

A powerful concept in SVM is the "kernel trick" equivalent to mapping the data to higher-dimensional feature space in which the data items can be separable. Despite the well recognized performance of SVM in machine learning community, the task of choosing the right type of kernel, for example, linear, polynomial, Gaussian is considered as a black art!

[3] By some of the authors of this paper.

5 PolyLA

This section presents our approach for the two-class classification by introducing PolyLA. For the training phase, it maps the classification problem to a combinatorial optimization 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, PolyLA trains the classifier by defining a polygon s. Subsequently, it uses s with ray casting to find if an item is part of the s.

Figure 4 presents an overview of the approach in the case of a simple two-class classification problem. The data are 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 be inside the polygon, while the second item will be classified as T_2 ("Class 2") since it is 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 PolyLA solution. In order to find whether a point is 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, E is the set of directed links in the graph. We attach a 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 $\bar{\pi}_i^D(t) = [\pi_{i1}^D(t).\mathbb{1}_{N(i)}(j), \pi_{i2}^D(t).\mathbb{1}_{N(i)}(2), \ldots \pi_{im}^D(t).\mathbb{1}_{N(i)}(m)]$. Where $\mathbb{1}_{N(i)}$ is the indicator function which is such $\mathbb{1}_{N(i)}(j)$ equals 1 if node $j \in N(i)$ otherwise $\mathbb{1}_{N(i)}(j) = 0$. This simple notation is just to emphasize that the only actions are the neighbors of the node i. Note also that $\pi_{ii}^D(t) = 0$. 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. Loosely 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.

The LA update equations at node S are given by:

$$\pi_{Sj}^D(t+1) = \pi_{Sj}^D(t) + \lambda f(s(t))(\delta_{ju} - \pi_{Sj}^D(t)) \tag{10}$$

Where u is the next hop chosen by the LA attached at the source S.

$$\delta_{ju} = \begin{cases} 1 & \text{if } j = u \\ 0 & \text{else} \end{cases} \tag{11}$$

Note that, initially

$\pi_{Sj}^D(0) = \frac{1}{|N(S)|}$, for $j \in N(S)$.

Similarly, we can define the equation for the update along the path $s(t)$ that starts at the source node S and ends at destination node $D = S$.

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

$$\pi_{Sj}^D = \begin{cases} 1 & \text{if } j = j^* \\ 0 & \text{else} \end{cases} \tag{12}$$

Algorithm 1 summarizes the entire process in a high-level pseudocode algorithm of PolyACO.

Algorithm 1 High-level pseudo code algorithm for PolyLA

```
CONSTRUCT_ENVIRONMENT(training_data)
while Not all the LA converged along a self-enclosing path do
    Next Node_j ← according to π^D_Sj
    while Node not at target D do
        SELECT_NEXT_VERTEX(According to Probability Vector at previous vertex)
        s(t) ← s(t) ∪ Last visited vertex by LA
    end while
    f(s(t)) ← Classification Performance of Path s(t)
    for all successive pairs of vertices (i, j) in in path s(t) do
        π^D_ij(t+1) = π^D_ij(t) + λf(s(t))(1 − π^D_ij(t))
    end for
end while
```

Example Suppose for example that from node S, node j_1 is visited, subsequently node j_2 then node j_3 then node j_4, then node S again. Hence, all the probability distributions $\bar{\pi}^D_S(t)$, $\bar{\pi}^D_{j_1}(t)$, $\bar{\pi}^D_{j_2}(t)$, $\bar{\pi}^D_{j_3}(t)$, $\bar{\pi}^D_{j_4}(t)$ are updated according the value of the path $s(t) = (j_1, j_2, j_3, j_4, S)$.

Theorem 1 *Let s^* is path yielding the highest $f(s)$. And let i^* a node along s^*. When the learning gain λ is sufficiently small, $\pi^D_{i^*j}$ in the cross-correlation learning algorithm converges to the scalar θ^D_{ij} which yields the highest accuracy, i.e., $\lim_{t\to\infty} P|\pi^D_{i^*j} - \theta^D_{i^*j}| > \epsilon = 0$, where $\theta^D_{i^*j} = 1$ if both node i and j are along the optimal path otherwise, $\theta^D_{i^*j} = 0$ if i along the best path while j is not.*

Proof We shall prove that the learning algorithm defined converges to the optimal solution defined by the edges of the optimal polygon.

In the stochastic network environment, according to the Kushner's weak convergence method [60] and following the proof in Vazquez-Abad and Mason's work [61] as well as the proof by Li et al. [37], we can derive from the cross-correlation algorithm that as the learning gain λ goes to zero, the following equation is satisfied:

$$\frac{\mathrm{d}\pi^D_{Sj}(t)}{\mathrm{d}t} = -\lambda\pi^D_{Sj}(t)(\Delta^D_{Sj}(t) - \sum_u \Delta^D_{Su}(t)\theta^D_{Su})$$

λ corresponds to an update rate.

Δ^D_{Sj} corresponds to the average value of $f(s(t))$ where $s(t)$ includes the nodes S and j: we describe it by $Sj \in s(t)$, meaning edge Sj belongs to the path. More formally $\Delta^D_{Sj} = E(f(s(t)) \mid \bar{\pi}^D_i(t), 1 \le i \le m, and\ Sj \in s(t))$.

To show that the solution is globally stable, let us define $M^D_S(t) = \sum_j \pi^D_{Sj}(t)\Delta^D_{Sj}$.

From the cross-correlation learning algorithm [37], we can write:

$$M^D_S(t+1) - M^D_S(t) = -\sum_j \pi^D_{Sj}(t)\left(\Delta^{D\,2}_{Sj} - \left(\sum_j \pi^D_{Sj}(t)\Delta^D_{Sj}\right)^2\right) \tag{13}$$

Let s^* the optimal path possessing the best performance. $M^D_S(t+1) - M^D_S(t) \le 0$ since, i.e., since $(\Delta^D_{Sj})^2 - (\sum_j \pi^D_{Sj}(t)\Delta^D_{Sj})^2$ equals the variance of Δ^D_{Sj}. Let $M(t) = \sum_{i^* \in s^*} M^D_{i^*}$. Then, $M(t)$ is monotonically decreasing with each update of the vector $\bar{\pi}$ for i along $s*$. Let $\Delta M(t) = \sum_{i \in s^*} M^D_{i^*}(t+1) - M^D_{i^*}(t)$. When $\pi^D_{i^*j} = \theta^D_{i^*j}$, $\Delta M(t) = 0$ and reaches a stationary state.

Hence, when the learning gain is sufficiently small, the expected rewards keep increasing with time. The optimal solution may be not unique, but these optimal solutions will give the same value for the objective function. □

5.2 Cost function

Equation 14 represents the cost function. The cost function takes into account the information about whether an item t_i is inside or outside of a polygon s (see Sect. 2). This cost function measures how good a polygon s is at encircling and isolating one class in the training data and is defined as:

$$f(s) = \frac{\sum_{t_i \in T_1} h(t_i, s) + \sum_{t_j \notin T_1}(1 - h(t_j, s))}{|T|}. \tag{14}$$

In layman's terms; function 14 gives the percentage of items that are either correctly inside or correctly outside of the polygon. From the example in Fig. 1, the red polygon s correctly encircles all items of class T_1, while correctly avoids to encircle any other items from the other class T_2. Since s is a polygon that perfectly separates the two classes, it gives $f(s) = 1$.[4]

The problem reduces to optimizing $f(s)$, given the training data T, subject to the search space $\mathbf{S}$—which is equivalent to finding an $s* \in \mathbf{S}$ so that $f(s*) \le f(s) \in \mathbf{S}$.

5.3 Ray casting

Vertical ray casting is used to consider whether an item is within or outside a many-edged polygon [62]. Ray casting is a simple algorithm that determines where a virtual ray enters and exits a given solid.

In a two-dimensional XY-plane, a ray is sent with a y-coordinate and starting at 0 and is increased by one very time an edged is passed. When the ray hits the item to be labeled, whether it is inside or outside the polygon is determined by reading the bit. An even number means outside, while an odd number means inside. Formally, for node t_i and a polygon s, we get $h(t_i, s)$ representing to what extent it is inside or outside of the polygon as follows, extending Eq. 1:

$$h(t_i, s) = \begin{cases} 1 & \text{if } t_i \in T_1 \text{ and is inside of } s. \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

$h(t_i, s)$ gives 1 if t_i is correctly inside of the polygon, 0 otherwise. Note that the cost function $f(s)$ in Eq. 14 handles both items correctly inside and correctly outside of polygons.

[4] Note that s is one of the possible polygons with the shortest circumference that is able to perfectly separate the data. The reason for this is explained in Sect. 5.5.

5.4 Remark about uniqueness of the path

An implicit assumption is that the optimal path is unique. However, in many cases, the optimal polygon is not unique and there might be multiple polygons yielding the same performance. This will result in multiple equilibrium [61]. Our experimental results confirm the convergence to one of the equilibriums.

5.5 Training phase

The classifier is trained using a guided walk with the team of distributed LA optimizing for the score function $f(s)$ in order to create a polygon. By virtue of the reinforcement learning mechanism, the actions of the team of LA will converge toward a polygon that is a good separator. This polygon is the key to the classification.

Note that the classifier, implicitly, performs optimization according to the score function $f(s)$.

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

The LA random walker is not allowed to walk on nodes that has previously been selected, except for the initial starting node.

5.6 Bootstrapping the source node

A detail worth mentioning is the way by which we choose of the source node of the polygon. The performance of the scheme is dependent on the right choice of the source vertex for the polygon. In order to deal with this disadvantage, we allow the scheme to re-adjust its choice of the source vertex. 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. 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 the latter simple strategy gave good performance.

6 Experiments

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

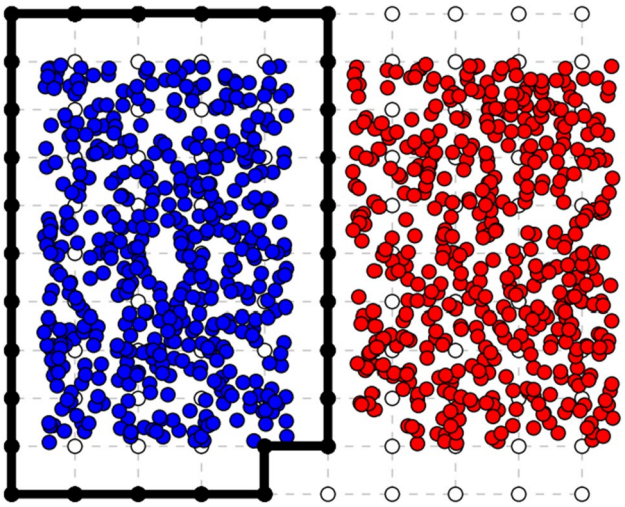

Fig. 5 Simple data set with 0% noise

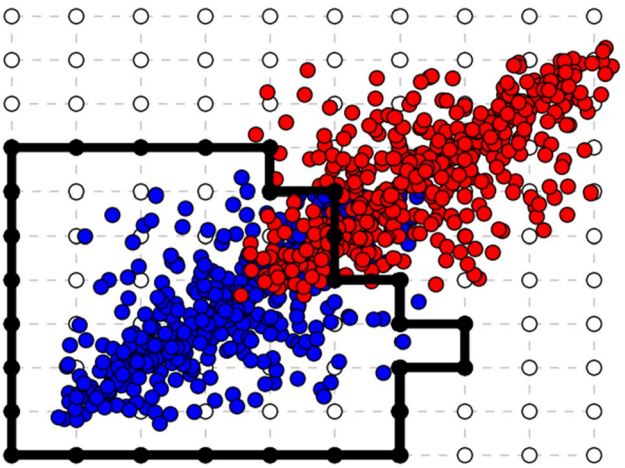

Fig. 6 Gaussian distribution with overlap

The data are generated by various functions intended to show the performance of PolyLA in various settings. In each experiment, 2000 data points are generated, of which half are randomly selected for training and the rest used for classification. Further, the data always contain two classes: the blue T_1 class and the red T_2 class.

The granularity of the grids is always chosen as 10×10. A summary of the results is presented in Table 1.

6.1 Simple environments

We present a simple experimental settings as proof of concept of PolyLA. This section empirically shows that the

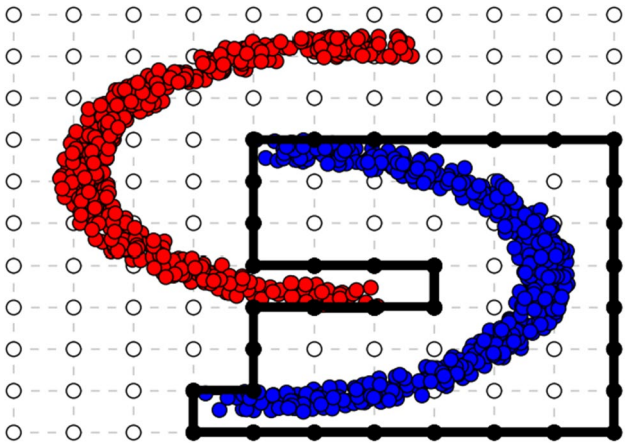

Fig. 7 Half-moon

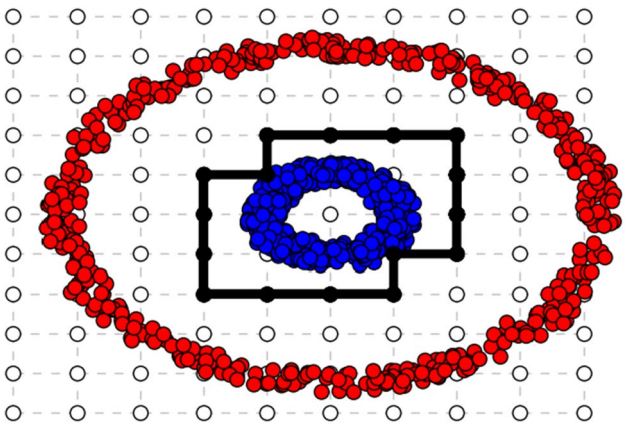

Fig. 8 Circular without noise

approach works in a simple environment with two easily separable sets of data. The data are composed of two blocks of data: T_1 and T_2. Figure 5 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, PolyLA gave an accuracy of 100%.

6.2 Gaussian

Figure 6 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, due to the overlapping data no classifier will be able to give a perfect result and is therefore a good test for PolyLA.

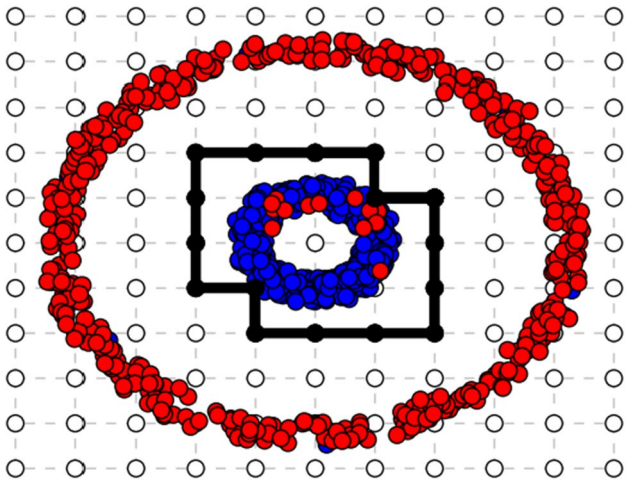

Fig. 9 Circular with 5% noise

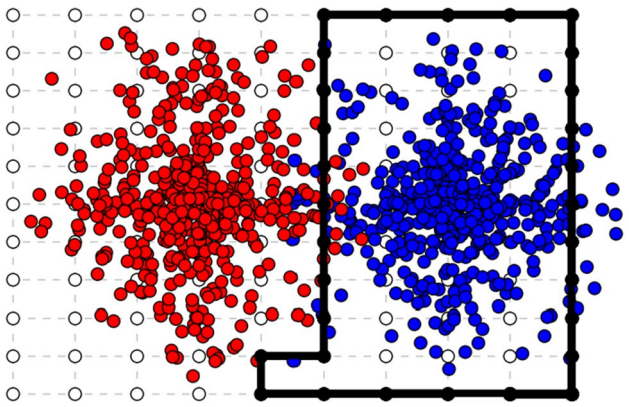

Fig. 10 Gaussian blob distance 140

For the classification, the PolyLA classifier gave a accuracy of 0.846, a recall of 0.836 and a precision of 0.853. For comparison purposes, linear and polynomial SVM gave the same data accuracies of 0.837 and 0.842, respectively.

These high numbers indicate that PolyLA is able to perform in line with SVM even when data are overlapping—without loss of precision or recall.

6.2.1 Semi-circles

Figure 7 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.

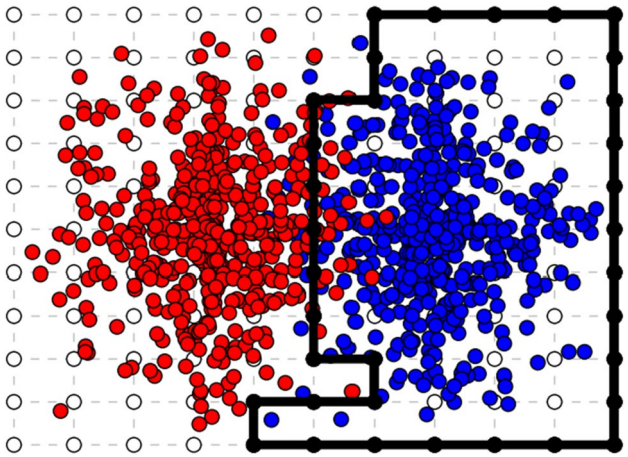

Fig. 11 Gaussian blob distance 120

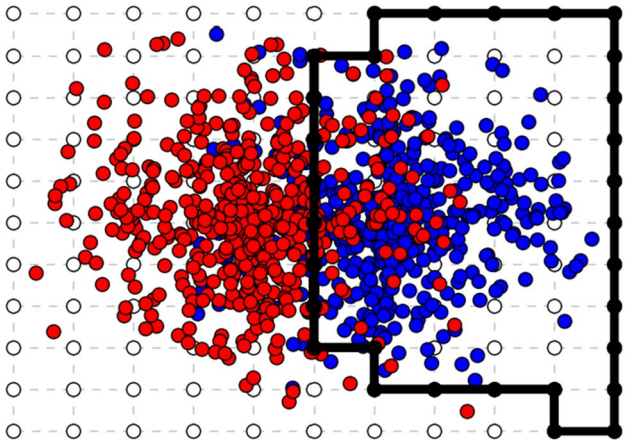

Fig. 12 Gaussian blob distance 60

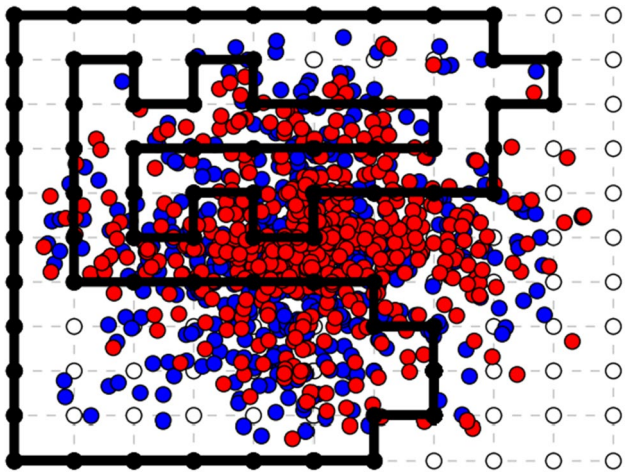

Fig. 13 Gaussian blob distance 0—indistinguishable from random noise

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

6.2.2 Circles

Figure 8 illustrates the case of nonlinear classification boundary in the form of a circle.

Despite the added complexity, the PolyLA approach works perfectly by surrounding the data from the blue class without including the red. Again, the accuracy, precision and recall for PolyLA are 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 PolyLA, we need to rely on a RBF kernel.

In Fig. 9, we add some noise to the data of 5%. Noise means simply that some points of one class are overlapping with the other class making impossible to separate between these overlapping points. Despite the added noise, the scheme performs well. We would expect an approximate 2.5% loss in accuracy because of the 5% noise. Our empirical results confirm this by giving an accuracy of 0.973. For comparison purposes, SVM performance drops significantly by adding noise.

6.2.3 Gaussian blobs

Figures 10, 11, 12 and13 depict the case of data generated from Gaussian distributions with blob distance that are, respectively, 140, 120, 60 and 0.

These are interesting results because it explains the behavior of PolyLA when the data are overlapping more and more, and in turn becoming more and more difficult to separate. In the most extreme, with a distance of 0 in Fig. 13, the data are overlapping and should be indistinguishable from complete random data.

In Fig. 10, the data are barely overlapping and PolyLA yields in the classification phase an accuracy of 0.946, a recall of 0.936 and precision of 0.956.

In Fig. 11, the data are slightly more overlapping. However, PolyLA has barely any drop in performance. It is still able to accurately separate the data and yields in the classification phase an accuracy of 0.943, a recall of 0.936 and precision of 0.938.

In Fig. 12, the data are overlapping a lot and PolyLA yields in the classification phase an accuracy of 0.747, a recall of 0.684 and precision of 0.78. It is noteworthy that, by examining the figure, it is apparent that a higher granularity of the grid would give a better algorithm performance.

Table 1 Summary of PolyLA performance

Scenario	Accuracy	Precision	Recall	LSVM	PSVM	1NN	2NN	3NN	Figures
Proof of concept	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	Figure 5
Gaussian distributions	0.85	0.84	0.85	0.84	0.82	0.84	0.05	0.84	Figure 6
Semi-circles	1.0	1.0	1.0	0.91	1.0	0.92	1.0	1.0	Figure 7
Circular	1.0	1.0	1.0	0.54	0.89	0.85	0.84	0.85	Figure 8
Circular 5% noise	0.97	0.97	0.97	0.54	0.89	0.56	0.56	0.56	Figure 9
Gaussian blob distance 140	0.95	0.94	0.96	0.51	0.73	1.0	1.0	1.0	Figure 10
Gaussian blob distance 120	0.943	0.936	0.938	0.486	0.734	0.95	0.94	0.95	Figure 11
Gaussian blob distance 60	0.747	0.684	0.780	0.497	0.689	0.84	0.83	0.77	Figure 12
Gaussian blob distance 0	0.52	0.53	0.49	0.49	0.56	0.56	0.56	0.56	Figure 13
Iris	0.82	0.61	0.74	1.0	1.0	0.97	0.97	0.97	
Wine	0.68	0.32	0.58	0.69	0.66	1.0	10	1.0	

Compared with the accuracy of linear SVM (LSVM) polynomial SVM (PSVM) one hidden layer neural network (1NN), two hidden layers neural network (2NN), three hidden layer neural network (3NN)

In Fig. 13, the data are completely overlapping and the classes are indistinguishable from each other. Clearly, PolyLA has a very different behavior here than in Figs. 12, 11 and 10. In this scenario, there is no apparent pattern in the polygon. As with the data, the polygon appears random. Our empirical results confirm the results giving in the classification phase an accuracy barely above random of 0.526, a recall of 0.528 and a precision of 0.486.

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

6.2.4 Real-life data sets

In the above experiments, we have focused on illustrating the performance of the PolyLA using figures that demonstrate its ability to perform separation. At this juncture, we shall use two real-life data sets: the Iris data set and the Wine data set. It is worth mentioning that originally PolylA does not handle directly the case of multi-dimensional classification arising in the case of Iris and Wine data sets. We deal with the multi-dimensional case according to the method detailed in Sect. 2.3.

We also need to emphasize that PolyLA possesses similar performance to PolyACO by examining the results reported in [56]. In fact, PolyACO achieves 0.948 accuracy while PolyLA outperforms it by achieving 0.97 for the circular case with 5% noise. However, PolyACO achieves slightly higher performance for the Iris data set, namely 0.96 accuracy, while PolyLA achieves 0.82. When it comes to the Wine data set, the performance for PolyLA is 0.68 while PolyACO yields 0.69. Furthermore, in Table 1, we compare against three neural networks, one hidden layer neural network (1NN), two hidden layers neural network (2NN) and three hidden layer neural network (3NN). Please note that 2NN and 3NN are considered as deep learning algorithms. We observe from Table 1 that those neural networks outperform PolyLA, Linear SVM (LSVM) and Polynomial SVM (PSVM). Although PolyLA is able to find nonlinear classification boundaries that might be complex to find and consequently outperform LSVM, it is less accurate compared to deep neural networks which excel in builduing nonlinear decision boundaries.

7 Conclusion

In this paper, we propose a novel classifier in two-dimensional feature space based on the theory of Learning Automata. The essence of our scheme is to search for a separator in the feature space by imposing an LA-based random walk in a grid system. To each node in the gird, we attach an LA, whose actions are the choices of the edges forming the separator. Indeed, PolyLA has appealing properties compared to SVM. While SVM performance is subject to the user choice of the kernel function, PolyLA can find arbitrarily complex separator in feature space without any user guidance. We provide theoretical results that demonstrate the convergence of PolyLA based on the theory of weak convergence [60]. Comprehensive experimental results illustrate the performance of our method and its superiority to SVM in most cases. PolyLA faces challenges when dealing with multi-dimensional data as inevitably, the number of planes explodes as the number of features increases. It would be interesting to investigate mitigating this issue in future work.

References

1. Caruana R, Niculescu-Mizil A (2006) An empirical comparison of supervised learning algorithms. In: Proceedings of the 23rd international conference on machine learning. ACM, pp 161–168
2. Caruana R, Karampatziakis N, Yessenalina A (2008) An empirical evaluation of supervised learning in high dimensions. In: Proceedings of the 25th international conference on machine learning. ACM, pp 96–103
3. Madjarov G, Kocev D, Gjorgjevikj D, Džeroski S (2012) An extensive experimental comparison of methods for multi-label learning. Pattern Recognit 45(9):3084–3104
4. Goodwin M, Tufteland T, Ødesneltvedt G, Yazidi A (2017) Polyaco+: a multi-level polygon-based ant colony optimisation classifier. Swarm Intell 11(3–4):317–346
5. Agache M, Oommen BJ (2002) Generalized pursuit learning schemes: new families of continuous and discretized learning automata. IEEE Trans Syst Man Cybern Part B Cybern 32(6):738–749
6. Lakshmivarahan S (1981) Learning algorithms theory and applications. Springer, Berlin
7. Najim K, Poznyak AS (1994) Learning automata: theory and applications. Pergamon Press, Oxford
8. Narendra KS, Thathachar MAL (1989) Learning automata: an introduction. Prentice-Hall, Inc, Upper Saddle River
9. Obaidat MS, Papadimitriou GI, Pomportsis AS (2002) Learning automata: theory, paradigms, and applications. IEEE Trans Syst Man Cybern Part B Cybern 32(6):706–709
10. Poznyak AS, Najim K (1997) Learning automata and stochastic optimization. Springer, Berlin
11. Thathachar MAL, Sastry PS (2003) Networks of learning automata: techniques for online stochastic optimization. Kluwer Academic, Boston
12. Tsetlin ML (1973) Automaton theory and the modeling of biological systems. Academic Press, New York
13. Misra S, Oommen BJ (2004) GPSPA: a new adaptive algorithm for maintaining shortest path routing trees in stochastic networks. Int J Commun Syst 17:963–984
14. Obaidat MS, Papadimitriou GI, Pomportsis AS, Laskaridis HS (2002) Learning automata-based bus arbitration for shared-edium ATM switches. IEEE Trans Syst Man Cybern Part B 32:815–820
15. Oommen BJ, Roberts TD (2000) Continuous learning automata solutions to the capacity assignment problem. IEEE Trans Comput 49:608–620
16. Papadimitriou GI, Pomportsis AS (2000) Learning-automata-based TDMA protocols for broadcast communication systems with bursty traffic. IEEE Commun Lett 4:107–109
17. Atlassis AF, Loukas NH, Vasilakos AV (2000) The use of learning algorithms in ATM networks call admission control problem: a methodology. Comput Netw 34:341–353
18. Atlassis AF, Vasilakos AV (2002) The use of reinforcement learning algorithms in traffic control of high speed networks. In: Zimmermann H-J, Tselentis G, van Someren M, Dounias G (eds) Advances in computational intelligence and learning. International Series in Intelligent Technologies, vol 18. Springer, Dordrecht, pp 353–369
19. Vasilakos AV, Saltouros MP, Atlassis AF, Pedrycz W (2003) Optimizing QoS routing in hierarchical ATM networks using computational intelligence techniques. IEEE Trans Syst Man Cybern Part C 33:297–312
20. Seredynski F (1998) Distributed scheduling using simple learning machines. Eur J Oper Res 107:401–413
21. Kabudian J, Meybodi MR, Homayounpour MM (2004) 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. Nevada, Las Vegas, pp 638–642
22. Meybodi MR, Beigy H (2002) New learning automata based algorithms for adaptation of backpropagation algorithm pararmeters. Int J Neural Syst 12:45–67
23. Unsal C, Kachroo P, Bay JS (1997) Simulation study of multiple intelligent vehicle control using stochastic learning automata. Trans Soc Comput Simul 14:193–210
24. Oommen BJ, de St Croix EV (1995) Graph partitioning using learning automata. IEEE Trans Comput 45:195–208
25. Collins JJ, Chow CC, Imhoff TT (1995) Aperiodic stochastic resonance in excitable systems. Phys Rev E 52:R3321–R3324
26. Cook RL (1986) Stochastic sampling in computer graphics. ACM Trans Graph 5:51–72
27. Barzohar M, Cooper DB (1996) Automatic finding of main roads in aerial images by using geometric-stochastic models and estimation. IEEE Trans Pattern Anal Mach Intell 7:707–722
28. Brandeau ML, Chiu SS (1989) An overview of representative problems in location research. Manag Sci 35:645–674
29. Bettstetter C, Hartenstein H, Prez-Costa X (2004) Stochastic properties of the random waypoint mobility model. J Wirel Netw 10:555–567
30. Rowlingson BS, Diggle PJ (1991) SPLANCS: spatial point pattern analysis code in S-plus. University of Lancaster, North West Regional Research Laboratory
31. Paola M (1998) Digital simulation of wind field velocity. J Wind Eng Ind Aerodyn 74–76:91–109
32. Cusumano JP, Kimble BW (1995) A stochastic interrogation method for experimental measurements of global dynamics and basin evolution: application to a two-well oscillator. Nonlinear Dyn 8:213–235
33. Baddeley A, Turner R (2005) Spatstat: an R package for analyzing spatial point patterns. J Stat Softw 12:1–42
34. Oommen BJ, Agache M (2001) Continuous and discretized pursuit learning schemes: various algorithms and their comparison. IEEE Trans Syst Man Cybern Part B Cybern 31:277–287
35. Misra S, Oommen BJ (2005) Dynamic algorithms for the shortest path routing problem: learning automata-based solutions. IEEE Trans Syst Man Cybern Part B Cybern 35(6):1179–1192
36. Misra S, Oommen BJ (2006) An efficient dynamic algorithm for maintaining all-pairs shortest paths in stochastic networks. IEEE Trans Comput 55(6):686–702
37. Li H, Mason L, Rabbat M (2009) Distributed adaptive diverse routing for voice-over-ip in service overlay networks. IEEE Trans Netw Serv Manag 6(3):175–189
38. Mason L (1973) An optimal learning algorithm for s-model environments. IEEE Trans Autom Control 18(5):493–496
39. Beigy H, Meybodi MR (2006) Utilizing distributed learning automata to solve stochastic shortest path problems. Int J Uncertain Fuzziness Knowl Based Syst 14(05):591–615
40. Torkestani JA, Meybodi MR (2010) An intelligent backbone formation algorithm for wireless ad hoc networks based on distributed learning automata. Comput Netw 54(5):826–843
41. Torkestani JA, Meybodi MR (2012) Finding minimum weight connected dominating set in stochastic graph based on learning automata. Inf Sci 200:57–77
42. Torkestani JA, Meybodi MR (2012) A learning automata-based heuristic algorithm for solving the minimum spanning tree problem in stochastic graphs. J Supercomput 59(2):1035–1054
43. Thathachar MAL, Sastry PS (2002) Varieties of learning automata: an overview. IEEE Trans Syst Man Cybern Part B Cybern 32(6):711–722
44. Sastry P, Thathachar M (1999) Learning automata algorithms for pattern classification. Sadhana 24(4):261–292

45. Shah S, Sastry PS (1999) New algorithms for learning and pruning oblique decision trees. IEEE Trans Syst Man Cybern Part C (Appl Rev) 29(4):494–505
46. Thathachar MAL, Sastry PS (1987) Learning optimal discriminant functions through a cooperative game of automata. IEEE Trans Syst Man Cybern 17(1):73–85
47. Santharam G, Sastry P, Thathachar M (1994) Continuous action set learning automata for stochastic optimization. J Frankl Inst 331(5):607–628
48. Zahiri S (2008) Learning automata based classifier. Pattern Recognit Lett 29(1):40–48
49. Zeng X, Liu Z (2005) A learning automata based algorithm for optimization of continuous complex functions. Inf Sci 174(3):165–175
50. Afshar S, Mosleh M, Kheyrandish M (2013) Presenting a new multiclass classifier based on learning automata. Neurocomputing 104:97–104
51. Howell M, Gordon T, Brandao F (2002) Genetic learning automata for function optimization. IEEE Trans Syst Man Cyber 32(6):804–815
52. Barto AG, Anandan P (1985) Pattern-recognizing stochastic learning automata. IEEE Trans Syst Man Cybern 3:360–375
53. Meybodi MR, Beigy H (2002) New learning automata based algorithms for adaptation of backpropagation algorithm parameters. Int J Neural Syst 12(01):45–67
54. Cochran JJ, Cox LA, Keskinocak P, Kharoufeh JP, Smith JC, Stützle T, López-Ibáñez M, Dorigo M (2011) A concise overview of applications of ant colony optimization. In: Cochran JJ, Cox LA, Keskinocak P, Kharoufeh JP, Smith JC (eds) Wiley encyclopedia of operations research and management science. https://doi.org/10.1002/9780470400531.eorms0001
55. Dorigo M, Birattari M, Stutzle T (2006) Ant colony optimization. IEEE Comput Intell Mag 1(4):28–39
56. Goodwin M, Yazidi A (2016) Ant colony optimisation-based classification using two-dimensional polygons. In: International conference on swarm intelligence. Springer, pp 53–64
57. Tufteland T, Ødesneltvedt G, Goodwin M (2016) Optimizing polyaco training with GPU-based parallelization. In: International conference on swarm intelligence. Springer, pp 233–240
58. Goodwin M, Tufteland T, Ødesneltvedt G, Yazidi A (2016) Polyaco+: a many-dimensional polygon-based ant colony optimization classifier for multiple classes. Journal Article (under review)
59. Di Caro G, Dorigo M (1998) Antnet: distributed stigmergetic control for communications networks. J Artif. Intell. Res. 9:317–365
60. Kushner HJ, Clark DS (2012) Stochastic approximation methods for constrained and unconstrained systems, vol 26. Springer, Berlin
61. Vázquez-Abad FJ, Mason LG (1996) Adaptive decentralized control under non-uniqueness of the optimal control. Discrete Event Dyn Syst 6(4):323–359
62. Roth SD (1982) Ray casting for modeling solids. Comput Graph Image Process 18(2):109–144