Modeling Learning Automata based classification algorithms

Soumya Maulik

A thesis presented for the degree of
Masters by Research

School of Engineering
Waterford Institute of Technology
Ireland

April 2017

Supervisors: **Dr. Paul O'Leary** and **Dr. Derek Sinnott**

Declaration

The work in this thesis is based on research carried out at the School of Engineering, Waterford Institute of Technology, Ireland. No part of this thesis has been submitted elsewhere for any other degree or qualification and it all my own work unless referenced to the contrary in the text.

(Soumya Maulik)
Student Number- 20064915.

Acknowledgements

There are many people without whom the research presented here would not have taken shape. For that I must thank my supervisors, Dr. Paul O'Leary and Dr. Derek Sinnott for providing the funds for this research. I am extremely thankful to Dr. O'Leary for his tireless effort in scrutinizing the language in which the research is presented. I am grateful to Dr. Derek Sinnott for providing the funds for the computing facility used for the simulation. The richness of the technical suggestions from my supervisors helped greatly in constructing this document.

I would also like to thank Dr. Ken Thomas, Head of School Engineering for his kind words and inspiration.

I must thank Dr. Bernard Butler of TSSG, for his suggestions that helped in enriching and shaping the contents of the article to be published from this work. Most importantly, his precise suggestions led to an improvement in the quality of the figures presented here. I also should thank Dr. Kieran Murphy from the School of Science and Computing, for his queries on the research. Addressing those queries has helped greatly in improving the quality of this work. I also thank Dr. Denis Flynn and Dr. Brendan McCann for their inspiration and support.

I must thank my colleagues Clair McDonald, Anup Poudel and Joan Edwards, as they were always there in the research office whenever I required a distraction. I am lucky to be around some amazing people in the School of Engineering, who had the solutions to every problem I faced.

I must thank my parents for their inspiration and support. Without their sacrifice, this work would not have been possible. Last, but perhaps one of the most important, I should thank my wife for constantly motivating me and supporting me during the challenging periods.

Abstract

Learning Automata is extensively used as a tool by researchers to achieve solutions to various problems pertaining to engineering. One such application is a learning automata based classification algorithm. However, at present, learning automata based classifiers have been limited to only 2-class classification. In this research two learning automata based models, for multi-class classification, have been presented. The performance of the proposed techniques has been evaluated using data sets that are extensively used for benchmarking purposes. Since the ultimate goal of an efficient model is to be accurate with low computational complexity, both the performance and complexity of the proposed techniques have been evaluated using the benchmark data sets. Furthermore, the performance of the proposed model has been compared to the performance of existing and extensively used machine learning algorithms on the same benchmark data sets. Apart from modeling and testing the algorithms, the present research also proposes a methodology for evaluating the performance of machine learning algorithms. A detailed study of the existing performance evaluation techniques for machine learning algorithms has been provided; highlighting their drawbacks and then proposing a simple and efficient technique for evaluating their performance. For both pieces of the work, i.e. both the proposed learning automata based multi-class classification algorithms and the performance evaluation methodology, the theoretical analysis has been carried out in a tight mathematical framework supported by simulation results. A theoretical background of learning automata has also been provided, along with a description of the evolution of the proposed algorithms from the basics of learning automata theory.

Contents

Chapter 1

Introduction

1.1 Background & Motivation

We are in the era of big data, due to increased human access to the Internet, automation in industry and other factors. To give a few examples, one hour of video is uploaded to Youtube, every second [1], Walmart manages more than one million transactions per hour [2], and Walmart itself has 2.5 petabyte (10^{15} bytes) of storage for its user database [3]. With this deluge of data, there is a great need for tools that can analyze the data. The patterns that may exist in the data can assist in decision making processes, thereby providing benefits to the end users. The subject that has emerged from finding the patterns existing in data, is called *machine learning*. The most basic theoretical model of machine learning consists of an agent or automaton and an environment. Typically, the responsibility of the environment is to provide the data, based on which the agent makes decisions. The additional responsibility of the environment is to evaluate the decision of the agent. Based on the nature of the learning technique and the nature of the data, machine learning methods are divided into three broad categories *viz.* supervised learning, reinforcement learning and unsupervised learning. Supervised learning [4] is learning from the environment inputs, which are labeled data. For unsupervised learning [5], the automaton forms the natural groups or clusters of the environment unlabeled inputs. Reinforcement learning [6] involves sequential interaction between an active decision making agent and the labeled input supplied by the environment. In this work, the learning methods, comprising of data with labels, are considered.

The aim of all learning methods is to make the agent predict the data class labels accurately, when the environment provides unseen or unknown data. To fulfill this, normally, supervised learning methods train the agent with the available data and then apply the training experience on the unseen or unknown data. However, it is often impractical to obtain data samples of desired behavior that are both correct and representative of all the situations in which the agent has to act. Also, for supervised learning, when new data is available for training, the learning process has to repeat the entire episode from the beginning. In other words, for supervised learning the training has to be repeated for the older data, along with the newly available data set. This type of supervised learning is called offline supervised learning. Considering that we are living in the age of big data, offline supervised learning is highly inefficient. To overcome this inefficiency, significant research is ongoing to generate the equivalent online models, of their offline counterparts. One such way of converting an offline supervised algorithm to its online version, is by enforcing a sliding window of length w and repeating that in real time. However, the disadvantages of such a technique are:

- it is difficult to select the window length w [7, 8]; and

- by this process, it is professed that training with the previous w samples are all equally important, but the next data sample, just one step beyond, is not at all important.

Also, during the training phase of the supervised learning algorithm, the knowledgeable external supervisor or the environment provides the agent with the correct decisions, which then aligns itself in such a fashion, such that future predictions are accurate. In certain scenarios, the environment can only evaluate the decision of the agent by sending a binary signal, rather than the correct decision. In other words, if the agent's decision is accurate, the environment sends $+1$ and -1 or 0, if it is incorrect. This type of situation might arise when the environment has to maintain the privacy of the data.

In reinforcement learning, an automaton predicts a class label based on data provided by the environment, and then informs the decision back to the environment. The environment evaluates the decision and either rewards or punishes the agent in the form of a scalar signal. When the agent's decision is accurate, normally the environment sends back a reward of $+1$ and for an incorrect decision, the agent gets 0 or -1. The objective of the agent in reinforcement learning is to maximize the expected rewards or minimize the penalties respectively. The interaction between the agent and the environment is sequential. When the agent is supplied with a new data sample by the environment, it predicts the class label based on past experience, accrues a reward or penalty from the environment and then subsequently modifies the experience. Thus reinforcement learning avoids repetition of computation with the same data, whenever presented with a new sample, as was the case for the offline supervised learning algorithms.

1.2 Thesis Contribution

In the context of online unsupervised learning, multi-class classification algorithms has a subtle advantage of one-vs-all classification algorithms. The reason behind that is, in the context of one-vs-all classification, the overall computational complexity increases with the increase of the number of class labels in the data. However the multi-class classifiers are immuned to this problem. The main contribution of this thesis is the modeling of learning automata based multi-class classification algorithms and the analysis of their performances in comparison to other widely used classification algorithms using the benchmark analysis on four data sets from UCI Machine Learning repository.

The thesis also highlights a new methodology for evaluating the performance of classification algorithms. The validation of the working principle of the proposed performance evaluation technique has been carried out in tight mathematical framework.

1.3 Thesis Organization

In Chapter 2, a brief description will be presented on reinforcement learning. However, unlike in most of the articles in this area, which emphasize supervised learning techniques and tend to avoid reinforcement learning, in Chapter 2 the similarities between supervised learning and the reinforcement learning technique, will be depicted.

The theory of learning automata is based on reinforcement learning, which has left an indelible imprint in the development of modern reinforcement learning research. In the early days of its origin in Russia, learning automata theory was known for solving a non-associative, purely selectional learning problem [9]. The theory of learning automata was later extended to a more versatile and robust version, applicable in a number of areas in engineering [10]. A learning automata based model, capable of handling contextual input, is available in [11].

In Chapter 3 the selective, non-associative algorithms based on learning automata, are presented. Chapter 3 also presents the research responsible for the evolution of the associative learning automata based algorithms, from the context of non-associative algorithms. Learning automata based classification algorithms were limited to 2-class classification. In Chapter 4, the existing learning automata based 2-class classification algorithms will be explained followed by the description of learning automata based multi-class classification algorithms, which are the primary contribution of this research. The algorithms will be referred as M-RAMA, which stand for the acronym "*Modified REINFORCE Algorithm for Multi-action Automaton*". The performance of the algorithms, using a synthetic data set will be provided in Chapter 4. In Chapter 5, the comparison of the performance of the proposed algorithms compared to existing and extensively used classification algorithms, using benchmark data sets is highlighted. This section also proposes a new methodology for evaluating the performance of classification algorithms, which has subtle advantages over prevalent techniques, and which is therefore also a novel contribution of this work.

Chapter 2

Reinforcement Learning

Reinforcement Learning or learning by trial and error started in the psychology of animal training [12] in as early as 1928. However, the stimulus-response observed phenomenon in animal training was never developed into a mathematical model of the process. Clark L. Hull in 1943 was the first person who proposed the scientific laws of behavior [13]. Subsequent researches were carried out by William K. Estes [14], Cletus J Burke [15], Robert R. Bush and Frederick Mosteller [16] and others.

Another thread of reinforcement learning is the pursuit of solutions to the problems pertaining to optimal control, by using value functions and dynamic programming [17]. The last, but no less important thread of reinforcement learning, is the thread of learning by interaction, which was based on a technique popularly referred as the temporal-difference method [18–21].

All of the references to past research mentioned here, that relate to reinforcement learning, may differ in approach, but have a similar objective, which is to maximize or minimize the average reward or penalty respectively. Although the thread of dynamic programming does not involve active learning in a sense, nonetheless the convergence of all the references evolved to produce the modern field of reinforcement learning.

In this paragraph, a concise theoretical description of the evolution of reinforcement learning will be provided, followed by a more quantitative description of the components of reinforcement learning in the next paragraph. For the time being, it is assumed that the decision making task is controlled by the agent or automaton, and the environment is responsible for the decision evaluation.

In an interacting scenario, an agent or automaton selects actions and, based on the selected action, the environment sends an evaluative scalar feedback from a stationary probability distribution. The problems of this type are called *Bandit Problems*, so named by the analogy to a slot machine, where each play of one of the slot machine's levers is similar to the agent's action, and the environment feedback is considered equivalent to hitting the jackpot. The environment that evaluates the action, based on a stationary probability distribution, is called a *stationary environment* and for a varied distribution, it is called a *non-stationary environment*, which is discussed in the following paragraph.
In the generalized bandit problem, an agent at every instant has to select an action from a set of n available actions. This type of problem is referred to as an *n-armed bandit problem.* The scalar feedback received by the agent for a specific action, is called the *value* of that action. The aim of the agent is to select the action that maximizes the expected reward over some time period. To achieve this, the agent generally maintains the estimates of the action values and at every instant it is preferable to select the action that has the highest estimated value. The action which has the highest estimated value, is called a *greedy* action and the policy of

selecting the greedy action based on past experience is referred as *exploitation.* Since the goal of the agent is to maximize the expected reward, hence it might appear that exploitation is the correct policy to achieve the goal. However, the policy of exploitation is disadvantageous in scenarios where the agent is unaware of some values for actions which is not tried even for ones, and those action values might be greater than the greedy action. To avoid this, it is preferable to select a greedy action in most of the instances, but also to select a non-greedy action with a small probability, (say ϵ). This enables the agent to test every available action and thereby leaves scope to improve the accrued reward over the long run. This policy is referred as the ϵ *-greedy* method. While selection of a greedy action is called exploitation, the selection of a non-greedy action is referred as *exploration.*
Research on the trade-off between exploring and exploiting has been carried out in areas of identification and control [22], in genetic algorithms [23] and in several other areas. The theory of learning automata was developed to provide simple solutions for selection learning problems, like the n-armed bandit problems. The advantage of learning automata over other approaches of reinforcement learning for selection problems is that, apart from being simple, the learning automata based models are executable even in low-memory machines.

For some cases, the interaction between the agent and the environment breaks naturally into sequences and finally ends in a special state. Such sequences of interaction are called *episodes* and the final terminating state is called the *terminal* or *absorbing state.* These processes are collectively called *episodic tasks or finite-horizon tasks.* For other cases, interaction between the automaton and the environment continues indefinitely and these processes are called *continuing tasks or infinite horizon tasks.* For the episodic tasks, at every interaction the agent is assumed to possess a representation of its state and, on that basis, selects an action from the available set of actions. The current action then causes the agent to transit to a new state. This decision of selecting an action based on the state is called the *policy.* Hence, for the episodic tasks, at any instant t, for a selected action a_t by the agent, the environment gives an evaluative feedback consisting of both the reward or the penalty and the transition state of the agent at the instant $(t+1)$. If evaluative feedback of the environment is dependent only on the last action and the state of the agent and is independent of the any further past histories of the states and the actions, then the interaction is said to satisfy the *Markov property.* A reinforcement learning process that satisfies the Markov property is called a *Markov Decision Process(MDP).* C.J. Watkins [24] introduced the notion of MDP in the context of reinforcement learning and the method is known as *Q-Learning.* Since the environment informs the next state to which the agent will transit, based on the current state and the current action, it is assumed that the environment has the information of the state transition probabilities. More detail on this is provided in Section 3.2.2.

Most reinforcement learning algorithms estimate the *value function*, which estimates the future expected reward that can be accumulated for a given action, in a given state, and thereby follow a specific policy. Generally, the algorithms estimate the *state-value function* or *action-value function* or both. The state-value function estimates the average future rewards that can be accumulated, while starting in a particular state; whereas the action-value estimates the mean reward that can be accrued in the future, by taking a specific action when in a particular state. By the law of large numbers, both the state-value function and the action-value function for every state and action respectively, will converge to the true averages, provided that the number of states and actions encountered, approach infinity.
A policy is said to be an optimal policy, if by following that policy, the optimal state-value function attains the maximum possible value. In other words, a policy is said to be optimal, if by following that policy, the transitions of states occur in a manner which enables the agent to accrue the maximum possible discounted return. In a similar way, optimal policy can be

defined using the action-value function. Richard Bellman proposed a relationship between the value of a state and the values of its successor states for episodic problems, thereby recursively finding the optimal state-value function. This relationship is known as *Bellman's equation for state-value functions* or *Bellman's optimality equation* [17]. However, it was observed the the policy attains optimality long before the convergence of the value functions and Ronald Howard proposed a technique for finding the optimal policy called *policy iteration* [25]. Both the optimal value iteration and the optimal policy iteration technique proposed by Bellman and Howard require the environment's state transition model to be available to the agent beforehand. Bellman also coined the name *dynamic programming*, referring to a collection of algorithms that can be used to compute optimal policies, given a complete model of the environment as a Markov decision process. Bellman also concluded that the computational requirements grow exponentially with the number of state variables, in problems of dynamic programming, which he referred to as "the curse of dimensionality".

Although dynamic programming is computationally complex, it does not involve any active learning and also assumes a complete knowledge of states to be available to the agent, which is not related to the exact theory of reinforcement learning, but is nonetheless studied in every context of reinforcement learning. This is because Andrew Barto and Michael Duff [26] showed the possibility of the policy evaluation in the context of Monte Carlo algorithms. Monte Carlo methods do not assume complete knowledge of the environment and require only experience. Hence, they are capable of operating online. Without any prior knowledge about the environment, Monte Carlo methods are capable of obtaining the optimal value function. In Monte Carlo methods, the agent uses the experience while interacting with the environment by following some policy and updating the value function as the process reaches the terminal state. In other words, Monte Carlo methods update the value function incrementally on an episode-by-episode basis, using the estimated discounted rewards over the entire episode.

However, Monte Carlo methods are not suitable for continuing tasks and also fail to work in a non-stationary environment. Richard Sutton [21] proposed the *TD(0)* algorithm, which updates the value function in the very next step using the immediate reward/penalty and a constant step size learning parameter. Here *TD* stands for "Temporal Difference". TD algorithms are suitable for non-stationary environments and are capable both for continuing and episodic tasks. The next step updating technique and the constant step size parameter makes TD(0) an adaptive algorithm.

Until now, whichever algorithms of reinforcement have been discussed have been in the context of a non-associative setting, i.e. the algorithms find the optimal policy, which maps the agent's actions to states that maximize the expected reward. However, in associative settings, the environment sends the agent, situations in the form of a vector and the agent decides the best action for that given context. When the environment context is available beforehand then supervised learning algorithms happen to be the most suitable. In supervised learning, a function is assumed to represent the action-context dynamics and, based on the available context, the learning method constructs an approximation of the entire function. This method is referred to as function approximation and is extensively used as a classification technique. The function approximation technique can be used in reinforcement learning as well. In a nutshell, the TD(0) method, coupled with function approximation, can assist in building an online reinforcement learning based classifier. The assumed function can be thought of as the value function, where the agent's aim is to select the action, such that the mis-classification error is reduced or the classification accuracy is increased.

For supervised learning, during the training phase a knowledgeable supervisor provides the true class to the agent, which then minimizes the cost function, which is the difference between

the true class and the predicted class. However, for reinforcement learning, the agent informs the predicted class to the environment, which then evaluates the prediction and based on the assessment, provides the agent either with a reward for correct prediction or a penalty otherwise. Thus, in a way, although similarities exist between the reinforcement learning approach in the form of employing function approximation for both cases, there are also subtle differences between them; where the former technique comprises of a knowledgeable environment, the latter can operate even with a naive environment.
Learning automata based 2-class classification algorithm has been proposed in [31]. Although this research is an extension of [31], the present study depicts a learning automata based multi-class classifier from the existing binary classifier, however, the probability generating functions used in either research are entirely different. The former uses a sigmoid function as the function approximation, whereas the latter uses a softmax function.

Chapter 3

Learning Automata

Learning automata can vaguly defined as the theory that deals with the analysis and synthesis of **automata**, which operate in unknown **environments**. In this chapter the different forms of the constitutents of learning automata will be defined.

3.1 Environment

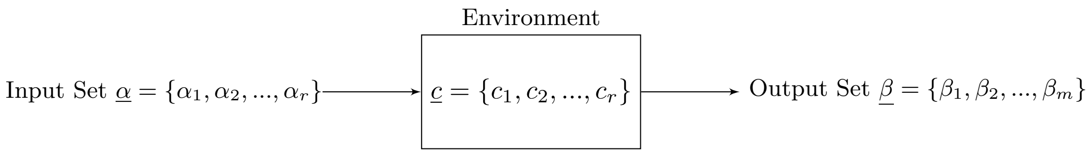
Environment
Input Set $\underline{\alpha} = \{\alpha_1, \alpha_2, ..., \alpha_r\}$
$\underline{c} = \{c_1, c_2, ..., c_r\}$
Output Set $\underline{\beta} = \{\beta_1, \beta_2, ..., \beta_m\}$

Figure 3.1: The Environment.

Unknown media in which an automaton or a group of automata can operate is referred as an environment. Mathematically, an environment is represented by a triple $\{\underline{\alpha}, \underline{c}, \underline{\beta}\}$, where $\underline{\alpha} = \{\alpha_1, \alpha_2, ..., \alpha_r\}$ represents a finite input set and $\underline{\beta} = \{\beta_1, \beta_2, ..., \beta_m\}$ represents an output set, where each $\beta_j \in [0, 1]$. The set $\underline{c} = \{c_1, c_2, ..., c_r\}$ represents the penalty probabilities, where each element $c_i \in \underline{c}$ corresponds to one input $\alpha_i \in \underline{\alpha}$.
Models of an environment, where the output can only attain two values 0 and 1, are referred to as P-Models. In such models, at any discrete time $t = n$, $n \in \{0, 1, ...\}$, $\beta(n) = 1$ is identified as an unfavourable response and $\beta(n) = 0$ a favourable response from the environment.
For an unfavourable output $\beta(n) = 1$, due to an input $\alpha(n) = \alpha_i$, the penalty probability is given by

$$P(\beta(n) = 1 | \alpha(n) = \alpha_i) = c_i \qquad \forall i \in \{1, 2, ..., r\}. \tag{3.1}$$

3.2 Automaton

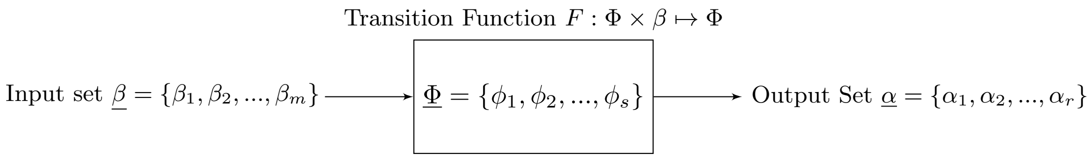
Transition Function $F : \Phi \times \beta \mapsto \Phi$
Input set $\underline{\beta} = \{\beta_1, \beta_2, ..., \beta_m\}$
$\underline{\Phi} = \{\phi_1, \phi_2, ..., \phi_s\}$
Output Set $\underline{\alpha} = \{\alpha_1, \alpha_2, ..., \alpha_r\}$

Figure 3.2: The Automaton.

The concept of an automaton, as understood in automata theory, is a very general one encompassing a wide variety of abstract systems. It is represented by a quintuple $\{\underline{\Phi}, \underline{\alpha}, \underline{\beta}, F(\cdot,\cdot), H(\cdot,\cdot)\}$, where $\underline{\Phi}$ is the set of internal states, $\underline{\beta}$ is the set of inputs, $\underline{\alpha}$ is the set of output actions, $F(\cdot,\cdot) : \underline{\Phi} \times \underline{\beta} \mapsto \underline{\Phi}$ is the transition function that maps the current state and the current input into the next state and $H(\cdot,\cdot) : \underline{\Phi} \times \underline{\beta} \mapsto \underline{\alpha}$ is the output function that maps the present state and input to the present action.
However, if the current output is only a function of the present state, then the automaton is referred to as a state-output automaton. The output function of a state output automaton is represented by $G(\cdot) : \underline{\Phi} \mapsto \underline{\alpha}$.
An automaton is called a **deterministic automaton**, if both F and G are deterministic mappings and in such cases, for a given initial state and input, the succeeding state and action are uniquely specified. If F or G is stochastic, the automaton is called a **stochastic automaton**.

3.2.1 Deterministic Automaton

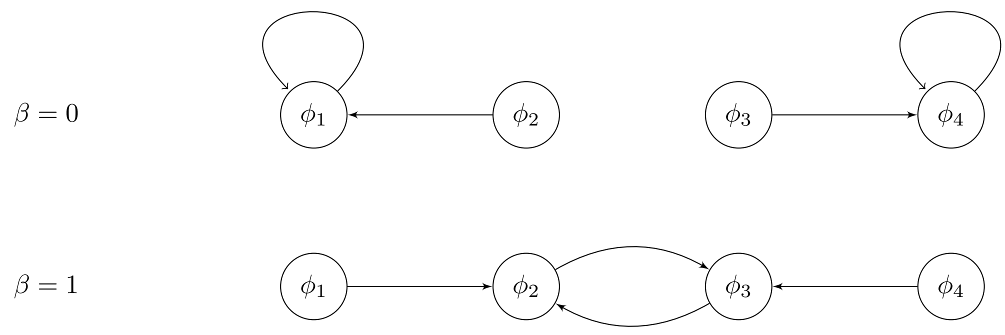

Figure 3.3: Transition graphs

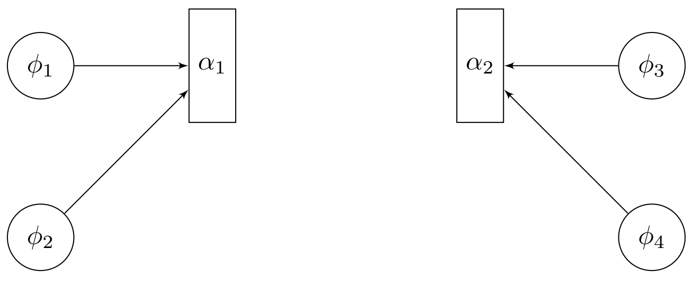

Figure 3.4: Output graph.

When the input set $\underline{\beta} = \{\beta_1, \beta_2, ..., \beta_m\}$ is finite, the mappings of a deterministic automaton can be conveniently represented either in the form of matrices or of graphs. Consider an automaton for which $\underline{\beta} = \{0, 1\}$ and $\underline{\alpha} = \{\alpha_1, \alpha_2\}$ and $\underline{\Phi} = \{\phi_1, \phi_2, \phi_3, \phi_4\}$. The transition function can be represented in terms of two matrices $F(\beta = 0)$ and $F(\beta = 1)$ where the entries $f_{ij}^{\beta(0)}$ and $f_{ij}^{\beta(0)}$ are defined as,

$$f_{ij}^{\beta} = \begin{cases} 1 & \text{if} \quad \phi_i \mapsto \phi_j \quad \text{for an input} \quad \beta, \\ 0 & \text{otherwise.} \end{cases}$$

$$F(0) = \begin{array}{c c} & \begin{array}{cccc} \phi_1 & \phi_2 & \phi_3 & \phi_4 \end{array} \\ \begin{array}{c} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \end{array} & \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \qquad F(1) = \begin{array}{c c} & \begin{array}{cccc} \phi_1 & \phi_2 & \phi_3 & \phi_4 \end{array} \\ \begin{array}{c} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \end{array} & \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Let, ϕ_1, ϕ_2 correspond to actions α_1 and ϕ_3,ϕ_4 to α_2 as shown in Figure3.4. Then the elements g_{ij} of the matrix representing the output function G can be defined as,

$$g_{ij} = \begin{cases} 1 & \text{if} \quad G(\phi_i) = \alpha_j, \\ 0 & \text{otherwise.} \end{cases}$$

Thus output matrix can be expressed as,

$$G = \begin{array}{c c} & \begin{array}{cc} \alpha_1 & \alpha_2 \end{array} \\ \begin{array}{c} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \end{array} & \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \end{array}$$

Hence, a transition function F for a deterministic automaton can be represented by m matrices of dimension $s \times s$, where m and s represents number of inputs and number of states respectively. Likewise, the output function G can be represented by an $s \times r$ matrix, where r represents the number of available actions.

3.2.2 Stochastic Automaton

In stochastic automaton the mappings of at least one of the transition functions F and output function G is stochastic.
When F is stochastic, given the present state and input, the next state is random and F gives the probabilities of reaching the various states. Thus, F can be specified by a set of conditional probability matrices $\{F(\beta_1), F(\beta_2), ..., F(\beta_m)\}$, where each $F(\beta)$,$\forall \beta \in \underline{\beta}$ is an $s \times s$ matrix. The elements of each $F(\beta)$ is given by,

$$f_{ij}^{\beta} = P\{\phi(n+1) = \phi_j \mid \phi(n) = \phi_i, \beta(n) = \beta\}, \qquad \begin{array}{l} i = 1, 2, ..., s, \\ j = 1, 2, \ldots, s, \\ \beta = \beta_1, \beta_2, ..., \beta_m. \end{array} \tag{3.2}$$

Example

From the graph as shown in Figure3.5 the selective elements of the transition function matrix F are given as,

$$f_{34}^{0} = P\{\phi(n+1) = \phi_4 \mid \phi(n) = \phi_3, \beta(n) = 0\} = 0.75.$$
$$f_{21}^{1} = P\{\phi(n+1) = \phi_1 \mid \phi(n) = \phi_2, \beta(n) = 1\} = 0.25.$$

When the output function is stochastic, the elements of conditional probability matrix G of dimension $s \times r$ representing the output function is given by,

$$g_{ij} = P\{\alpha(n) = \alpha_j \mid \phi(n) = \phi_i\}, \qquad \begin{array}{l} i = 1, 2, \cdots, s, \\ j = 1, 2, \cdots, r. \end{array} \tag{3.3}$$

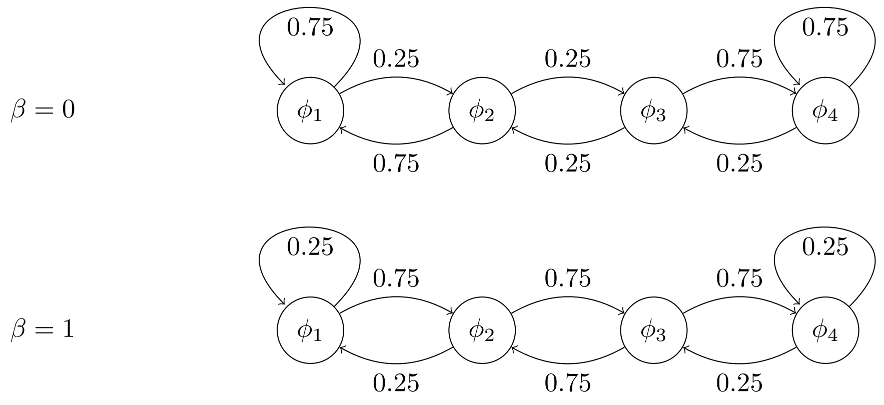

Figure 3.5: Stochastic state transition graphs

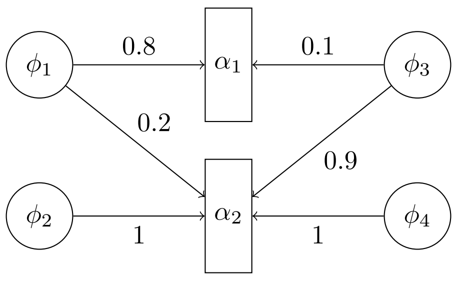

Figure 3.6: Stochastic output graph

From the output graph shown in Figure3.6, the elements of the output matrix can be calculated as follow,

$$g_{31} = P\{\alpha(n) = \alpha_1 \mid \phi(n) = \phi_3\} = 0.1.$$

Hence, to conserve probability measure we have,

$$\begin{aligned} \sum_{j=1}^{s} f_{ij}^{\beta} &= 1, \qquad && \forall \beta \in \underline{\beta}, i. \\ \sum_{j=1}^{r} g_{ij} &= 1, \qquad && \forall i. \end{aligned} \tag{3.4}$$

3.2.2.1 State and Action probabilities

At any instant n, let the state probabilities of an automaton be represented by,

$$\pi(n) = [\pi_1(n), \pi_2(n), \ldots, \pi_s(n)]^T. \tag{3.5}$$

Also,

$$\begin{aligned} \pi_j(0) &= P\{\phi(0) = \phi_j\} \\ \pi_j(n) &= P\{\phi(n) = \phi_j \mid \beta(0), \cdots, \beta(n-1)\}. \end{aligned}$$

Given the initial state probability $\pi(0)$ and the input $\beta(0)$, the state probability vector at an instant $n = 1$ is given by,

$$\begin{aligned} \pi_j(n) &= P\{\phi(1) = \phi_j \mid \beta(0)\} \\ &= \sum_{i=1}^{s} P\{\phi(1) = \phi_j \mid \phi(0) = \phi_i, \beta(0)\} P\{\phi(0) = \phi_i\} \\ &= \sum_{i=1}^{s} f_{ij}^{\beta(0)} \pi_i(0). \end{aligned} \tag{3.6}$$

The vector form of Eq.3.6 can be expressed as,

$$\pi(1) = F^T(\beta(0))\pi(0). \tag{3.7}$$

Using the chain-rule, the state vector at any instant n can be written as,

$$\pi(n) = F^T(\beta(n-1))F^T(\beta(n-2)) \cdots F^T(\beta(0))\pi(0). \tag{3.8}$$

However, it is also possible to calculate the action probability vector $p(n)$ whose i^{th} component $p_i(n)$ is given by,

$$p_i(n) = P\{\alpha(n) = \alpha_i \mid \beta(0), \ldots, \beta(n-1)\}, \qquad (i = 1, \ldots, r). \tag{3.9}$$

Also, Eq.3.9 can be expressed as,

$$\begin{aligned} p_i(n) &= \sum_{j=1}^{s} P\{\alpha(n) = \alpha_i \mid \phi(n) = \phi_j\} P\{\phi(n) = \phi_j \mid \beta(0) \ldots, \beta(n-1)\} \\ &= \sum_{j=1}^{s} g_{ji}\pi_j(n). \end{aligned} \tag{3.10}$$

The total action probability vector can be expressed as,

$$p(n) = G^T\pi(n). \tag{3.11}$$

3.2.2.2 Fixed structure and Variable structure Automata

An automaton can be classified in the set of **fixed structure automata**, if the conditional probabilities f_{ij} and g_{ij} are independent of the instant n and the input sequence.
In a real world scenario, the input $\beta(n)$ to the automaton changes with the instant n for an output $\alpha(n)$ and in such cases it is always beneficial to update f_{ij} or g_{ij} for the automaton to adapt to the system. The class of automaton where there is a provision for updating the conditional probabilities is called a **variable-structure stochastic automaton** class.

3.3 Learning Automata System

A learning automata system consists of the Environment and Automaton connected in a feedback arrangement, as shown in the block diagram in Figure3.7. The output, $\beta(n)$, of the environment forms the input to the automaton and the action(output), $\alpha(n)$, of the automaton provides input to the environment.

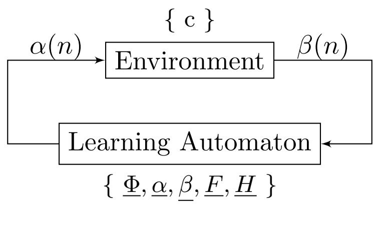
{ c }
$\alpha(n)$
Environment
$\beta(n)$
Learning Automaton
{ $\underline{\Phi}, \underline{\alpha}, \underline{\beta}, \underline{F}, \underline{H}$ }

Figure 3.7: Feedback connection of automaton and environment.

Starting from an initial state, $\phi(0)$, the automaton generates the corresponding action $\alpha(0)$, which is provided to the environment as the input. Depending on the environment output $\beta(0)$, the automaton attains the present state $\phi(1)$. This sequence of operations is repeated to result in a sequence of states, actions and responses of the learning automata system. An automaton acting in an unknown random environment that tries to improve its **performance** (in some sense), by interaction and a feedback strategy, is referred to as a **Learning Automaton**.

3.3.1 Learning Automata System Performance

At the beginning of the learning process, no prior information is available to the automaton on the basis of which the different actions $\alpha_i (i = 1, 2, \ldots, r)$ can be selected. In such scenarios, the most unbiased option of selecting an action is to choose each action with equal probability. Thus the probability of the i^{th} action is given by,

$$p_i(n) = \frac{1}{r} \qquad i = 1, 2, \ldots, r.$$

Such an automaton is called a **pure chance automaton**. To evaluate the performance of any automaton operating in an environment at an instant n, as $n \to \infty$, this can be compared to a pure chance automaton introduced in the system at that instant. However, to make such a comparison, the environment must be considered stationary, i.e. the penalty probabilities $\{c_i, c_2, \ldots, c_r\}$ have to be independent of the instant n. Let $M(n)$ represents the average penalty at any instant n for a given action probability vector. Then the average penalty at n can be expressed as

$$\begin{aligned} M(n) &= E\{\beta(n) \mid p(n)\} \\ &= P\{\beta(n) = 1 \mid p(n)\} \\ &= \sum_{i=1}^{r} P\{\beta(n) = 1 \mid \alpha(n) = \alpha_i\} P\{\alpha(n) = \alpha_i\} \\ &= \sum_{i=1}^{r} c_i p_i(n). \end{aligned} \tag{3.12}$$

For a pure chance automaton, the average penalty remains constant over time and is expressed as,

$$\begin{aligned} M(0) &= \frac{1}{r} \sum_{i=1}^{r} c_i, \\ &= M_0 \text{ (say)}. \end{aligned} \tag{3.13}$$

However, as $p(n)$, $\lim_{n\to\infty} p(n)$ are random variables and consequently, $M(n)$, $\lim_{n\to\infty} M(n)$ are also random variables, hence, for proper evaluation of any automaton, one has to compare the $E[M(n)]$ with M_0. Calculating $E[M(n)]$ gives the following,

$$\begin{aligned} E[M(n)] &= E\{E[\beta(n) \mid p(n)]\} \\ &= E[\beta(n)] \\ &= E[\sum_{i=1}^{r} c_i p_i(n)] \\ &= \sum_{i=1}^{r} c_i E[p_i(n)] \\ &= c_1 E[p_1(n)] + c_2 E[p_2(n)] + \ldots + c_r E[p_r(n)]. \end{aligned} \tag{3.14}$$

Definition 1.1: A learning automaton is said to be **expedient** *iff* ,

$$\lim_{n\to\infty} E[M(n)] < M_0. \tag{3.15}$$

From Eq3.12,

$$\begin{aligned} \inf M(n) &= \inf_{p(n)} \{\sum_{i=1}^{r} c_i p_i(n)\} \\ &= \min_i \{c_i\} \qquad \Big[\because \sum_{i=1}^{r} p_i(n) = 1\Big] \\ &\triangleq c_l. \end{aligned} \tag{3.16}$$

Definition 1.2: A learning automaton is said to be **optimal** *iff*,

$$\lim_{n\to\infty} E[M(n)] = c_l \quad \text{where,} \quad c_l = \min_i\{c_i\}. \tag{3.17}$$

Definition 1.3: A learning automaton is said to be **absolutely expedient** *iff*,

$$\begin{aligned} E[M(n+1) \mid p(n)] < M(n) \qquad & \forall n, \\ & p_i(n) \in [0,1], \\ & \forall\{c_i\}(i = 1, 2, \ldots, r). \end{aligned} \tag{3.18}$$

Taking the expectation of Eq.3.18 gives ,

$$E[E[M(n+1) \mid p(n)]] = E[M(n+1)],$$

Therefore
$$E[M(n+1)] < E[M(n)]. \tag{3.19}$$

Thus Eq.3.19 shows that $E[M(n)]$ is strictly monotonically decreasing with n in all stationary environments.

3.4 Reinforcement Schemes: A brief introduction

In order to make an automaton expedient, rules must be specified for updating the probability functions of the feedback based strategy of the LA systems that would result in $E[M(n)]$ attaining its minimum or maximum value depending on the nature of the problem.
The simplest idea behind the reinforcement scheme for updating the action probabilities for any action α_i, selected by an automaton at an instant n, which received a favorable input $\beta(n) = 0$ from the environment, is that the action probability $p_i(n)$ is increased such that $p_i(n+1) \geq p_i(n)$ and all other components of $p(n)$ are decreased. For an unfavorable input from the environment, i.e. $\beta(n) = 1$ for an action $\alpha(n) = \alpha_i$, the probability $p_i(n)$ is decreased such that $p_i(n+1) < p_i(n)$ and all other components of $p(n)$ are increased. These increases and decreases in $p_i(n)$ are called reward and penalty respectively. If $p(n+1)$ is a linear function of $p(n)$, then such a reinforcement scheme is said to be linear; otherwise it is termed as nonlinear.
A reinforcement scheme where $p(n+1)$ is linearly dependent on $p(n)$ and the update scheme follows the reward and penalty model, is collectively referred to as a **Linear Reward-Penalty** scheme. In certain linear schemes, where, due to an unfavorable input from the environment ($\beta(n) = 1$), the action probabilities for the instant $(n+1)$ are retained at their previous values, this is referred to as a **Linear Reward-Inaction** scheme.
With a similar proposition, the state transition probabilities can be updated. For instance, if $\phi(n) = \phi_i$, $\phi(n+1) = \phi_j$ and $\beta(n) = \beta$, then f_{ij}^{β} is increased if $\beta = 0$ and decreased when $\beta = 1$. To preserve the probability measure, that $\sum_{j=1}^{s} f_{ij}^{\beta} = 1$, for all $i \in (1, 2, \ldots, s)$, all other elements of the state transition matrix (F) corresponding to the i^{th} row must be changed in the opposite fashion.
However, to reduce the complexity of the reinforcement schemes, it can be assumed that each state corresponds to a distinct action and hence $r = s$. This assumption reduces the evaluation of the number of functionals at an instant n, as the update of either the state transition function or the action probabilities will then represent the entire scenario of the feedback based LA system.

3.4.1 General Reinforcement scheme

A general update scheme, at an instant n, for a variable structure automaton operating in a stationary P-Model environment, can be represented as follows:

If

$$\alpha(n) = \alpha_i \quad (i = 1, 2, \dots, r),$$

- Case I: When $\beta(n) = 0$

$$\begin{aligned} p_j(n+1) &= p_j(n) - g_j[p(n)] \quad \forall j \neq i, \\ p_i(n+1) &= p_i(n) + \sum_{\substack{j=1 \\ j\neq i}}^{r} g_j[p(n)]. \end{aligned} \tag{3.20}$$

- Case II: When $\beta(n) = 1$

$$\begin{aligned} p_j(n+1) &= p_j(n) + h_j[p(n)] \quad \forall j \neq i, \\ p_i(n+1) &= p_i(n) - \sum_{\substack{j=1 \\ j\neq i}}^{r} h_j[p(n)]. \end{aligned} \tag{3.21}$$

The functions g_j and h_j $\forall j \in (1, 2, \dots, r)$, as shown in Eq.3.20 and Eq.3.21, are referred to as reward and penalty functions respectively.
The following section highlights the description of the process of representing the updating schemes of an action probability sequence using a discrete-time Markov process.

3.4.2 Reinforcement Schemes as Markov Process

The vector $p(n)$ defined in Eq.3.20 and Eq.3.21 is a random vector. If the automaton is assumed to be operating in a stationary environment, i.e. $c_i \forall i \in (1, 2, \dots, r)$, and the reward-penalty functions g and h are independent of the stage number n, the probability $p(n+1)$ is determined entirely by $p(n)$ and hence $\{p(n)\}_{n\geq 0}$ is a discrete-time homogeneous Markov process.
At every instant n, the elements of $p(n) \in [0, 1]$ and $\sum_{i=1}^{r} p_i(n) = 1$. Hence the unit simplex

$$S_r \triangleq \{p \mid p^T = [p_1, p_2, \dots, p_r], 0 \leq p_i \leq 1, \sum_{i=1}^{r} p_i = 1\} \tag{3.22}$$

represents the state space of the process $\{p(n)\}_{n\geq 0}$. The interior where all $p_i \in (0, 1)$ is denoted as S_r^0.
Let e_i represent the r-dimensional unit vector, where the i^{th} element is unity and is represented as $e_i^T \triangleq [0, 0, \dots, 1, 0, 0]$. Hence e_i is the vertex of the simplex S_r. Let the set of all vertices of S_r be represented by V_r such that

$$V_r \triangleq \{e_1, e_2, \dots, e_r\}. \tag{3.23}$$

A state $p^* \in S_r$ is called an ***absorbing state*** if $p(n) = p^* \Rightarrow p(k) = p^*$ with probability 1 (w.p.1) $\forall k \geq n$. The reward and penalty function as shown in Eq.3.20 and Eq.3.21 can be chosen in a manner so that reinforcement schemes have one or more absorbing states. Such an updating algorithm is referred as an ***absorbing algorithm***. Updating algorithms that are devoid of *absorbing states* are called ***non-absorbing algorithms***. The L_{R-P} reinforcement scheme falls under the category of non-absorbing algorithm and L_{R-I}, is in the class of absorbing algorithm.
In Section 3.4 a brief introduction to the Linear Reward-Penalty (L_{R-P}) scheme has been highlighted and the following section will discuss in detail the reward and penalty functions, as well as the other parameters for performance evaluation of the scheme.

3.4.3 Linear Reward-Penalty Scheme

For mathematical simplicity, the automaton considered for analysis of the L_{R-P} scheme is considered to have only two actions. However, in a similar analysis to previously, it can be extended to multi-action automata as well.
Let the reward and penalty function be represented by

$$\begin{aligned} g_j(p(n)) &= ap_j(j) \\ \text{and} \quad h_j(p(n)) &= b(1-p_j(n)), \end{aligned} \tag{3.24}$$

where, $0 < a < 1$ and $0 < b < 1$ is referred to as the reward-penalty parameter.When $a = b$, the reinforcement scheme is referred as symmetric L_{R-P} scheme. For mathematical simplicity, only symmetric L_{R-P} will be analyzed in the present section, however the same can be carried out for a general L_{R-P} scheme in similar manner. For the instance when $b = 0$, L_{R-P} scheme boils down to L_{R-I} reinforcement scheme.
Substituting the reward and penalty functions shown in Eq.3.24 for $b = a$ into Eq.3.21 yields the following.
If

$$\alpha(n) = \alpha_1,$$

- Case I: When $\beta(n) = 0$

$$\begin{aligned} p_1(n+1) &= p_1(n) + a(1-p_1(n)), \\ p_2(n+1) &= (1-a)p_2(n). \end{aligned} \tag{3.25}$$

- Case II: When $\beta(n) = 1$

$$\begin{aligned} p_1(n+1) &= (1-a)p_1(n), \\ p_2(n+1) &= p_2(n) + a(1-p_2(n)). \end{aligned} \tag{3.26}$$

In a similar fashion the equations for updating the action probabilities can be expressed when $\alpha(n) = \alpha_2$.
By investigating the asymptotic behavior of the action probabilities of an automaton following the L_{R-P} scheme, it can proved that the scheme is expedient. To model the environment where the automaton is present, it is assumed that the penalty probabilities are $\{c_1, c_2\}$. A common method to investigate the asymptotic behavior of the action probabilities is to calculate the conditional expectation of $p_i(n+1)$, given $p(n)$. Hence for the L_{R-P} scheme,

$$\begin{aligned} E[p_1(n+1) \mid p(n)] &= [p_1(n) + a(1-p_1(n))][p_1(n)(1-c_1) + p_2(n)c_2] \\ &+ [(1-a)p_1(n)][p_1(n)c_1 + p_2(n)(1-c_1)] \\ &= [1 - a(c_1+c_2)]p_1(n) + ac_2. \end{aligned} \tag{3.27}$$

Taking the expectation of both the sides of Eq.3.27

$$E[p_1(n+1)] = [1 - a(c_1+c_2)]E[p_1(n)] + ac_2. \tag{3.28}$$

The quantity in Eq.3.28 represents a linear difference equation and its general solution can be expressed as

$$E[p_1(n)] = [1 - a(c_1+c_2)]^n p_1(0) + \frac{ac_2}{a(c_1+c_2)}. \tag{3.29}$$

Hence,

$$\lim_{n\to\infty} E[p_1(n)] = \frac{c_2}{c_1+c_2} \qquad \text{if } [1 - a(c_1+c_2) < 1]. \tag{3.30}$$

Similarly, it can be shown that

$$\lim_{n\to\infty} E[p_2(n)] = \frac{c_2}{c_1+c_2}. \tag{3.31}$$

From Eq.3.14

$$\begin{aligned}\lim_{n\to\infty} E[M(n)] &= c_1 \lim_{n\to\infty} E[p_1(n)] + c_2 \lim_{n\to\infty} E[p_2(n)] \\ &= \frac{c_1c_2}{c_1+c_2} + \frac{c_1c_2}{c_1+c_2} \\ &= 2\frac{c_1c_2}{c_1+c_2}.\end{aligned} \tag{3.32}$$

And from Eq.3.13

$$\begin{aligned}M_0 &= \frac{1}{2}\sum_{i=1}^{2} c_i \\ &= \frac{c_1+c_2}{2}.\end{aligned} \tag{3.33}$$

Hence, $E[M(n)] < M_0$ as $n \to \infty$ and $c_1 \neq c_2 \neq 0$. Thus from the above analysis it can be concluded that the L_{R-P} scheme is expedient.

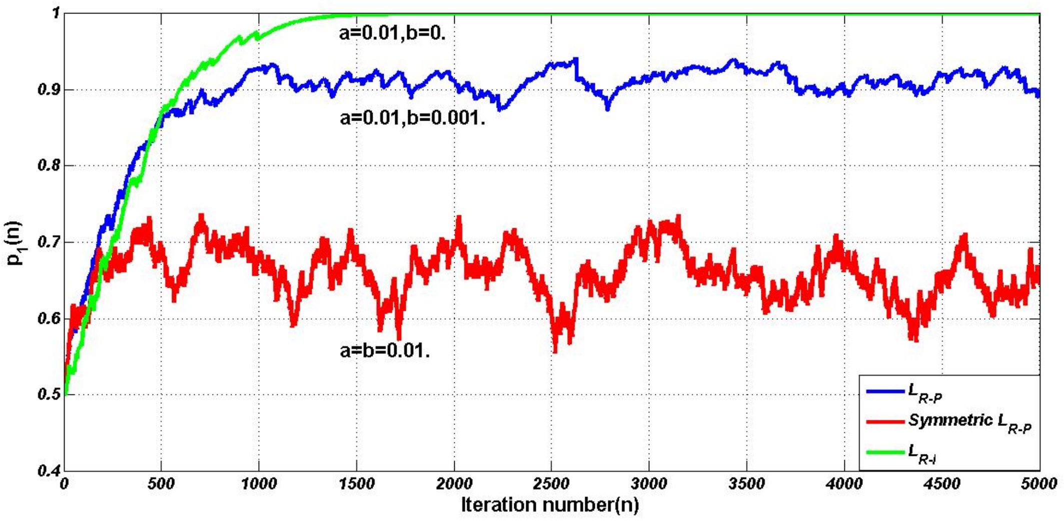

Figure 3.8: Probability of α_1 versus trial instance

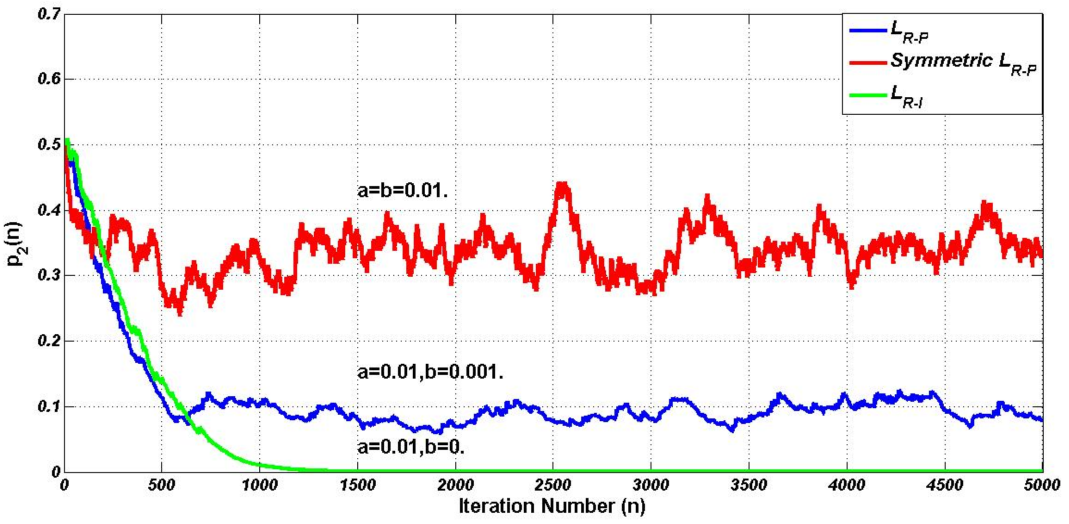

Figure 3.9: Probability of α_2 versus trial instance

From Eq.3.25 and Eq.3.26 it can be observed that $\{p(n)\}_{n\geq 0}$ is a discrete-time Markov process

defined on the state space which represented by a unit simplex $S_2 : \{p \mid p_1+p_2 = 1, p_1, p_2 \geq 0\}$. Since $p_1(n)$(or $p_2(n)$) changes, depending on the environment response due to the selection of either α_1 or α_2, hence $p(n) = e_i$ does not guarantee $p(k) = e_i \; \forall k > n$, provided $c_i \neq 0$, $c_j \neq 1 \; \forall i, j \in \{1, 2\}$ and $i \neq j$, where $e_1^T = [1, 0]$ and $e_2^T = [0, 1]$. Thus, the L_{R-P} reinforcement scheme is a non-absorbing algorithm, as it is devoid of absorbing states and this will be demonstrated with a simple simulation experiment in the following.
The simulation experiment considers an automaton consisting of two actions $\alpha = \{\alpha_1, \alpha_2\}$ and operating in a stationary environment characterized by the penalty probabilities $c = \{0.4, 0.8\}$. The action probabilities, $p_1(n)$ and $p_2(n)$ versus the iteration instances for L_{R-P}, symmetric L_{R-P} and L_{R-I} reinforcement schemes are shown in Figure3.8 and Figure3.9 respectively. It can be observed from the figures that as the learning progresses, the probability of selecting the first action($p_1(n)$) increases and the probability of second action($p_2(n)$) decreases. This is because of the fact that the environment generates a lesser penalty for the former over the latter. Neither of the L_{R-P} schemes have absorbing states (theoretically this has been explained earlier) which is also evident from Figure3.8 and Figure3.9. However, for the L_{R-I} scheme, as learning progresses, the probability of selecting α_1 attains a maximum value of 1 and that of α_2 a possible minimum value of 0, which confirms that the scheme has absorbing states. Since, the L_{R-I} scheme has absorbing states, hence its performance is better over the L_{R-P} scheme when operating in a stationary environment. However, the absorbing algorithms e.g. the L_{R-I} reinforcement scheme are inexpedient for non-stationary environments, as for these algorithms once the absorbing state is attained, the action probabilities remain unchanged even with the change in the environment penalty probabilities. Further details on the topics explained in this Section can be found in [10, 27–29].

3.5 Non-stationary Environments

Earlier linear reinforcement schemes for an automaton operating in a stationary environment have been described. As mentioned in the preceding section, an environment is said to be stationary if the penalty probability c_i for an action α_i $(i = 1, 2, \ldots, r)$ is invariant of the stage n. In the present section, the reinforcement scheme for an automaton operating in a non-stationary environment will be detailed. An environment is referred to as non-stationary if the penalty probability c_i corresponding to any action α_i $(i = 1, 2, \ldots, r)$ varies with time (or equivalently stage number n). Thus, as an environment response changes, the ordering of the automaton actions with respect to the performance criterion $(E[M(n)])$ may vary. If a learning automaton operates with strategies mentioned earlier in such an environment, it may become less expedient and even inexpedient. For modeling, the simplest non-stationary environment is to consider that $c(n)$ can assume s number of values, where $s \in \mathbb{Z}$ and $s < \infty$. This implies that the automaton operates in one of a finite set of stationary environments $E_i(i = 1, 2, \ldots, s)$, where E_i represents the i^{th} state of the environment E. Assuming that the automaton A comprises of s sub-automata, each sub-automaton A_i is deployed in a stationary environment E_i and updates its action probabilities based on the response of the environment corresponding to an action $\alpha_j^i(j = 1, 2, \ldots, r)$. Further, it is also assumed that each sub-automaton has an identical reinforcement scheme and the automaton is aware of the specific environment into which it is operating at any instant.
The assumptions and the methodology mentioned earlier decompose the variability of a non-stationary environment into a finite number of stationary environments. However, the process suffers from two principal drawbacks.

- Assigning a separate automaton A_i for every environment E_i is practically infeasible for very large s. Further, since the action probabilities of an automaton A_i are updated if and only if it is operating in an environment E_i, an insufficient number of switching instances in the environment E_i may lead the action probabilities of A_i not to converge.

- It has been assumed that the automaton is aware of the environment E_i, in which it is operating at any instant. This assumption is not pragmatic and in many real world scenarios this requirement may not be satisfied.

Keeping in mind the drawbacks of the proposed methodology, a learning algorithm will be highlighted in the Chapter 4 to circumvent these disadvantages.

Chapter 4

Learning Automata based classifiers

In Chapter 3 the basic models of learning automata, pertaining to optimal action selection, have been mentioned. Also, it has been highlighted that those models are insufficient for tackling non-stationary environments. In this chapter, models that can handle non-stationary environments will be discussed. It will also be argued that these learning automata based models, which have evolved for tackling non-stationary environments, are indeed suitable for classification tasks as well.

4.1 Parameterized Stochastic Learning

In Chapter 3, the drawbacks of decoupling the scenario where an automaton operating in a non-stationary environment, with multiple automata acting in a finite number of stationary environments, has been presented. In this section, an algorithm is presented, based on parameterizing the action probabilities and constructing mapping vectors representing the environment states. At any instant n the environment provides the automaton an input $x(n) \in X$, where $x(n) \in \mathbb{R}^l$. $x(n)$ is called a **context** or **feature vector** and X, the **context** or **feature space** of the environment E. The automaton selects an action $\alpha(n) \in \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$, which is provided as input to the environment. The probability of selecting an action $\alpha(n) = \alpha_i (i = 1, 2, \ldots, r)$ for a given context $x(n)$, at an instant n, is given by $p_i(x(n), n)$. Based on the action $\alpha(n)$ and the context vector $x(n)$, the environment sends a reinforcement signal $\beta(n) \in \{0, 1\}$, to the automaton. The process of associating an action with the environment context, is termed as **associative reinforcement learning**. The penalty probability for an associative reinforcement learning unit can be expressed as

$$c_{ij} = Pr\{\beta(n) = 1 \mid x(n) = x_i, \alpha(n) = \alpha_j\} \quad \begin{array}{l} i \in \{1, 2, \ldots, s\}, \\ j \in \{1, 2, \ldots, r\}. \end{array} \tag{4.1}$$

The aim of the learning process is to respond to each context vector x_i, with an action α_j, with probability 1, which can be achieved using some reinforcement scheme. A common updating algorithm, based on the reward-penalty model, for the associative reinforcement learning process of a two action automaton, is often referred to as an **associative reward-penalty**, (A_{R-P}), scheme. In the following section, a detailed analysis of the A_{R-P} scheme will be provided, explaining the process of making an automaton, operating in a non-stationary environment, expedient.

4.2 Associative Reward-Penalty Scheme

As mentioned earlier, a reinforcement scheme based on contextual information and the reward-penalty model for a two action automaton is commonly termed as an associative reward-penalty(A_{R-P}) scheme. The context space X, of the non-stationary environment E, is the union of all the context vectors. Mathematically, the context space can be expressed as $X = \cup_{i=1}^{s} x_i$, where $x_i \in \mathbb{R}^l$ and s represents the number of states of the environment. The set of actions of an automaton at any instant n is represented by $\{\alpha_1, \alpha_2\}$ and the probability of each action is represented by $p(\alpha_j \mid x_i)$. To conserve the probability measure,

$$\sum_{j=1}^{2} p(\alpha_j \mid x_i) = 1 \qquad \forall i \in \{1, 2, \dots, s\}. \tag{4.2}$$

Let a vector $\theta \in \mathbb{R}^l$ represent the internal state of the automaton and is termed as a **parameter vector**. The probabilities of the actions at any instant n are calculated based on the value of $\theta(n)$, using a function $f(\theta(n), x(n) = x_i)$, which is called a **parameterizing function**. The parameterizing function is defined as,

$$f(\theta, x) = \theta^T x, \tag{4.3}$$

such that $f(\theta, x) \in [-1, 1]$.
At any instant n, given the context vector $x(n) = x_i$ the selection of any action $\alpha(n)$ is expressed as

$$\begin{aligned} &p(\alpha(n) = \alpha_1 \mid x_i) = 1 \quad \text{and} \quad p(\alpha(n) = \alpha_2 \mid x_i) = 0 \quad \text{if} \quad f(\theta(n), x(n) = x_i) + \eta(n) > 0 \\ &p(\alpha(n) = \alpha_1 \mid x_i) = 0 \quad \text{and} \quad p(\alpha(n) = \alpha_2 \mid x_i) = 1 \quad \text{if} \quad f(\theta(n), x(n) = x_i) + \eta(n) \leq 0, \end{aligned} \tag{4.4}$$

where $\eta(n)$ are independent and identically distributed (i.i.d.) random variables, within an interval $[-1, 1]$, with a known distribution Ψ. Although the utility of the distribution function Ψ may not be clear at this juncture, however is it will be clear at the later part of this segment. At this point it should be borne in mind that since Ψ is a part of the automaton rather than the environment, hence it is a known function.
If it is assumed that there exists at least a $\theta^* \in \theta$, such that the hyperplane defined by ${\theta^*}^T x = 0$ divides the context space X into two regions, such that α_1 is the optimal action when ${\theta^*}^T x > 0$ and α_2 is the optimal action when ${\theta^*}^T x \leq 0$. In order to search θ^*, there is a necessity to design an algorithm using the feedback based strategy of the learning unit.
Let

$$\alpha(n) = \begin{cases} 1 & \text{if} \quad \alpha(n) = \alpha_1, \\ -1 & \text{if} \quad \alpha(n) = \alpha_2. \end{cases} \tag{4.5}$$

At each instant the automaton selects an action $\alpha(n) = \alpha_i (i = 1, 2)$, based on the decision shown in Eq.4.4 and the environment sends back a feedback signal $\beta(n) \in \{-1, 1\}$, where $\beta(n) = 1$ denotes a favorable response and unfavorable, otherwise.

In the remaining part of this section, the updating algorithm of the A_{R-P} scheme will be highlighted and its similarities with the L_{R-P} algorithm, by assigning Ψ with a specific distribution, will be depicted.
The reinforcement algorithm for the A_{R-P} scheme is expressed as

$$\begin{aligned} &\theta(n+1) = \theta(n) - \rho(n)\{E[\alpha(n) \mid \theta(n), x(n)] - \beta(n)\alpha(n)\}x(n) \qquad \text{when} \quad \beta(n) = 1, \\ &\theta(n+1) = \theta(n) - \lambda\rho(n)\{E[\alpha(n) \mid \theta(n), x(n)] - \beta(n)\alpha(n)\}x(n) \quad \text{when} \quad \beta(n) = -1. \end{aligned} \tag{4.6}$$

where $0 < \lambda \leq 1$, ρ is a decaying step size, s.t. $\rho \geq 0$, $\sum_n \rho(n) = \infty$ and $\sum_n \rho(n)^2 \leq \infty$. From Eq.4.4, Eq.4.5 and Eq.4.6 it can be seen that for $E[\eta(n)] = 0$, if $\theta^T x > 0$, then action

$\alpha(n) = \alpha_1$ is the probable action and $\alpha(n) = \alpha_2$ otherwise. Now let us consider distribution function Ψ is as follows

$$\Psi(r) = Pr\{\eta(n) \leq r\} = \begin{cases} \frac{r+1}{2} & -1 \leq r \leq 1 \\ 0 & \text{otherwise.} \end{cases} \tag{4.7}$$

From Eq.4.7 and Eq.4.5,

$$\begin{aligned} p(\alpha(n) = -1) &= Pr\{\theta(n)^T x(n) + \eta(n) \leq 0\} \\ &= \Psi(-\theta(n)^T x(n)). \end{aligned} \tag{4.8}$$

and

$$\begin{aligned} p(\alpha(n) = 1) &= Pr\{\theta(n)^T x(n) + \eta(n) > 0\} \\ &= 1 - \Psi(-\theta(n)^T x(n)). \end{aligned} \tag{4.9}$$

From Eq.4.8 and Eq.4.9

$$\begin{aligned} E[\alpha(n) \mid \theta(n), x(n)] &= -1.p(\alpha(n) = -1) + 1.p(\alpha(n) = 1) \\ &= -\Psi(-\theta(n)^T x(n)) + 1 - \Psi(-\theta(n)^T x(n)) \\ &= 1 - 2\Psi(-\theta(n)^T x(n)) \\ &= 1 - 2\left[\frac{-\theta(n)^T x(n) + 1}{2}\right] \\ &= \theta(n)^T x(n). \end{aligned} \tag{4.10}$$

Now, consider the scenario when the action chosen by an automaton $\alpha(n) = \alpha_1$ and the environment response $\beta(n) = 1$, at any instant n, then from Eq.4.9

$$\begin{aligned} p(\alpha(n+1) = 1) &= 1 - \Psi(-\theta(n+1)^T x(n)) \\ &= 1 - \frac{1 + (-\theta(n+1)^T x(n))}{2} \\ &= \frac{1 + \theta(n)^T x(n) - \rho(n)\|\hat{x}\|^2(\theta(n)^T x(n) - 1)}{2} \qquad \text{from Eq.4.6)}, \\ &= 1 - \frac{1 - \theta(n)^T x(n)}{2} + \frac{\rho(n)\|\hat{x}\|^2(1 - \theta(n)^T x(n))}{2} \\ &= 1 - \Psi(-\theta(n)^T x(n)) + \rho(n)\|\hat{x}\|^2 \Psi(-\theta(n)^T x(n)) \\ &= 1 - \Psi(-\theta(n)^T x(n)) + \rho(n)\|\hat{x}\|^2 \Psi(-\theta(n)^T x(n)) + \rho(n)\|\hat{x}\|^2 - \rho(n)\|\hat{x}\|^2 \\ &= 1 - \Psi(-\theta(n)^T x(n)) - \rho(n)\|\hat{x}\|^2(1 - \Psi(-\theta(n)^T x(n)) + \rho(n)\|\hat{x}\|^2 \\ &= p(\alpha(n) = 1) + \rho(n)\|\hat{x}\|^2(1 - p(\alpha(n) = 1)). \end{aligned} \tag{4.11}$$

So, for $0 \leq \rho(n)\|\hat{x}\|^2 \leq 1$ in Eq.4.11, the A_{R-P} algorithm reduces to the L_{R-P} scheme, as shown in Eq.3.25 and Eq.3.26. However, for a different and specific distribution of Ψ and choosing $\lambda = 0$ in Eq.4.6, it can be shown that the A_{R-P} algorithm boils down to L_{R-I} algorithm.

Since the A_{R-P} algorithm use a step size $\rho(n)$, which asymptotically decays to zero, i.e. $\rho(n) \to 0$ as $n \to \infty$ and hence is not suitable for online adaptation to problems, where there could be slower variation of parameters than estimated. Hence, the use of algorithms with a constant step size is necessary in such real-time scenarios, where there is a requirement for online adaptation. Another limitation of the A_{R-P} algorithm is that the reinforcement scheme is feasible only for a two-action automaton, which may not be true in practice, where an automaton is required to have multiple actions. Further details of the A_{R-P} reinforcement scheme and its potential applications can found in [11].

In the following section, a version of associative learning will be considered, involving a constant step size in the reinforcement scheme, which is appropriate for online adaptation.

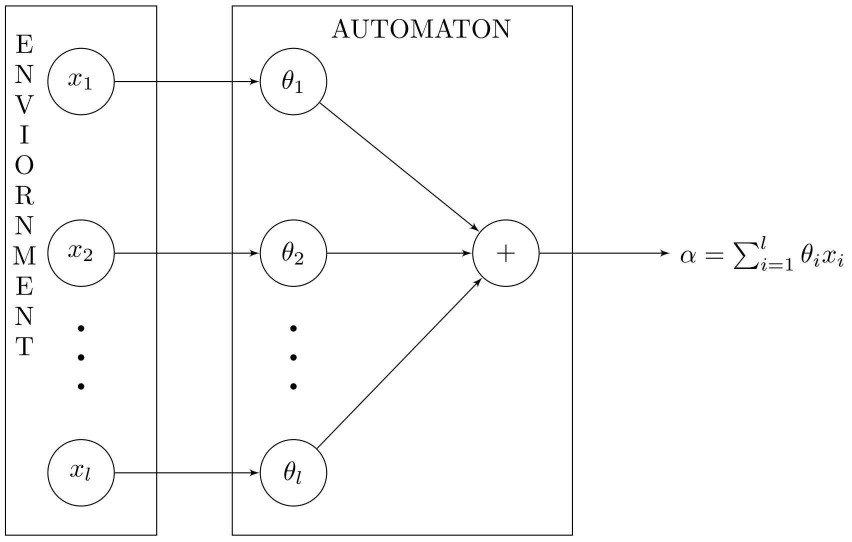

Figure 4.1: A LA system comprising of a single learning unit

4.3 REINFORCE Algorithm

In earlier sections it has been mentioned that the aim of all reinforcement schemes is to minimize (or maximize) the expected penalty (or reward), by selecting an appropriate action at every instant n. In the preceding section, the A_{R-P} reinforcement scheme has been described. However, the algorithm suffers from a major problem, that it is not appropriate for online adaptation. In this section, a reinforcement scheme, suitable for handling the environment context and also capable of online adaptation, will be described. A generalized REINFORCE updating scheme is usually explained by using a feedforward network, consisting of multiple learning agents. However, for the sake of simplicity and since a feedforward network comprising of a single learning agent is enough to serve our requirement, the REINFORCE algorithm will be explained using a single learning automaton only.

Let $g(\zeta, \theta, x) = P\{\alpha(n) = \zeta \mid \theta, x\}$ represent the probability mass function that determines the probability of selecting an α for the automaton at any instant n, depending on the parameters and the incoming environment context. Based on the automaton output $\alpha(n)$ at any instant n, the environment evaluates it and sends a reinforcement signal β to the automaton. The probability mass function for determining the automaton output is often called the probability generating function and the environment generated reinforcement signal is termed the scalar reinforcement signal.

The probability generating function is assumed to be a Bernoulli distribution which is expressed as

$$g(\zeta, \theta, x) = \begin{cases} 1-p & \text{if} \quad \zeta = 0, \\ p & \text{if} \quad \zeta = 1, \end{cases} \tag{4.12}$$

where the quantity p is computed in the following manner,

$$\begin{aligned} p &= f(s), \\ s &= \theta^T x = \sum_{i=1}^{l} \theta_i x_i. \end{aligned} \tag{4.13}$$

This special case of probability generating function, where the output values are 0 and 1, depending on the probabilities defined in Eq.4.12, is called the Bernoulli semi-linear unit. The function f defined in Eq.4.13 is called a *squashing* function and a commonly used version of this, is a logistic function defined by

$$f(s) = \frac{1}{1+e^{-s}}. \tag{4.14}$$

At every instant n the automaton updates its internal state represented by θ depending on the environment generated scalar reinforcement β. The updating scheme is defined as

$$\theta_i(n+1) = \theta_i(n) + a(\beta(n) - b_i)e_i, \tag{4.15}$$

where a is a learning rate factor, b_i is the reinforcement baseline and $e_i = \frac{\partial}{\partial \theta_i}\ln g$ is called the characteristic eligibility of θ_i. The learning rate, as in earlier reinforcement schemes, is $0 < a \leq 1$. The update scheme name REINFORCE is an acronym for "**RE**ward **I**ncrement = **N**onnegative **F**actor * **O**ffset **R**einforcement * **C**haracteristic **E**ligibility". From Eq.4.12 the characteristic eligibility e_i can be represented as

$$\frac{\partial}{\partial p}\ln g = \begin{cases} \frac{1}{p} & \text{if} \quad \alpha(n) = 1, \\ -\frac{1}{1-p} & \text{if} \quad \alpha(n) = 0. \end{cases} \tag{4.16}$$

where $p \neq 0$ and $p \neq 1$. The characteristic eligibility can be expressed in a generalized form as

$$\frac{\partial}{\partial p}\ln g = \frac{\alpha(n) - p}{p(1-p)}. \tag{4.17}$$

From Eq.4.13

$$\begin{aligned} \frac{dp}{ds} &= \frac{df(s)}{ds} = f'(s) \quad \text{and} \quad , \\ \frac{\partial s}{\partial \theta_i} &= x_i, \end{aligned}$$

Therefore

$$dp = f'(s)ds = f'(s)x_i\partial\theta_i. \tag{4.18}$$

Substituting Eq.4.18 in Eq.4.16

$$\begin{aligned} \frac{1}{f'(s)x_i}\frac{\partial}{\partial \theta_i}\ln g &= \frac{\alpha(n) - p}{p(1-p)}, \\ \implies \frac{\partial}{\partial \theta_i}\ln g &= \frac{\alpha(n) - p}{p(1-p)} f'(s)x_i. \end{aligned} \tag{4.19}$$

The derivative of the function $f(s)$ can be calculated using Eq.4.14 and is as follows

$$\begin{aligned} p = f(s) &= \frac{1}{1+e^{-s}}, \\ \implies 1 - p &= \frac{1 + e^{-s} - 1}{1+e^{-s}} = \frac{e^{-s}}{1+e^{-s}} = e^{-s}p. \end{aligned}$$

Let

$$1 + e^{-s} = z \implies -e^{-s}ds = dz.$$

Therefore

$$f(s) = \frac{1}{z} \quad \text{implies} \quad f'(z) = -\frac{1}{z^2},$$

Hence

$$\begin{aligned} f'(s) &= -\frac{-e^{s}}{(1+e^{-s})^2} = \frac{1}{(1+e^{-s})}.\frac{e^{-s}}{(1+e^{-s})}, \\ &= p(1-p). \end{aligned} \tag{4.20}$$

Substituting the derivative of $f(s)$ from Eq.4.20 in Eq.4.18

$$\begin{aligned} \frac{\partial}{\partial \theta_i}\ln g &= \frac{p(\alpha(n) - p)(1-p)}{p(1-p)}x_i, \\ &= (\alpha(n) - p)x_i. \end{aligned} \tag{4.21}$$

Substituting the value of the character eligibility e_i of Eq.4.21 in Eq.4.15

$$\Delta\theta_i(n) = a(\beta(n) - b_i)(\alpha(n) - p)x_i, \tag{4.22}$$

where $\Delta\theta_i(n) = \theta_i(n+1) - \theta_i(n)$. It can be shown that the unbiased estimate of $\frac{\partial E\{\beta|\theta\}}{\partial\theta_i}$ equals the quantity $[(\beta(n) - b_i)(\alpha(n) - p)x_i]$, and hence from Eq.4.22 it implies that the update of the automaton internal states θ_i occurs in a manner, such that the performance measure is increased. A detailed analysis of the REINFORCE algorithm can be found in [30]. However, the REINFORCE algorithm shows unbounded behavior when there are no local maxima of $E[\beta \mid \theta]$. This is explained in the following using a simple example.

Example Consider an automaton unit interacting with the environment that has a set of context vectors $x = \{x_1, x_2\}$, where $x_1 = (0,1)^T$ and $x_2 = (1,0)^T$ and where the context vectors are assumed to arrive with equal probabilities. Also the set of actions of the automaton is represented as $y = \{\alpha_1, \alpha_2\}$ and the set of automaton internal states is $\theta = \{\theta_1, \theta_2\}$. Finally, the expected value of the reinforcement is given as

$$d(\alpha_1, x_1) = d(\alpha_2, x_2) = 1 - d(\alpha_2, x_1) = 1 - d(\alpha_1, x_2) = 0.9.$$

From Eq.4.12, Eq.4.13 and Eq.4.14, the probability generating function can be expressed as

$$\begin{aligned} g(\alpha_1, \theta, x) &= \frac{1}{1 + e^{-\theta^T x}}, \\ g(\alpha_2, \theta, x) &= 1 - \frac{1}{1 + e^{-\theta^T x}}. \end{aligned} \tag{4.23}$$

Given the expected reinforcement, then the expected scalar reinforcement signal (SRS) from the environment can be expressed as

$$\begin{aligned} E[\beta \mid \theta] &= \frac{1}{2}\Big(E[\beta \mid \theta, x_1] + E[\beta \mid \theta, x_2]\Big) \\ &= \frac{1}{2}\Big(E[\beta \mid \theta, x_1, \alpha_1]g(x, \alpha_1, \theta) + E[\beta \mid \theta, x_1, \alpha_2]g(x, \alpha_2, \theta) \\ &\quad + E[\beta \mid \theta, x_2, \alpha_1]g(x, \alpha_1, \theta) + E[\beta \mid \theta, x_2, \alpha_2]g(x, \alpha_2, \theta)\Big) \\ &= \frac{1}{2}\Big(\frac{0.9}{1 + e^{-\theta_2}} + 0.1[1 - \frac{1}{1 + e^{-\theta_2}}] + \frac{0.1}{1 + e^{-\theta_1}} + 0.9[1 - \frac{1}{1 + e^{-\theta_1}}]\Big) \\ &= \frac{1}{2}\Big(\frac{0.9}{1 + e^{-\theta_2}} - \frac{0.1}{1 + e^{-\theta_2}} + \frac{0.1}{1 + e^{-\theta_1}} - \frac{0.9}{1 + e^{-\theta_1}} + 1\Big) \\ &= \frac{0.4}{1 + e^{-\theta_2}} - \frac{0.4}{1 + e^{-\theta_1}} + 0.5. \end{aligned}$$

□

Thus, using the previous derivation, the gradients of the SRS with respect to the automaton internal states can be calculated as follows

$$\begin{aligned} \frac{\partial E[\beta \mid \theta]}{\partial\theta_1} &= -\frac{0.4e^{-\theta_1}}{(1 + e^{-\theta_1})^2}, \\ \frac{\partial E[\beta \mid \theta]}{\partial\theta_2} &= \frac{0.4e^{-\theta_2}}{(1 + e^{-\theta_2})^2}. \end{aligned} \tag{4.24}$$

Defining continuous time interpolations $\theta^a(\cdot)$ of $\theta(\cdot)$ for a specific learning rate $a > 0$ as

$$\theta^a(t) = \theta(n) \quad \text{for} \quad t \in [an, a(n+1)]. \tag{4.25}$$

Then using the weak convergence techniques, it can be shown that the sequence θ^a converges weakly, as $a \to 0$, to $z(\cdot)$, where $z(\cdot)$ satisfies the ODE

$$\dot{z} = \nabla_\theta E[\beta \mid \theta], \qquad z(0) = \theta(0). \tag{4.26}$$

Thus from Eq.4.24 and using Eq.4.26, the corresponding ODE can be expressed as

$$\begin{aligned}\frac{d\theta_1}{dt} &= \frac{\partial E[\beta \mid \theta]}{\partial \theta_1} = -\frac{0.4e^{-\theta_1}}{(1+e^{-\theta_1})^2},\\ \frac{d\theta_2}{dt} &= \frac{\partial E[\beta \mid \theta]}{\partial \theta_2} = \frac{0.4e^{-\theta_2}}{(1+e^{-\theta_2})^2}.\end{aligned} \tag{4.27}$$

The pair of decoupled ODEs in Eq.4.27 show that θ_1 and θ_2 decreases and increases respectively, without bound. θ_1 diverges to $-\infty$ and θ_2 to $+\infty$. Thus, from the mathematical viewpoint mentioned above, it can be concluded that the REINFORCE algorithm exhibits unbounded behavior, which is undesirable. In the next section an algorithm will be proposed that will overcome the limitations of the REINFORCE algorithm.

4.4 Modified REINFORCE algorithm

As mentioned earlier, the aim of the REINFORCE algorithm is to maximize the expected reward $E[\beta \mid \theta]$ over the entire space θ. However, the REINFORCE algorithm exhibits unbounded behavior and the explanation has been mentioned in the previous section. To mitigate this problem, one way is to modify the problem of unconstrained maximization to a constrained maximization problem. The modified version of the REINFORCE algorithm is commonly referred to by using the same name, but with the addition of “Modified”, before it.

The updating scheme of the Modified REINFORCE algorithm is as follows

$$\theta_i(n+1) = \theta_i(n) + a\beta(n)e_i + aK_i[h_i(\theta_i(n)) - \theta_i(n)], \qquad \forall i = (1,2,\ldots,l), \tag{4.28}$$

where $h(\theta) = [h_1(\theta_1), h_1(\theta_1)], \ldots, h_l(\theta_l)$ and the function h_i is defined as

$$h_i(\eta) = \begin{cases} L_i & \text{for} \quad \eta \geq L_i \\ \eta & \text{for} \quad |\eta| \leq L_i \\ -L_i & \text{for} \quad \eta \leq -L_i \end{cases}$$

and $L_i,\ K_i > 0$ are constants. The nature of the window function $h(\eta)$ is displayed in Figure4.2

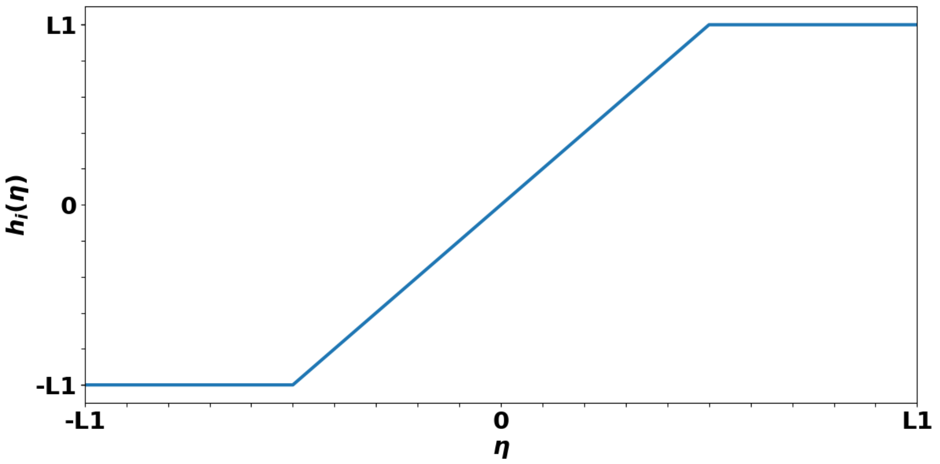

Figure 4.2: The window function $h_i(\eta)$

The addition of the constraint to the conventional REINFORCE algorithm is represented by the third term in Eq.4.28. Thus, the third term acts as a penalizing function in such a

manner that it pulls back the internal state values to remain in the window $-L_i \leq \eta \leq L_i$. For a detailed reading on the Modified-REINFORCE algorithm, article [31] can be referred.The algorithms discussed so far consider that an automaton has the option of selecting an action out of an available two actions at every instant n.
The contribution in this thesis is to extend the Modified REINFORCE algorithm for a multi-action automaton. In earlier sections it has been mentioned that the selection of an action $\alpha(n)$ at any instant n is determined by an action probability generating function and the same for the REINFORCE algorithm is represented by Eq.4.12. Thus, for this approach it can be borne in mind that extending the modified REINFORCE algorithm requires a change in the probability generating function and in the following section it will be discussed in detail.

4.5 Modified REINFORCE Algorithm for Multi-Action Automaton

As mentioned in the preceding sections, the algorithms highlighted have been investigated only for an automaton having only two actions. In this section an action probability will be proposed that can extend the algorithms to include multi-action automaton.
For an automaton with r actions, the set of actions is represented by $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ and therefore, the matrix representing the internal states of the automaton will have $r \times l$ elements, where l represents the number environment context. The internal state of the automaton at any instant for an action $\alpha(n) = \alpha_i$ is represented by $\theta_i = \{\theta_{i1}, \theta_{i2}, \ldots, \theta_{il}\}$, where $i \in \{1, 2, \ldots, r\}$. The probability of selecting an action $\alpha(n)$ at any instant is determined by the action probability generating function, represented by

$$g(\alpha(n) = \alpha_i, x, \theta) = \frac{e^{\theta_i^T x}}{\sum_{j=1}^{r} e^{\theta_j^T x}}. \tag{4.29}$$

Let the quantity in the numerator of Eq.4.29 be represented as $z = \sum_{j=1}^{r} e^{\theta_j^T x}$. The characteristic eligibility can be expressed as

$$\begin{aligned}
\frac{\partial}{\partial \theta_i} \ln g(\alpha_i, x, \theta) &= \frac{1}{g(\alpha_i, x, \theta)} \frac{\partial}{\partial \theta_i} g(\alpha_i, x, \theta) \\
&= \frac{1}{g(\alpha_i, x, \theta)} \frac{\partial}{\partial \theta_i} \frac{e^{\theta_i^T x}}{z} \\
&= \frac{1}{g(\alpha_i, x, \theta)} \frac{z x e^{\theta_i^T x} - e^{\theta_i^T x} x e^{\theta_i^T x}}{z^2} \\
&= \frac{x e^{\theta_i^T x}}{g(\alpha_i, x, \theta)} \left[\frac{1}{z} - \frac{e^{\theta_i^T x}}{z^2}\right] \\
&= z x \left[\frac{1}{z} - \frac{e^{\theta_i^T x}}{z^2}\right] \\
&= x \left[1 - \frac{e^{\theta_i^T x}}{z}\right] \\
&= x[1 - g(\alpha_i, x, \theta)].
\end{aligned} \tag{4.30}$$

In a similar fashion, it can be shown that, for all $j \neq i$

$$\frac{\partial}{\partial \theta_j} \ln g(\alpha_i, x, \theta) = -x \frac{e^{\theta_j^T x}}{z}, \qquad \forall j \neq i. \tag{4.31}$$

Thus, the update scheme can be expressed, by substituting the magnitudes of the characteristic eligibility from Eq.4.30 and Eq.4.31 in Eq.4.28, as shown in Eq.4.32.

Assuming $\alpha(n) = \alpha_i$

$$\begin{aligned}\theta_i(n+1) &= \theta_i(n) + a\beta(n)x[1 - g(\alpha_i, x, \theta)] + aK_i[h_i(\theta_i(n)) - \theta_i(n)],\\ \theta_j(n+1) &= \theta_j(n) - a\beta(n)x\left[\frac{e^{\theta_j^T x}}{z}\right] + aK_i[h_j(\theta_j(n)) - \theta_j(n)] \qquad i \neq j.\end{aligned} \tag{4.32}$$

The algorithm represented using Eq.4.32 will be called "M-RAMA(Local)", which stands for the acronym "*Modified REINFORCE Algorithm for Multi-Action Automaton (Local)*". This algorithm is suitable for local optimization and in many cases local results do not suffice. For such cases, an algorithm capable to find the global optimum is necessary. To find the global optimum of stochastic differential equations, either the simulated annealing algorithm [42] or the constant temperature heat bath algorithm [43] are generally used. For both algorithms, a random term is incorporated in the stochastic differential equation to circumvent it from being stuck at a local optimum. For the simulated annealing algorithm, the random term is slowly reduced to zero; whereas for the constant temperature heat bath model, it is maintained at a low constant value. Since the proposed methodology is an online algorithm, the simulated annealing technique is insufficient, as it is based on a decaying model. Hence the global equivalent of the scheme presented in Eq.4.32 has to be based on the heat bath technique. The algorithm which is capable of finding the global optimum is presented in Eq.4.33

$$\begin{aligned}\theta_i(n+1) &= \theta_i(n) + a\beta(n)x\left[1 - \frac{e^{\theta_i^T x}}{\sum_{j=1}^{r} e^{\theta_j^T x}}\right] + aK_i h_i^{'}(\theta_i(n)) + \sqrt{a}\zeta_i(n),\\ \theta_j(n+1) &= \theta_j(n) - a\beta(n)x\left[\frac{e^{\theta_j^T x}}{z}\right] + aK_i h_j^{'}(\theta_j(n)) + \sqrt{a}\zeta_j(n) \qquad i \neq j.\end{aligned} \tag{4.33}$$

where h_i is defined as

$$h_i(\eta) = \begin{cases} -(\eta - L_i)^{2J} & \text{for} \quad \eta \geq L_i \\ 0 & \text{for} \quad |\eta| \leq L_i \\ -(\eta + L_i)^{2J} & \text{for} \quad \eta \leq -L_i \end{cases}$$

and ζ is a sequence of i.i.d. random variables of zero mean and variance σ^2. J is a positive integer and $h^{'}$ is the first derivative of h.

The algorithm presented using Eq.4.33 will be called "M-RAMA(Global)". In the remaining part of this section, the performance of M-RAMA(Local) will be analyzed using synthetic data, which will give a preliminary insight of the working principle of the proposed technique.

The utility of the proposed algorithm can be explained by a simple example. Let an automaton with three actions $\alpha = \{\alpha_1, \alpha_2, \alpha_3\}$ operate in an environment with context vector $x = \{x_1, x_2\}$, where $\{x_1, x_2\} \in [-1, 1]$. The context space spanned by x_1 and x_2 is divided into three regions by the lines $x_2 - 2x_1 = 1$, $x_2 + 2x_1 = 1$ and in each region a particular automaton action is optimal and is shown in Figure4.3. The aim of the reinforcement scheme mentioned earlier is to align the internal states θ of the automaton in such a manner such that for every $x(n) \in \text{Region}_i$, $\max(g(\alpha, x, \theta)) = g(\alpha(n) = \alpha_i, x, \theta) > g(\alpha(n) = \alpha_j, x, \theta)$, where $i \neq j$. The given automaton consists of three actions, hence the internal states of the automaton have to be 3-dimensional. In order to satisfy the calculation of the probability generating function for an automaton having 3-dimensional internal states, the environment context at all instances has be be three dimensional, i.e. $x(n) = \{x_1, x_2, x_3\}$. In such cases, a conventional approach is to consider $x_3(n) = 1$, $\forall n$. Thus the internal state vector of the automaton, for an action α_i, can be represented as $\theta_i = \{\theta_{i1}, \theta_{i2}, \theta_{i3}\}$.

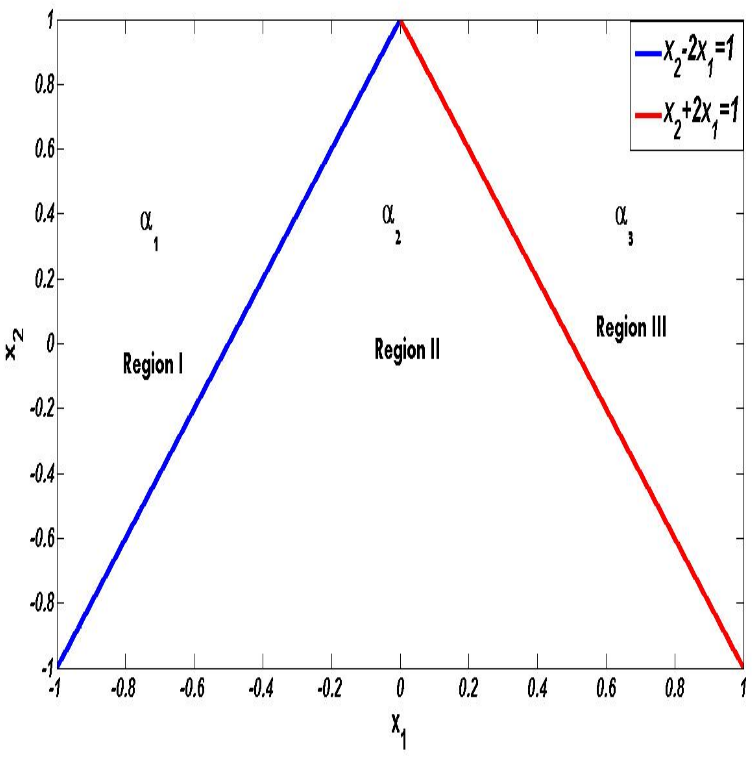

Figure 4.3: Segregation of context space into three distinct regions and the corresponding optimal actions in those regions

Every online algorithm requires random samples from the context space to be provided during the training phase. In this case, during the training of the proposed algorithm, $10,000$ randomly generated points in the context space, are supplied as input. The training phase utilizes the update equations, as mentioned in Eq.4.32, and, for this example, the simulation is carried out with the learning parameter $a = 0.1$. The scalar reinforcement signal β is assumed to be 0.9, for optimal action and 0.1 otherwise. The initial values of all of the internal states is assumed to be zero. The training outcome is shown in Figure4.4, where the algorithm learns well, as can be seen from the colors of the graph, in the mainly correct selection of the optimal action for a given a set of context inputs.

Similar to other online algorithms, during the testing phase, evaluation of accuracy in learning of optimal parameters is carried out. For evaluation, a testing set comprising of $10,000$ random points in the context space is generated. For this example, instances where the algorithm chose incorrect actions is shown in Figure4.5, which is small in number.
To verify the performance of the proposed algorithm, the average probability of selecting an optimal action is calculated, by executing the algorithm for 1000 instances. The accuracy of the optimal action selection is greater than 98% and is shown in Figure4.6.
The algorithm described in this section can also be considered as a linear classification or recognition of objects/patterns, where the feature attributes are numerical quantities. Hence, the example described above can be thought of as a 2-feature, 3-class linear pattern recognition problem.
The learning automata algorithms discussed so far explain the process of making an automaton expedient, both when the environment is stationary and non-stationary. As mentioned earlier, the aim of the learning process is to select an action $\alpha(n)$ at every instant n such that the expected reward/penalty is maximized/minimized. In all earlier discussions it has been assumed that the automaton has a finite number of options in selecting an action at an instant n, i.e. $\alpha(n) \in \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$, where $r \in \mathbb{N}$. However, as the number of automaton actions

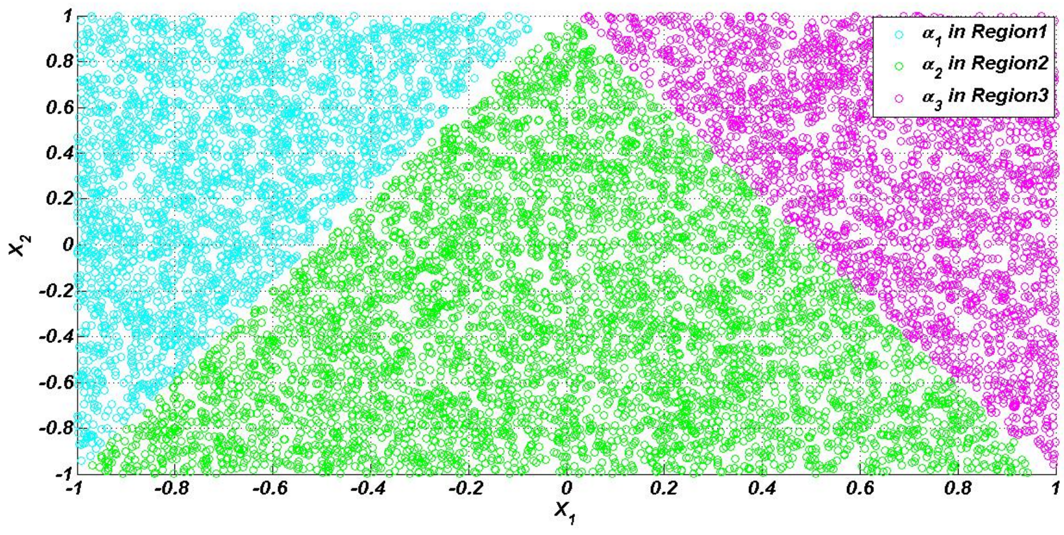

Figure 4.4: Selection of optimal action depending on the environment context

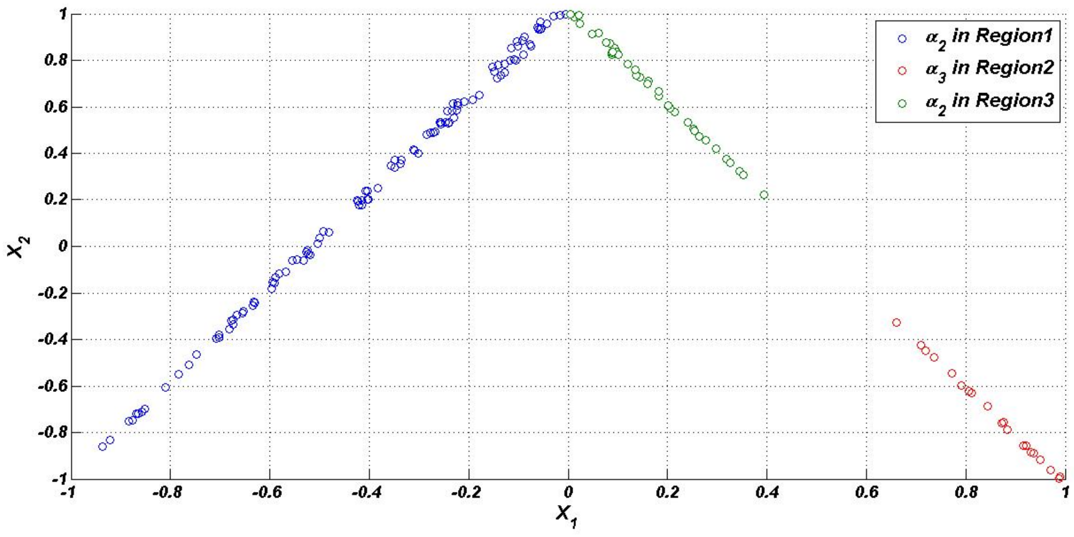

Figure 4.5: Instances of incorrect action selected for context points in different regions

increases, the number of iterations required to attain convergence to select an optimal action also increases significantly. In certain cases, it is required that the set of actions represents an interval on a real line. A possible solution to such cases is to discretize the entire interval into smaller segments and each element of the action set is made to represent the magnitude of the smaller segment. Subsequently, the reinforcement schemes, like the L_{R-P} scheme, are implemented to solve the problem. However, this approach is somewhat inaccurate in the sense that, either the discretization may be too coarse for the problem or finer segmentation increases the number of automaton actions, which in turn makes the system too slow.

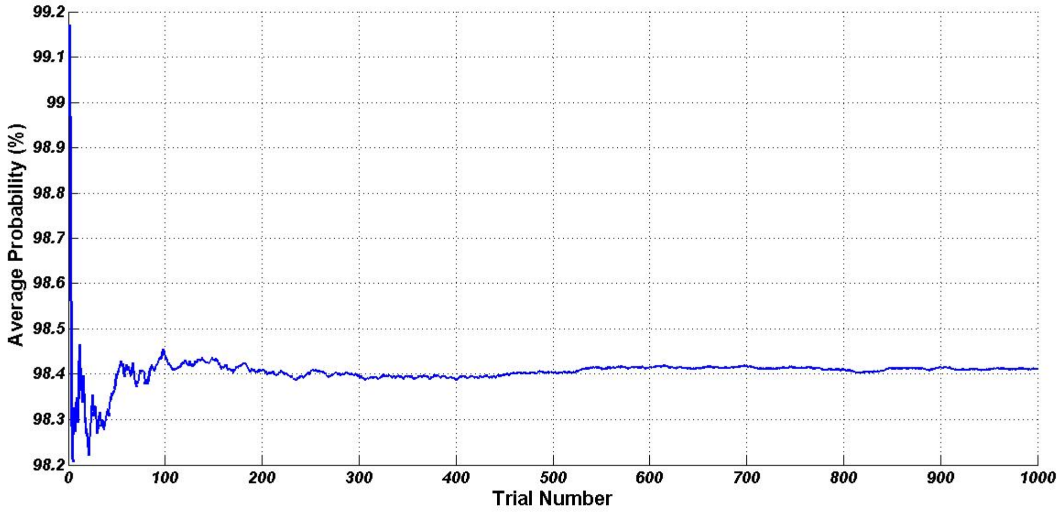

Figure 4.6: Average success probability in selecting the optimal action in all regions.

Chapter 5

Performance

5.1 Resource requirement for selected machine learning algorithms

The theoretical estimate for the computing resource requirement of an algorithm for any given problem can be analyzed using computational complexity theory [32]. In general, the resource requirement for an algorithm is either measured by its runtime or memory requirement or by both. A naive way of defining the runtime of an algorithm is the number of operations executed as a function of the input size, whereas the memory demand is the usage of memory by the algorithm, while it is executing.

Runtime analysis options:

The runtime of an algorithm can be analyzed using any of the following:

- Worst-case analysis: represents the upper bound of the number of operations of an algorithm required for any arbitrary input. For example, with an input of size n, the runtime of the merge sort algorithm is upper bounded by $6n \log n + 6n$. The designer's aim should be that the worst-case runtime grows slowly with input size. The most widely accepted convention is to represent the runtime of all algorithms, except the randomized algorithms, by worst-case analysis.

- Average-case analysis: is the average runtime of an algorithm over all possible input sizes. Typically, the runtime of randomized algorithms is represented using average-case analysis. Randomized algorithms have the ability to make random selections, which leads to the calculation of the expected running time.

- Benchmark: represents the runtime of an algorithm when tested under a certain number of benchmark inputs, which can be assumed to represent practical or typical inputs for the algorithm.

- Asymptotic analysis: represents the runtime of an algorithm for very large input sizes. A widely used convention for representing the asymptotic behavior of a deterministic algorithm is to suppress the constants and lower order terms of the worst-case analysis. For example, in the case of the merge sort algorithm, its asymptotic behavior is represented by $n \log n$, where n represents the input size.

Further details of the various approaches for algorithm analysis is available in [32].

The Asymptotic runtime can be expressed in generalized mathematical notation as follows:

Definition: $T(n) = O(f(n))$ if and only if there exists constants c, $n_0 > 0$, such that

$$T(n) \leq cf(n), \quad \forall n \geq n_0.$$

where, $T(n)$ represent the asymptotic notation of the worst-case runtime of an algorithm and n represents the input size.

The earlier mathematical statement can be expressed in words as follows: for a large input n, $T(n)$, i.e. the asymptotic worst-case runtime, which, for large input size is the Asymptotic runtime, is bounded above by a constant multiple of a function $f(n)$.
In this work, the resource requirements of algorithms are evaluated using Benchmark analysis. Although Benchmark analysis is considered a naive approach, but at the initial stage of algorithm design it can still be useful, if the domain of its use is known beforehand. Here, the benchmark analysis is implemented by selecting a small number of diverse data sets from scientific data repositories. For the resource requirement evaluation purpose, the IRIS flower and the Wine data set from UC Irvine (UCI) Machine Learning Repository has been selected as the benchmark data sets. All the data sets considered here are used by numerous scientific articles to benchmark the performance of algorithms. In the remaining part of this section, a comparison of the resource requirements of M-RAMA(Local) and M-RAMA(Global) with the widely used algorithms like Support Vector Machine with Linear, Quadratic, Gaussian and Polynomial kernel, Neural Network, Decision Tree, Adaboost is presented. The resource requirement profiles of the machine learning algorithms for the benchmark data sets are measured by the embedded profiling tool in MATLAB [33]. The resource requirement profile of the selected algorithms on the IRIS flower and Wine data sets is presented in Table5.1 and Table5.2 respectively. For ease of analysis of Table5.1 and Table5.2, the following definitions are of use

- Allocated memory: is the total amount of memory allocated within the function and its children.
- Freed memory: is the total amount of memory released within the function and its children.
- Peak memory: is the maximum amount of memory usage at any instant during the execution of the function and its children.
- Self memory: is the memory used by the function only. In other words, self memory does not consider the memory usage by the children of the parent function.

Here, the units for measurement of time and memory are second and kilobit respectively. For further details on the quantities described earlier, the reader is referred to [33].

5.2 Simulation configurations

In the previous section, the resource requirements of selected machine learning algorithms on the IRIS and Wine data sets has been presented. In the remaining part of the chapter a comparison of the performance of both M-RAMA(Local) and M-RAMA(Global) with the other machine learning algorithms will be presented. The performance of some of the existing machine learning algorithms will be evaluated on the benchmark data sets and then compared with the performance of M-RAMA(Local) and M-RAMA(Global) on the same collection of the data sets. For comparison of the performance of various algorithms, apart from the IRIS Flower and Wine data sets, the Breast Tissue and the Vehicle data set from the UCI Machine Learning Repository, has also been considered. Also, for the purpose of performance comparison, the Bagging and the Naive Bayes' classifiers have been incorporated in the selected set of machine learning algorithms chosen earlier for the resource requirement benchmarking purpose.

Algorithm	Total Time (sec)	Total Allocated Memory (kb)	Total Freed Memory (kb)	Total Peak Memory (kb)
M-RAMA(Global)	0.130	28	90	12
M-RAMA(Local)	0.115	24	88	12
SVM(Linear Kernel)	0.592	40	360	40
SVM(RBF Kernel)	0.52	72	328	60
SVM(Polynomial Kernel)	0.29	40	360	40
SVM(Quadratic Kernel)	0.287	40	360	40
Neural Network	0.64	24	132	24
Decision Tree	0.623	212	132	164
Adaboost	5.5	456	400	144

Table 5.1: Resource requirement for the algorithms when using the Wine data set

Algorithm	Total time (sec)	Total allocated memory (kb)	Total freed memory (kb)	Total peak memory (kb)
M-RAMA (Global)	0.153	16	80	12
M-RAMA (Local)	0.138	16	76	12
SVM (Linear kernel)	0.41	16	216	16
SVM (RBF kernel)	0.41	16	216	16
SVM (Polynomial kernel)	0.304	48	168	36
SVM (Quadratic kernel)	0.284	24	216	24
Neural network	0.49	24	276	12
Decision Tree	0.65	336	348	280
Adaboost	2.57	588	144	328

Table 5.2: Resource requirement for the algorithms when using the IRIS data set

Algorithm	**Average accuracy**(%)			
	Wine data set	IRIS data set	Vehicle data set	Breast tissue data set
M-RAMA (Global)	98.0	96.0	66.0	64.9
M-RAMA (Local)	98.0	95.9	66.0	65.0
SVM (Linear kernel)	96.7	96.5	66.2	58.5
SVM (RBF kernel)	80.5	94.5	46.1	58.8
SVM (Polynomial kernel 3^{rd} order)	88.9	96.0	58.1	56.3
SVM (Polynomial kernel 4^{th} order)	88.7	93.8	54.5	53.4
SVM (Polynomial kernel 5^{th} order)	84.7	90.5	50.1	48.9
SVM (Quadratic kernel)	89.8	**97.5**	64.8	58.9
Neural network	**98.1**	95.7	**69.5**	57.3
CART	90.3	94.6	59.5	64.0
AdaBoost	96.2	95.1	49.6	52.9
Bagging	97.6	94.8	68.2	**67.1**
Naive Bayes	97.2	95.3	46.5	63.8

Table 5.3: Average test set accuracy of selected Machine Learning algorithms on the Wine, the IRIS, the Vehicle and Breast Tissue data sets

In this section, the configurations of the algorithms under which the simulation experiment has been conducted, is mentioned. For the training of the Support Vector Machine algorithms, the Sequential Minimal Optimization (SMO) [34] is used as the optimization method for the Linear Kernel and Least Squares [35] for the rest. The rationale for this approach comes from [34], where the author presented the advantage of using SMO over Quadratic Programming (QP) as the optimization technique for Support Vector Machines with a Linear Kernel. The benefit arises as, the SMO technique breaks down a large QP problem into a series of the smallest possible QP problems. This approach of dividing a large problem into smaller sub-problems reduces the computational complexity manifold.
Apart from this benefit, the simulation study also reveals that the performance efficiency of SVM with a Linear Kernel and using SMO as optimization method, is far better than using the SVM with Linear Kernel and using Least Squares as the optimization technique. Also, for all the SVM algorithms a one-vs-all classification approach has been used, in terms of performance, the one-vs-all approach is as accurate as any other scheme for multiclass classification [36].
The multiclass AdaBoost [37] algorithm is one of the most commonly used boosting algorithms, which uses multiple *probably approximately correct* or PAC learning units. An individual PAC unit performs slightly better than a random guessing algorithm and thus is a "weak" learner. However, the performance of the combined unit has arbitrarily high accuracy. The performance evaluation of the Naive Bayes' classifier on the Vehicle data set has been presented in [38] and is the motivation for the inclusion of Naive Bayes' classifier in the list of selected algorithms, to benchmark against the originally published results for the same data set. For the multiclass Adaboost and Bagging algorithms, 100 tree classifiers are employed. For all data sets, the performance of three-layered Neural Networks has also been presented. The hidden layer of the network comprises three, four, nine and ten units for the Wine, the IRIS, the Breast Tissue and the Vehicle data sets respectively. The number of deployed units in the hidden layer are exactly similar to the dimension of the feature space.

It has been mentioned previously that the learning automata based models (M-RAMA) are incremental algorithms. In other words, given the feature vector and the corresponding class, during the process of training, the learning automata based classification algorithm updates the internal state in a manner such that the misclassification error is reduced. The update equation for M-RAMA (Local) is given by Eq.5.1

$$\begin{aligned}\theta_i(n+1) &= \theta_i(n) + a\beta(n)x\left[1 - \frac{e^{\theta_i^T x}}{\sum_{j=1}^{r} e^{\theta_j^T x}}\right] + aK_i[h_i(\theta_i(n)) - \theta_i(n)],\\ \theta_j(n+1) &= \theta_j(n) - a\beta(n)x\left[\frac{e^{\theta_j^T x}}{z}\right] + aK_i[h_j(\theta_j(n)) - \theta_j(n)] \qquad i \neq j.\end{aligned} \tag{5.1}$$

where, θ represents the internal state, $h(\theta) = [h_1(\theta_1), h_2(\theta_1), \ldots, h_l(\theta_l)]$ and the function h_i is defined as

$$h_i(\eta) = \begin{cases} L_i & \text{for} \quad \eta \geq L_i \\ \eta & \text{for} \quad |\eta| \leq L_i \\ -L_i & \text{for} \quad \eta \leq -L_i \end{cases}$$

and L_i, $K_i > 0$ are constants. x represents the feature or context vector and n signifies the iteration.

The update equation of M-RAMA (Global) is depicted in Eq.5.2,

$$\begin{aligned}\theta_i(n+1) &= \theta_i(n) + a\beta(n)x[1 - \frac{e^{\theta_i^T x}}{\sum_{j=1}^{r} e^{\theta_j^T x}}] + aK_i h_i^{'}(\theta_i(n)) + \sqrt{a}\zeta_i(n),\\ \theta_j(n+1) &= \theta_j(n) - a\beta(n)x[\frac{e^{\theta_j^T x}}{z}] + aK_i h_j^{'}(\theta_j(n)) + \sqrt{a}\zeta_j(n) \qquad i \neq j.\end{aligned} \tag{5.2}$$

Parameters	Data sets			
	Wine	IRIS	Vehicle	Breast tissue
a	0.2	0.2	0.2	0.2
σ	0.002	0.001	0.001	10^{-6}

Table 5.4: Parameter values used for different data sets

where h_i is defined as

$$h_i(\eta) = \begin{cases} -(\eta - L_i)^{2J} & \text{for} \quad \eta \geq L_i \\ 0 & \text{for} \quad |\eta| \leq L_i \\ -(\eta + L_i)^{2J} & \text{for} \quad \eta \leq -L_i \end{cases}$$

and ζ is a sequence of i.i.d. random variables of zero mean and variance σ^2. J, a positive integer and $h^{'}$ is the first derivative of h. For both Eq.5.1-5.2, L is a window function.

For the simulation experiment, L is assumed to be 100, J is 1 and K is 15. The rest of the parameters used in either Eq.5.1 or in Eq.5.2 are summarized in Table5.4. With the parameter values mentioned earlier, Figure5.1-5.4 shows the training process of the proposed learning algorithms on the selected data sets. For the completion of the training process, each training instance for every data set are duplicated for a certain number of times. The training instances are duplicated for 20, 40, 100 and 60 times for the Wine, IRIS, Breast Tissue and Vehicle data sets respectively. For scaling of the data features, a mean normalization of the features has been implemented. The scaled version $\underline{x_i}$ of a feature x_i is represented as,

$$\underline{x_i} = \frac{x_i - \bar{x_i}}{\sigma_i},$$

where $\bar{x_i}$ and σ_i signifies the mean and standard distribution of the i^{th} feature.

In [39], the authors have conducted performance evaluation of 179 classification algorithms belonging to 17 families over 121 data sets. However, the authors have mostly used default configurations of classification functions available in different libraries, spread over many language platforms. The article again concludes that the best performing classifier is the Random Forest. However, as mentioned in the earlier part of this section, the default technique for SVM in MATLAB is SMO, but most of the kernels perform much better when Least Squares is used as an optimization method. Hence, a default configuration might not be the best. In this work, the selection of the algorithms is semi-random and out of the 17 available families of algorithms, algorithms from 7 families, mostly with their best performing configuration, has been included.

5.3 Performance evaluation methodology

A widely used approach for evaluating the performance of an algorithm is to calculate the accuracy over as few as ten instances [40]. The final accuracy is presented in terms of the mean and the standard deviation of the outcomes. The representation of the performance distribution, using only the first two moments perhaps assumes the non-existence of higher order moments, which is true only for normal distributions. To the best of our knowledge, most of the articles depict the performance of algorithms using only the first moments, assuming the outcomes to have a Gaussian distribution and do so without performing the normality test [41]. As mentioned in [42], 50% of the scientific articles published have at least one statistical error and here it will be verified whether it is sufficient to portray the behavior following the course chosen by the existing articles. Also, typically representing a performance

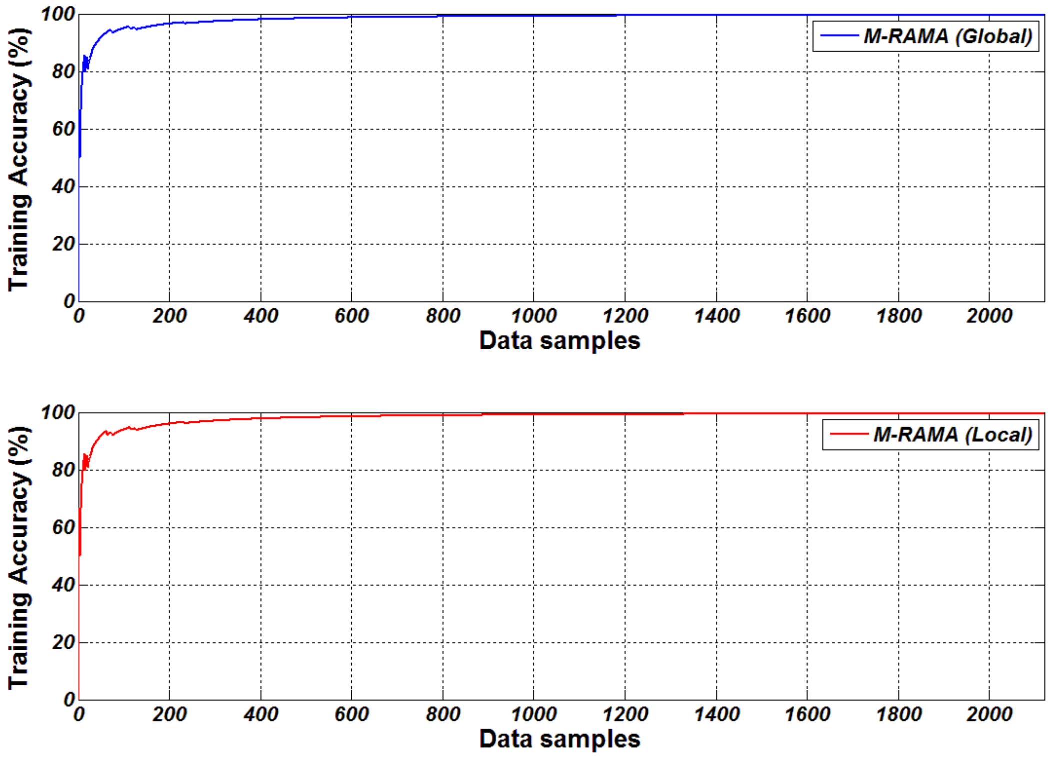

Figure 5.1: Training process of the proposed algorithms on the Wine data set

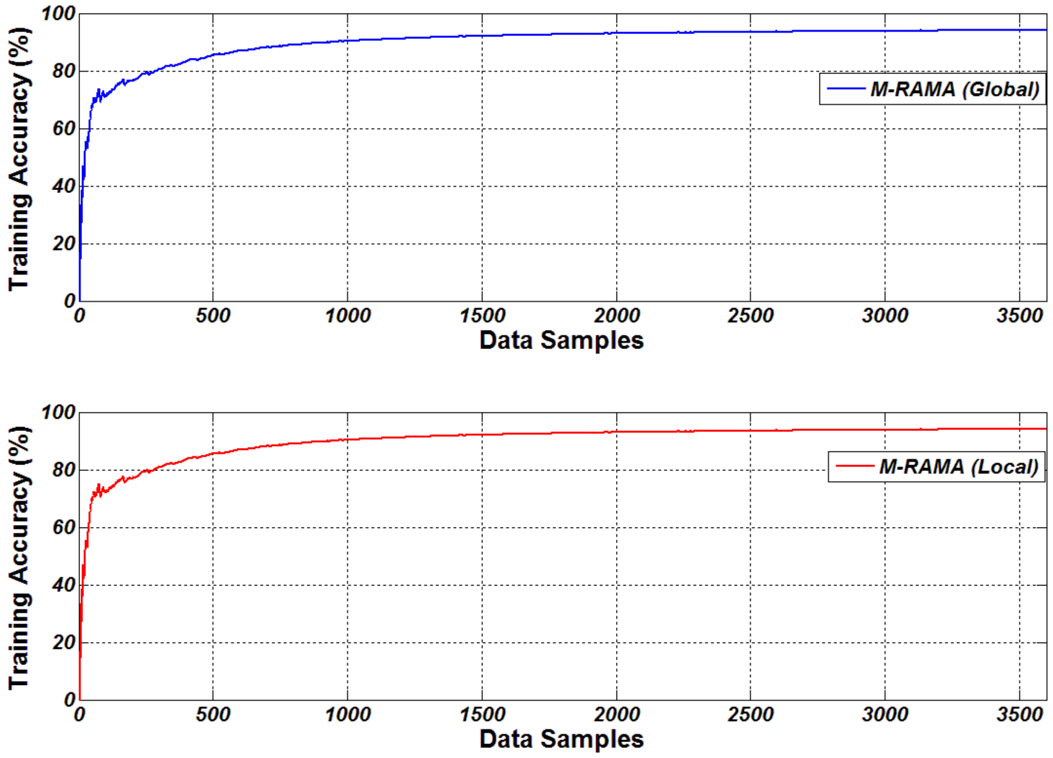

Figure 5.2: Training process of the proposed algorithms on the IRIS data set

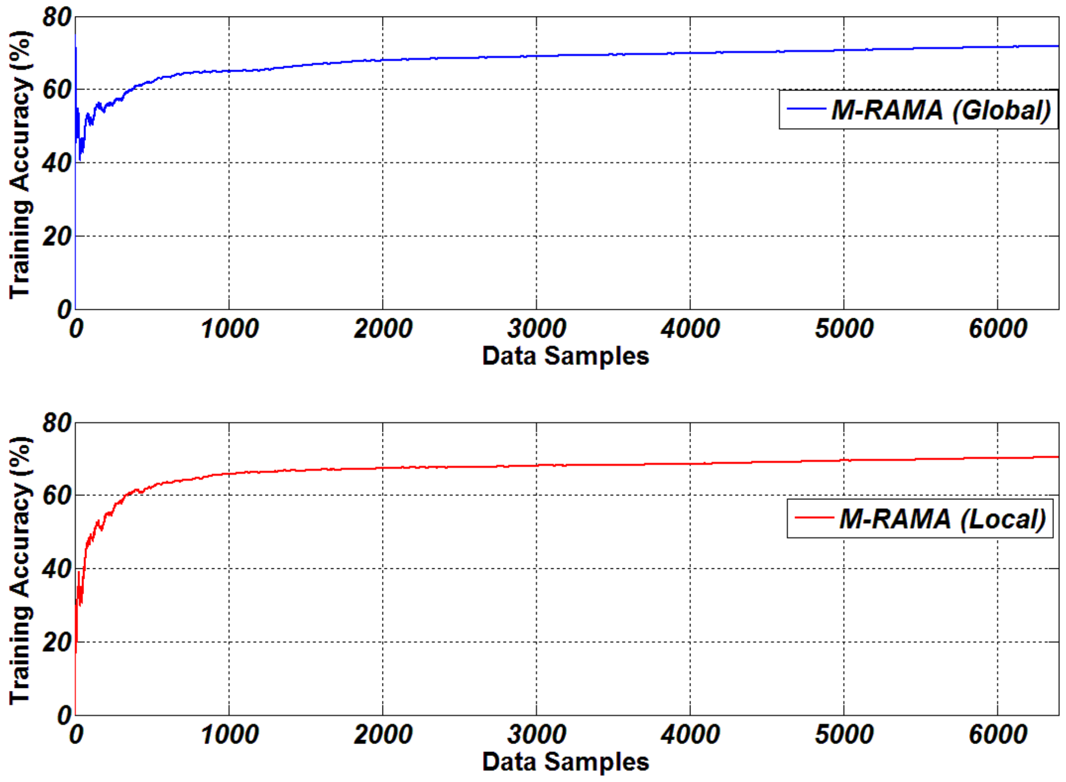

Figure 5.3: Training process of the proposed algorithms on the Breast Tissue data set

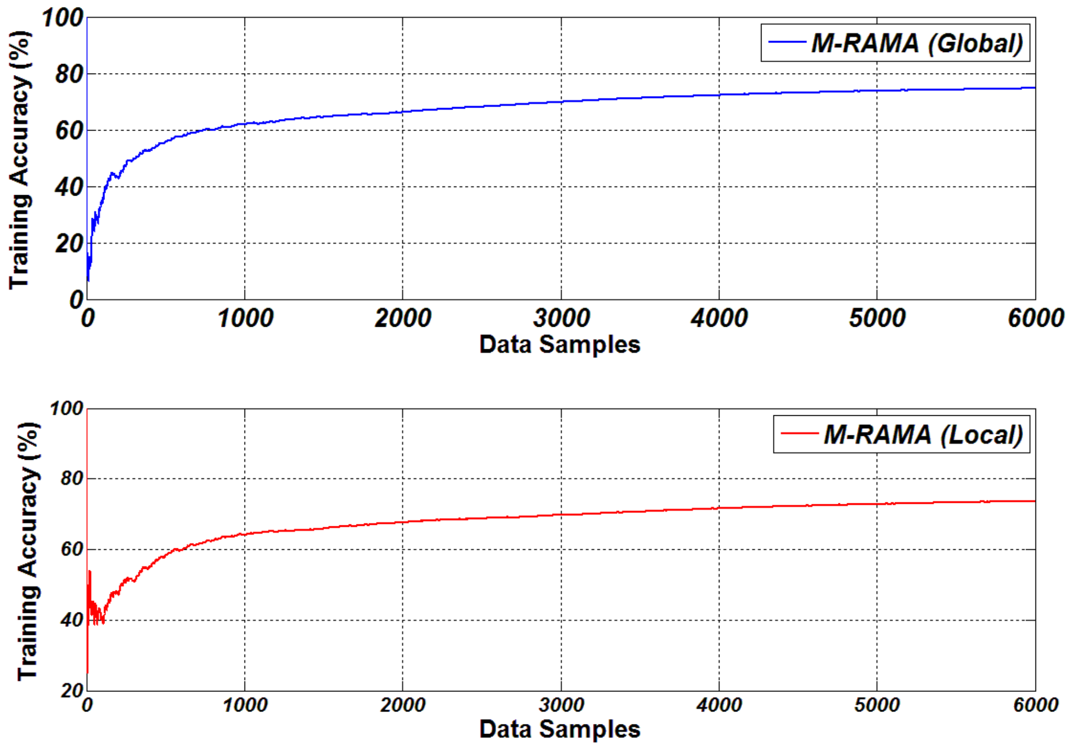

Figure 5.4: Training process of the proposed algorithms on the Vehicle data set

distribution of an algorithm using its mean and standard deviation is valid only when the distribution is symmetric. For a unimodal, symmetric distribution, the mean, the median and the mode are all similar; whereas, for a multimodal distribution, the mean and the median are not identical. Also for a skewed distribution [43], the mean and median are dissimilar. The performance distribution of all the algorithms considered over the data sets will be presented pictorially in Figure5.10-5.12 . Apart from this, the skewness and kurtosis of the performance distributions will be investigated. For a normal distribution, the skewness and kurtosis are 0 and 3 respectively. For a negatively skewed distribution the mean of the distribution is situated to the left of the mode of the distribution and for a positively skewed distribution, the mean is towards the right of the mode. Also, normality tests like the Jarque-Bera test [44], the Lilliefors test [45], the Anderson-Darlington test [46] and the Chi-Squared test [47] are included, which might assist in coosing the correct way for representation of the performance of the algorithms. For all the hypotheses, a 0 test result indicates the null hypothesis, i.e. the distribution is Gaussian; whereas, a 1 indicates that the test result rejects the null hypothesis at the 5% significance level.

5.4 Performance of selected machine learning algorithms

For the simulation study, the training and test set constitutes 60% and 40% of the total data points, respectively. For example, while working with the IRIS data set, the training and test sets constitute 90 and 60 samples respectively out of the total 150 data points. A prevalent practice is to select the training and test data set randomly. Hence, for the IRIS flower data set, the random selection technique generates $\binom{150}{90} = \binom{150}{60} = 4.6215 \times 10^{42}$ number of training and test sets. The large number training/test set indicates that it is difficult to evaluate the performance of an algorithm on every training/test set due to limited computing resources. On the other hand, selecting some specific training/test set from the available combinations introduces bias and might not represent the actual performance of the model algorithm.

Even if a small number of training/test sets are randomly selected, to the best of our knowledge, there is no empirical rule regarding the number of such training/test sets to be selected for optimal evaluation of the algorithm performance. In such scenarios, an efficient approach can be the representation of the performance of the designed algorithm by the moving average accuracy. This gives the stopping criterion for choosing the number of the instances required for evaluating an algorithm over the data set under consideration. The process is described as follows:

- Step I: Select training and test set based on random sampling with replacement.
- Step II: Evaluate the performance on the selected training and test set. Let $T_{st}(i)$ represents the test set accuracy at instant $i = 1, \ldots, M$.
- Step III: The moving average accuracy at every instant i by

$$\mu_i = \frac{T_{st}(i) + (i-1)\mu_{i-1}}{i}, \quad \forall i > 1,$$

  where μ_{i-1} represents the average accuracy at any instant $(i-1)$.
- Step IV: Repeat Step I to Step III until, $|\mu_i - \mu_{i-1}| < \epsilon, \forall i \in [k, k+s]$, is attained, where ϵ is a very small positive quantity, $0 < s < k$ and $0 < k < M$.

The technique, though simple, is still extremely useful in the sense that, in spite of evaluating the algorithms over a relatively small number of combinations, it assists in representing the near actual performance of the algorithms to what would have been, if evaluated over all the available combinations. It will be proved in the following that the series of the moving mean test accuracy, of the sequence of test accuracies, does converge.

Proposition 5.1. *If $\{X_n\}$ is a sequence of i.i.d random variables representing the percentage test set accuracies, then $X_n \in [0, 100]$ for all $n \in \mathbb{N}$.*

Theorem 5.1. *(Popoviciu's inequality on variance) If $X_n \in [a, b]$ for all $n \in \mathbb{N}$, where $a, b \in \mathbb{R}$, then $Var(X_n) \leq \frac{(b-a)^2}{4}$.*

Definition 5.1. *If $\{F_n\}$ be a sequence of events in the probability space $(\Omega, \mathcal{F}, P)$, where Ω is the sample space and $\mathcal{F}$ is the σ-algebra. Then the event,*

$$\bigcap_{n=1}^{\infty} \bigcup_{m=n}^{\infty} F_m = \{\omega \in \Omega | \omega \in F_m \textit{infinitely often}\}.$$

is called "F_n infinitely often" and is abbreviated as "F_n i.o.".

Lemma 5.1. *(Borel-Cantelli Lemma) If $\sum_{n=1}^{\infty} P(F_n) < \infty$, then $P(F_n i.o.) = 0$.*

Proof. By Definition 5.1, F_ni.o. $= \bigcap_{n=1}^{\infty} \bigcup_{m=n}^{\infty} F_m$ and hence for each n,

$$P(F_n\text{i.o.}) \leq P(\bigcup_{m=n}^{\infty} F_m) \leq \sum_{m=n}^{\infty} P(F_m).$$

Now, given $\sum P(F_m) < \infty$, then

$$\lim_{n \to \infty} P(F_n\text{i.o.}) = 0.$$

□

Definition 5.2. *If $\{X_n\}$ is a sequence of random variables and assuming X to be some other variable, then,*
(i) X_n converges in probability to X if for all $\epsilon > 0$, $\lim_{n\to\infty} P(|X_n - X| > \epsilon) = 0$ and is represented by $X_n \xrightarrow{p} X$.
(ii) X_n converges almost surely to X if $P(\lim_{n\to\infty} X_n = X) = 1$ and is represented by $X_n \xrightarrow{a.s.} X$.

Theorem 5.2. *Let $\{X_n\}$ be a sequence of random variable defined over a probability space. If $X_k \xrightarrow{p} X$ then there exists a subsequence $\{X_{n_j}\} \subset \{X_k\}$ such that*

$$X_{n_k} \xrightarrow{a.s.} X.$$

Proof. For each $j \in \mathbb{N}$, let n_j is so large such that,

$$P(|X_{n_j} - X| > \frac{1}{j}) \leq \frac{1}{j^2}.$$

Let $A_j = \{|X_{n_j} - X| > \frac{1}{j}\}$.
Since,

$$\sum \frac{1}{j^2} < \infty,$$

then
$$P(A_j) < \infty.$$

Hence from the Borel-Cantelli lemma 5.1, $P(A_j\text{i.o.}) = 0$. □

Theorem 5.3. *(Markov Inequality) If $X \geq 0$ is a random variable, then for any $\alpha > 0$,*

$$P(X > \alpha) \leq \frac{E[X]}{\alpha}.$$

Proof. The expectation of the non-negative random variable X,

$$\begin{aligned} E[X] &= E[X1_{\{X\leq\alpha\}}] + E[X1_{\{X>\alpha\}}], \\ &\geq E[X1_{\{X>\alpha\}}], \\ &\geq \alpha E[1_{\{X>\alpha\}}] = \alpha P(X>\alpha), \end{aligned}$$

Hence, $P(X > \alpha) \leq \frac{E[X]}{\alpha}$. □

Theorem 5.4. *(Chebyshev Inequality) If X is a random variable with mean μ and variance $\sigma^2 < \infty$, then ,*

$$P(|X-\mu| > k\sigma) \leq \frac{1}{k^2}, \forall k > 0.$$

Proof. As $|X-\mu|^2 \geq 0$ and hence from Markov Inequality 5.3,

$$P(|X-\mu|^2 > (k\sigma)^2) \leq \frac{E[|X-\mu|^2]}{(K\sigma)^2} = \frac{\sigma^2}{k^2\sigma^2} = \frac{1}{k^2}.$$

Therefore $$P(|X-\mu| > k\sigma) \leq \frac{1}{k^2}.$$

□

Theorem 5.5. *(Sandwich Theorem) Let $\{X_n\}$, $\{Y_n\}$ and $\{Z_n\}$ be sequences such that $X_n \leq Y_n \leq Z_n$ and let $X_n \to l$ and $Z_n \to l$, then Y_n also converges to l.*

Proof. Please refer [48]. □

Theorem 5.6. *If $\{X_n\}$ is a sequence of independent and identically distributed random variable with $E[X_i] < \infty$ and if $S_n = \sum_{i=1}^n X_i$, then $\frac{S_n}{n} \xrightarrow{a.s.} E[X]$, i.e. $P(|\frac{S_n}{n} \to E[X]|) = 1$.*

Proof. If $\{X_n\}$ is a sequence of i.i.d. random variables representing the test set accuracies in percentage, then using Definition 5.1, every element in the sequence is bounded. Then using Theorem 5.1, the variance σ_X of the distribution is bounded from above. Hence, $\sigma_X < \infty$. Since X_i are independent random variables, using the linearity property of expectation

$$E[S_n] = nE[X],$$

Therefore $$E[\frac{S_n}{n}] = E[X].$$

Also,

$$\begin{aligned} Var(S_n) &= nVar(X), \\ Var(\frac{S_n}{n}) &= \frac{\sigma_X^2}{n}. \end{aligned}$$

Using Chebyshev Inequality 5.4,

$$\begin{aligned} P(|\frac{S_n}{n} - E[X]|^2 > \epsilon^2) &\leq \frac{E[|\frac{S_n}{n} - E[X]|^2]}{\epsilon^2}, \\ &\leq \frac{Var(\frac{S_n}{n})}{\epsilon^2}, \\ &\leq \frac{\sigma_X^2}{n\epsilon^2}. \end{aligned}$$

Therefore $$P(|\frac{S_n}{n} - E[X]| > \epsilon) \leq \frac{\sigma_X^2}{n\epsilon^2}.$$

Taking limits of n,

$$\begin{aligned} \lim_{n\to\infty} P(|\frac{S_n}{n} - E[X]| > \epsilon) &= \lim_{n\to\infty} \frac{\sigma_X^2}{n\epsilon^2} \\ &= 0 \end{aligned}$$

Therefore $$\frac{S_n}{n} \xrightarrow{p} E[X],$$

Hence $$\frac{S_n}{n} \xrightarrow{p} \mu,$$

Therefore, under the given conditions $\frac{S_n}{n}$ converges in probability to μ. Using this result and coupled with the Theorem 5.2 it can be concluded that there exists a subsequence n_j such that $\frac{S_n}{n} \xrightarrow{a.s.} \mu$.
Let, $n_j = j^2$, $\forall j \geq 1$ be the subsequence.
Then,

$$P(|\frac{S_{j^2}}{j^2} - \mu| > \epsilon) \leq \frac{\sigma_X^2}{j^2\epsilon^2},$$

Therefore $$\sum_{j=1}^{\infty} P(|\frac{S_{j^2}}{j^2} - \mu| > \epsilon) \leq \infty, \quad \forall \epsilon > 0.$$

Using the Borel-Cantelli Lemma 5.1,

$$\frac{S_{j^2}}{j^2} \xrightarrow{a.s.} \mu, \quad \text{as} \quad j \to \infty.$$

However, this result is true for only the square terms in the sequence and is yet to be the proved for the non-square instances in the sequence. Let i be s.t, $j^2 \leq i \leq (j+1)^2$.
Since $X_j \geq 0$,

$$S_{j^2} \leq S_n \leq S_{(j+1)^2},$$
$$\frac{S_{j^2}}{(j+1)^2} \leq \frac{S_n}{n} \leq \frac{S_{(j+1)^2}}{j^2},$$
$$\frac{S_{j^2}}{(j+1)^2}\frac{j^2}{j^2} \leq \frac{S_n}{n} \leq \frac{S_{(j+1)^2}}{j^2}\frac{(j+1)^2}{(j+1)^2},$$

Therefore $$\frac{S_{j^2}}{j^2}\frac{j^2}{(j+1)^2} \leq \frac{S_n}{n} \leq \frac{S_{(j+1)^2}}{(j+1)^2}\frac{(j+1)^2}{j^2}.$$

As $\lim j \to \infty$,

$$\mu \leq \frac{S_n}{n} \leq \mu.$$

Applying the Sandwich Theorem 5.5 to the last expression ,

$$\frac{S_n}{n} \xrightarrow{a.s.} \mu.$$

□

Hence, for a given $\epsilon > 0$, there exists some $n > N$, for which $\frac{S_n}{n} \to \mu$, where $n, N \in \mathbb{N}$. This forms the backbone of the performance evaluation methodology, where ϵ is fixed to a small quantity and then efforts were made to find out the n for which the population mean actually converge to the distribution mean of the test set accuracy. In [38] the author intuitively felt that 500 subsets of the total number of combinations are required for estimating the algorithm performance. It is felt that the author actually tried to highlight that within 500 combinations, the parameter(error/accuracy) representing the algorithm will converge. However, without explicitly mentioning an ϵ, the notion of such declaration is mathematically vague. It is also felt that the actual choice of the number of combinations required for convergence is not independent of the data set under consideration, apart from algorithm under test.

As mentioned earlier, a commonly applied approach for selecting the number of training and test samples is to split the sample set in a ratio of 60% and 40%, where the larger set is used for training of the algorithm and the remaining for the testing purpose. However, in [38], the author has evaluated the performance of a handful of algorithms on the vehicle data set, using 100 samples of the available 846 data samples as the training set, and the remaining as the test set. In order to benchmark our proposed algorithms, an identical training/test set size is followed as mentioned in [38], though it is an unconventional strategy. Other than the Vehicle data set, the training and the test sets are split in a ratio of 6 : 4 for all the remaining data sets.

Table5.3 represents the Mean Accuracy of the selected algorithms, estimated over 5000 instances, on all of the four data sets considered here. It has been mentioned earlier that ϵ is assigned an arbitrary small quantity and then efforts are made to find the number of instances for which the parameter representing the performance of an algorithm converge. For the simulation experiment, the magnitude of $\epsilon = 0.007$ is assumed. Table5.9 depicts the instances required for the convergence of chosen machine learning algorithms on the data sets. The Moving Mean Accuracy (MMA) of the algorithms as a function of instances on the selected data sets is represented via Figure5.5-5.8. The MMA in Figure5.5-5.8 is displayed over the window of the starting instance to the instance of convergence of M-RAMA(Global).

Some of the statistical parameters of the accuracy distribution of the selected algorithms on all the four data sets are presented in Table5.5-5.8. All of these statistical estimates are calculated up to the instance of convergence of M-RAMA(Global). The accuracy distributions of the chosen algorithms on all the four data sets are shown in Figures5.9-5.12. These figures represent the distribution up to the instances of convergence of the individual algorithms. For example, from Table5.9, the SVM (Linear Kernel) on the IRIS data set converges at 1791 and Figure5.10, depicts the accuracy distribution up to instance 1791.

5.5 Analysis of simulation results

For the problems pertaining to classification, the objective is either to minimize the misclassification error or to maximize the classification accuracy [49]. For a multivariate objective function $f(x_1, \cdots, x_m)$, $x_i \in \mathbb{R}$, the aim of an algorithm under test, is to select the best set of $\{x_1^*, \cdots, x_m^*\}$, which gives the least number of misclassified data samples or the maximum number of accurately classified data points. In other words, the aim of an algorithm is to search for the global optimum. But often learning algorithms get stuck at a local optimum, which prevents attaining the goal of least misclassification error or maximum classification accuracy.

The error/accuracy distribution portrays the behavior of an algorithm. In other words, it shows how often an algorithm selects the optimum parameters $\{x_1^*, \cdots, x_m^*\}$. An algorithm that selects the global parameters frequently will have the same statistical estimates, i.e. similar mean, median and mode. For such algorithms the error/accuracy distribution is a normal distribution. In many cases algorithms fail to select the best set of parameters and get stuck in a local optimum. For such algorithms the performance distribution is skewed.

Tables5.10-5.13 present the normality test results of the performance distribution, for every algorithm, on all of the data sets considered here. For most algorithms, the performance distribution is negatively skewed, except for the Naive Bayes' on the Vehicle data set.

For the performance distributions of every algorithm on the Wine and IRIS data set, there is a rejection of the null hypothesis as the outcome of the normality test result, as shown in Tables5.10-5.11.

For the Vehicle data set, the Jarque-Bera test, the Anderson-Darlington test and the Chi-Squared test returns null hypothesis of the performance distribution for the SVM with a fourth order polynomial kernel. The Jarque-Bera test procedure is based on the skewness and kurtosis of the sample distribution. In other words, the Jarque-Bera test on a data sample returns a null hypothesis when the skewness and the kurtosis of the distribution is similar to that of a normal distribution and this is validated in the following. From Table5.12, both the skewness and kurtosis of the performance distribution of SVM, with a fourth order polynomial kernel, has an approximately similar value to a normal distribution and also, the Jarque-Bera test return a null hypothesis.

Similar observations can be made from Table5.13, where, for the SVM with a Linear kernel, a fifth order kernel and a quadratic kernel on the Breast Tissue data set, the Jarque-Bera test returns a null hypothesis along with the magnitude of skewness and kurtosis identical to a normal distribution.

The normality test results of the accuracy distributions of the selected machine learning algorithms reveal that denoting their performances with the first two moments are insufficient. It is suggested that the parameters such as the mode of the distribution, the maximum and minimum accuracy should be included, as they give deeper insights for performance analysis of the algorithm under test.

On the whole, M-RAMA has performed well in the tests presented here. From Table5.3 it can be seen that M-RAMA has one of highest mean accuracies of all the considered data sets. Also, Tables5.5-5.8 shows that M-RAMA has one of the lowest standard deviations on every selected data set. Moreover, Tables5.5-5.8 shows that M-RAMA has one of the highest minimum and maximum accuracies, which highlights the stability of M-RAMA. Furthermore, Table5.9 shows that M-RAMA requires one of lowest number of instances for convergence, as per the definition given in Section 5.4.

According to [38], the accuracy of the Naive Bayes' algorithm on the Vehicle data set after 500 instances is $46.80\% \pm 0.16$. The present study, with the same simulation configuration, finds that the mean accuracy of the Naive Bayes' algorithm after 500 instances is 46.7%, which is similar to [38] and that after 5000 iterations, the mean accuracy dips slightly to 45.5%. In the same scenario, the accuracy of M-RAMA is 66%, which indicates better accuracy of M-RAMA, under those circumstances.

In Section 5.4, it was mentioned that the number of instances required for the simulation study is dependent on both the algorithm under test and also on the data set. The MMA for the SVM algorithm with Gaussian Kernel or Radial Basis function (RBF) shows undulating characteristics even after 500 instances, which can be seen in Figure5.5. However, the MMA of the same algorithm shows smooth behavior for the Vehicle data set, even for a small number of instances. Also, from Table5.9, the convergence of SVM with the RBF kernel requires 3621 and 2197 instances on the Wine and Vehicle data sets respectively. Thus the number of instances for convergence for the algorithm under discussion is lower in the case of the Vechicle data set than for the Wine data set. This again validates the assertion that the required number of instances for the performance evaluation is not independent of the data set under consideration.

The resource requirements of the selected Machine Learning algorithms on chosen data sets are mentioned in Section 5.1. Tables5.1-5.2 reveal that the M-RAMA algorithms require less computational resources than the other chosen algorithms for these data sets. M-RAMA (Local) requires even less resources than its global counterpart, owing to the fact that the M-RAMA (Global) requires an additional random term.

Further work will examine the algorithmic complexity, as a function of the number of data set features, where performance for much larger features should be examined through a generalized version of complexity analysis.

Algorithm	Wine data set					
	Minimum Accuracy (%)	Maximum Accuracy (%)	Standard Deviation	Median Accuracy (%)	Modal Accuracy	Mean Accuracy (%)
M-RAMA(Global)	90.0	100.0	1.8	98.3	98.3	98.0
M-RAMA(Local)	90.0	100.0	1.8	98.3	98.3	98.0
SVM(Linear Kernel)	86.7	100.0	2.2	96.7	98.3	96.7
SVM(RBF Kernel)	58.3	96.7	6.6	81.7	81.7	80.5
SVM(Polynomial Kernel 3^{rd} order)	70.0	100.0	4.3	90.0	88.3	88.9
SVM(Polynomial Kernel 4^{th} order)	70.0	100.0	4.7	88.3	90.0	88.7
SVM(Polynomial Kernel 5^{th} order)	56.7	98.3	5.7	85.0	85.0	84.7
SVM(Quadratic Kernel)	76.7	100.0	3.8	90.0	90.0	89.8
Neural Network	90.0	100.0	1.6	98.3	98.3	98.1
CART	70.0	100.0	4.5	90.0	90.0	90.3
AdaBoost	86.7	100.0	2.2	96.7	96.7	96.2
Bagging	88.3	100.0	1.9	98.3	98.3	97.6
Naive Bayes	83.3	100.0	2.0	96.7	98.3	97.2

Table 5.5: Some parameters of the accuracy distribution of selected Machine Learning algorithms on the Wine data set

Algorithm	IRIS data set					
	Minimum Accuracy (%)	Maximum Accuracy (%)	Standard Deviation	Median Accuracy (%)	Modal Accuracy	Mean Accuracy (%)
M-RAMA(Global)	88.3	100.0	2.2	96.7	96.7	96.0
M-RAMA(Local)	88.3	100.0	2.2	96.7	96.7	95.9
SVM(Linear Kernel)	88.3	100.0	1.9	96.7	96.7	96.5
SVM(RBF Kernel)	83.3	100.0	2.4	95.0	95.0	94.5
SVM(Polynomial Kernel 3^{rd} order)	85.0	100.0	2.4	96.7	96.7	96.0
SVM(Polynomial Kernel 4^{th} order)	81.7	100.0	2.9	93.3	95.0	93.8
SVM(Polynomial Kernel 5^{th} order)	75.0	100.0	3.9	91.7	91.7	90.5
SVM(Quadratic Kernel)	90.0	100.0	1.8	98.3	98.3	97.5
Neural Network	85.0	100.0	2.2	95.0	96.7	95.7
CART	81.7	100.0	2.7	95.0	95.0	94.6
AdaBoost	83.3	100.0	2.2	95.0	95.0	95.1
Bagging	86.7	100.0	2.2	95.0	95.0	94.8
Naive Bayes	85.0	100.0	2.3	95.0	95.0	95.3

Table 5.6: Some parameters of the accuracy distribution of selected Machine Learning algorithms on the IRIS data set

Algorithm	Breast Tissue data set					
	Minimum Accuracy (%)	Maximum Accuracy (%)	Standard Deviation	Median Accuracy (%)	Modal Accuracy	Mean Accuracy (%)
M-RAMA(Global)	38.1	85.7	6.4	64.3	66.7	65.0
M-RAMA(Local)	38.1	83.3	6.4	64.3	66.7	65.0
SVM(Linear Kernel)	33.3	78.6	6.4	59.5	59.5	58.5
SVM(RBF Kernel)	31.0	81.0	6.8	59.5	59.5	58.8
SVM(Polynomial Kernel 3^{rd} order)	31.0	81.0	7.6	57.1	54.8	56.3
SVM(Polynomial Kernel 4^{th} order)	23.8	81.0	7.8	52.4	54.8	53.4
SVM(Polynomial Kernel 5^{th} order)	19.0	73.8	8.1	50.0	50.0	48.9
SVM(Quadratic Kernel)	33.3	81.0	6.7	59.5	59.5	58.9
Neural Network	28.6	81.0	6.9	57.1	57.1	57.3
CART	38.1	85.7	6.7	64.3	64.3	64.0
AdaBoost	16.7	76.2	8.2	52.4	52.4	52.9
Bagging	40.5	85.7	6.3	66.7	66.7	67.1
Naive Bayes	38.1	83.3	6.8	64.3	64.3	63.8

Table 5.7: Some parameters of the accuracy distribution of selected Machine Learning algorithms on the Breast Tissue data set

Algorithm	Vehicle data set					
	Minimum Accuracy (%)	Maximum Accuracy (%)	Standard Deviation	Median Accuracy (%)	Modal Accuracy	Mean Accuracy (%)
M-RAMA(Global)	54.3	75.1	3.0	66.1	67.0	66.0
M-RAMA(Local)	52.3	75.1	2.9	66.1	65.8	66.0
SVM(Linear Kernel)	57.8	73.3	2.4	66.4	67.6	66.2
SVM(RBF Kernel)	35.5	58.6	3.5	46.2	46.8	46.1
SVM(Polynomial Kernel 3^{rd} order)	46.9	67.4	3.1	58.2	58.6	58.1
SVM(Polynomial Kernel 4^{th} order)	43.8	65.0	3.2	54.6	55.9	54.5
SVM(Polynomial Kernel 5^{th} order)	36.6	61.7	3.6	50.1	49.7	50.1
SVM(Quadratic Kernel)	54.6	75.1	2.9	64.9	66.2	64.8
Neural Network	60.1	77.2	2.5	69.7	70.1	69.5
CART	46.9	69.7	3.7	59.7	59.7	59.5
AdaBoost	26.9	63.9	5.6	49.7	45.8	49.6
Bagging	51.9	74.4	2.4	68.5	69.8	68.2
Naive Bayes	33.2	60.5	3.6	46.4	46.4	46.5

Table 5.8: Some parameters of the accuracy distribution of selected Machine Learning algorithms on the Vehicle data set

Algorithm	**Number of Instances required to converge**			
	Wine data set	IRIS data set	Vehicle data set	Breast Tissue data set
M-RAMA(Global)	1874	1620	2456	4725
M-RAMA(Local)	1874	1620	2456	4725
SVM(Linear Kernel)	2161	1791	2008	3493
SVM(RBF Kernel)	3621	2049	2197	3574
SVM(Polynomial Kernel 3rd order)	3196	2338	2401	4784
SVM(Polynomial Kernel 4th order)	3363	2322	2161	4591
SVM(Polynomial Kernel 5th order)	4363	3879	4347	4174
SVM(Quadratic Kernel)	2689	1716	2340	3897
Neural Network	1821	1919	1995	4057
CART	2525	2115	2400	4990
AdaBoost	2251	2183	3357	4700
Bagging	2010	1925	2657	4053
Naive Bayes	2514	2149	2448	3750

Table 5.9: The number of instances required for convergence of selected Machine Learning Algorithms on chosen data sets.

Algorithm	**Wine Data Set**					
	Skewness	Kurtosis	Jarque-Bera test	Lilliefors test	Anderson-Darling test	Chi-square test
M-RAMA(Global)	−0.87	3.58	1	1	1	1
M-RAMA(Local)	−0.84	3.46	1	1	1	1
SVM(Linear Kernel)	−0.92	4.20	1	1	1	1
SVM(RBF Kernel)	−0.21	2.74	1	1	1	1
SVM(Polynomial Kernel 3^{rd} order)	−0.40	3.23	1	1	1	1
SVM(Polynomial Kernel 4^{th} order)	−0.49	3.15	1	1	1	1
SVM(Polynomial Kernel 5^{th} order)	−0.38	3.10	1	1	1	1
SVM(Quadratic Kernel)	−0.33	2.99	1	1	1	1
Neural Network	−1.00	4.66	1	1	1	1
CART	−0.60	3.35	1	1	1	1
AdaBoost	−0.83	4.06	1	1	1	1
Bagging	−0.84	4.01	1	1	1	1
Naive Bayes	−0.73	4.14	1	1	1	1

Table 5.10: Normality Test results of the performance distribution of selected machine learning algorithms on the Wine data sets.

Algorithm	**IRIS Data Set**					
	Skewness	Kurtosis	Jarque-Bera test	Lilliefors test	Anderson-Darling test	Chi-square test
M-RAMA(Global)	-0.17	2.48	1	1	1	1
M-RAMA(Local)	-0.16	2.48	1	1	1	1
SVM(Linear Kernel)	-0.41	3.21	1	1	1	1
SVM(RBF Kernel)	-0.34	3.23	1	1	1	1
SVM(Polynomial Kernel 3^{rd} order)	-0.79	3.97	1	1	1	1
SVM(Polynomial Kernel 4^{th} order)	-0.52	3.24	1	1	1	1
SVM(Polynomial Kernel 5^{th} order)	-0.53	3.29	1	1	1	1
SVM(Quadratic Kernel)	-0.67	3.34	1	1	1	1
Neural Network	-0.35	3.17	1	1	1	1
CART	-0.47	3.19	1	1	1	1
AdaBoost	-0.66	3.90	1	1	1	1
Bagging	-0.22	2.92	1	1	1	1
Naive Bayes	-0.39	3.39	1	1	1	1

Table 5.11: Normality Test results of the performance distribution of selected machine learning algorithms on the IRIS data sets.

Algorithm	**Vehicle Data Set**					
	Skewness	Kurtosis	Jarque-Bera test	Lilliefors test	Anderson-Darling test	Chi-square test
M-RAMA(Global)	−0.40	3.42	1	1	1	1
M-RAMA(Local)	−0.37	3.47	1	1	1	1
SVM(Linear Kernel)	−0.30	3.09	1	1	1	1
SVM(RBF Kernel)	−0.03	2.70	1	1	1	0
SVM(Polynomial Kernel 3rd order)	−0.22	3.00	1	1	1	1
SVM(Polynomial Kernel 4th order)	−0.07	2.96	0	1	0	0
SVM(Polynomial Kernel 5th order)	−0.10	2.97	1	1	1	1
SVM(Quadratic Kernel)	−0.23	3.11	1	1	1	1
Neural Network	−0.28	3.03	1	1	1	1
CART	−0.29	2.83	1	1	1	1
AdaBoost	−0.14	2.76	1	1	1	1
Bagging	−0.71	4.74	1	1	1	1
Naive Bayes	0.21	2.92	1	1	1	1

Table 5.12: Normality Test results of the performance distribution of selected machine learning algorithms on the Vehicle data sets.

Algorithm	**Breast Tissue Data Set**					
	Skewness	Kurtosis	Jarque-Bera test	Lilliefors test	Anderson-Darling test	Chi-square test
M-RAMA(Global)	−0.20	3.12	1	1	1	1
M-RAMA(Local)	−0.22	3.19	1	1	1	1
SVM(Linear Kernel)	0.00	2.98	0	1	1	1
SVM(RBF Kernel)	−0.17	3.11	1	1	1	1
SVM(Polynomial Kernel 3^{rd} order)	−0.14	2.97	1	1	1	1
SVM(Polynomial Kernel 4^{th} order)	−0.08	3.06	1	1	1	1
SVM(Polynomial Kernel 5^{th} order)	−0.07	2.98	0	1	1	1
SVM(Quadratic Kernel)	−0.02	2.99	0	1	1	1
Neural Network	−0.17	3.19	1	1	1	1
CART	−0.20	3.04	1	1	1	1
AdaBoost	−0.34	3.39	1	1	1	1
Bagging	−0.11	3.10	1	1	1	1
Naive Bayes	−0.13	2.89	1	1	1	1

Table 5.13: Normality Test results of the performance distribution of selected machine learning algorithms on the Vehicle data sets.

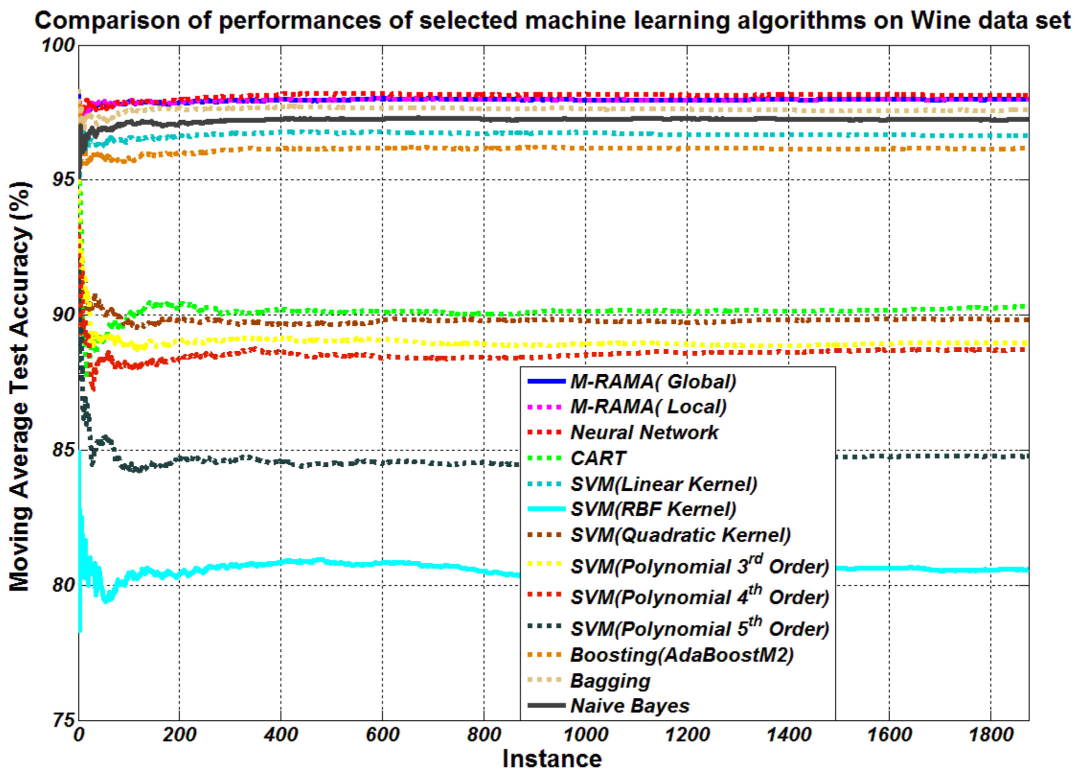

Figure 5.5: Performance of selected algorithms on the Wine data set

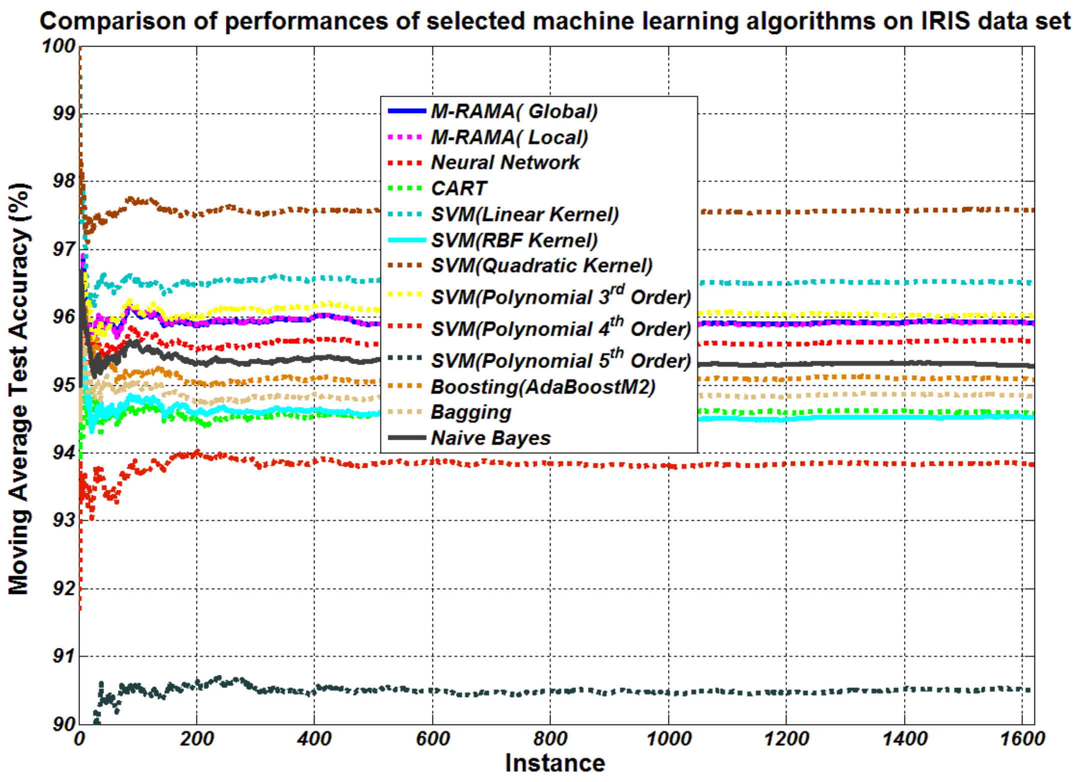

Figure 5.6: Performance of selected algorithms on the IRIS data set

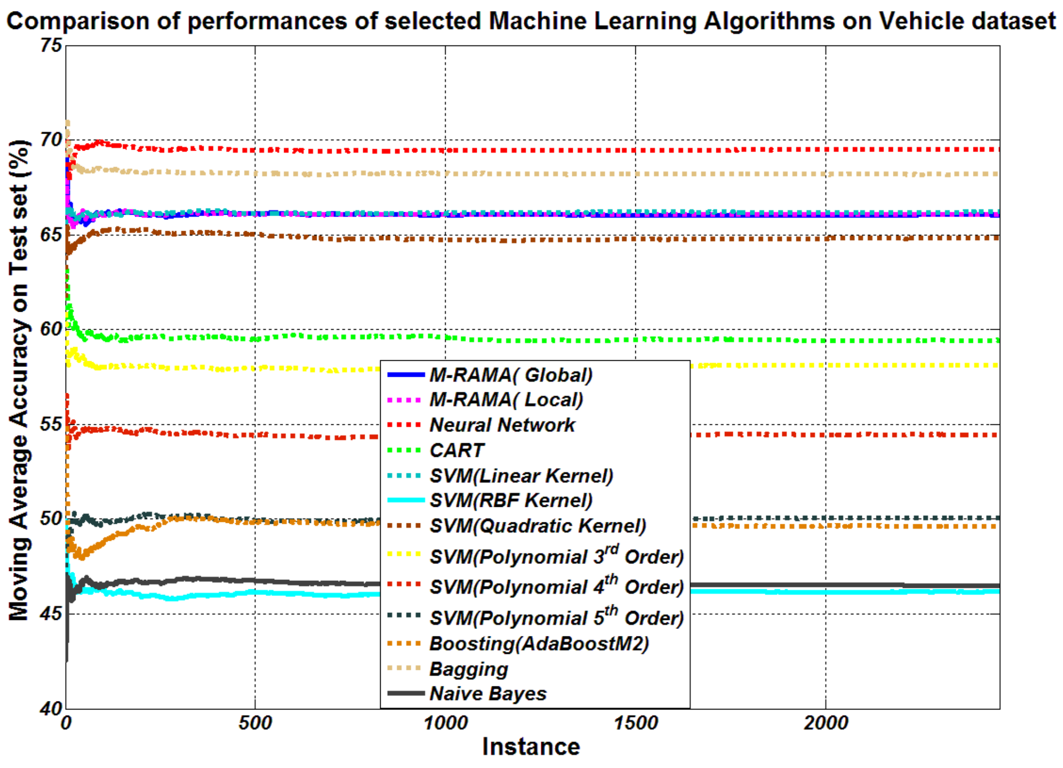

Figure 5.7: Performance of selected algorithms on the Vehicle data set

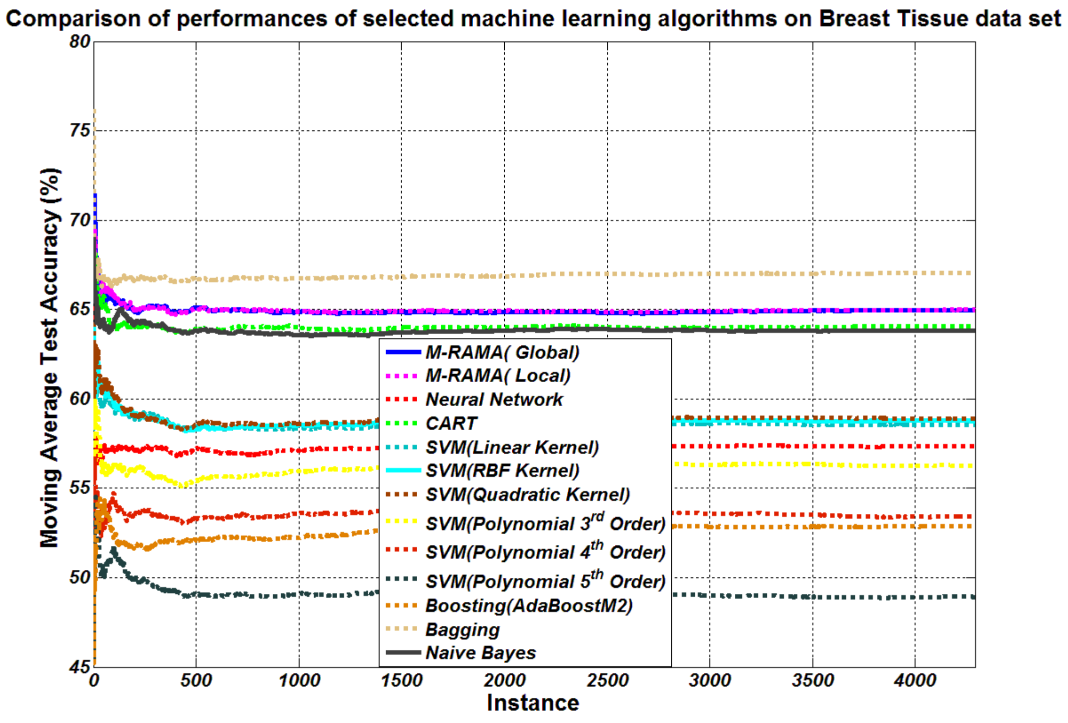

Figure 5.8: Performance of selected algorithms on the Breast Tissue data set

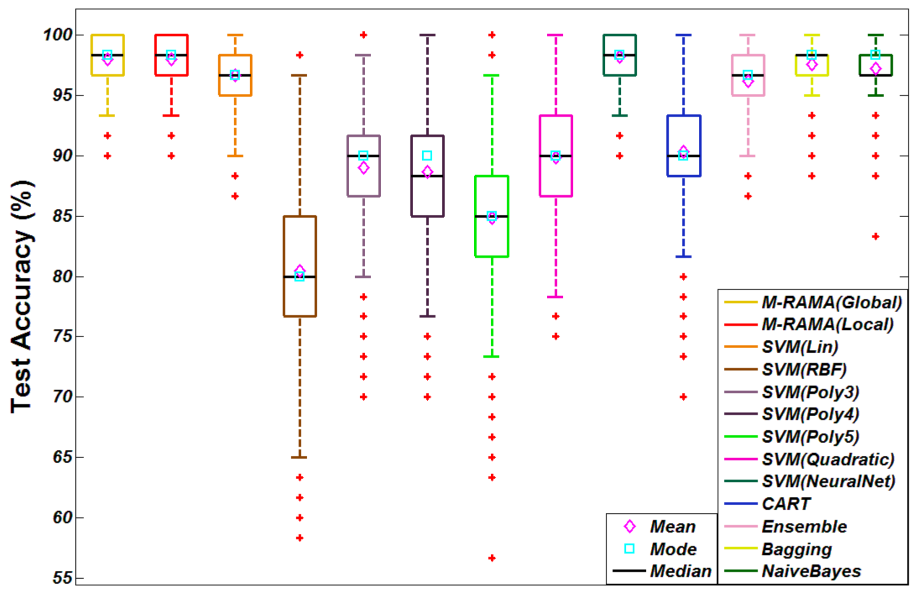

Figure 5.9: Performance distribution of selected algorithms on the Wine data set

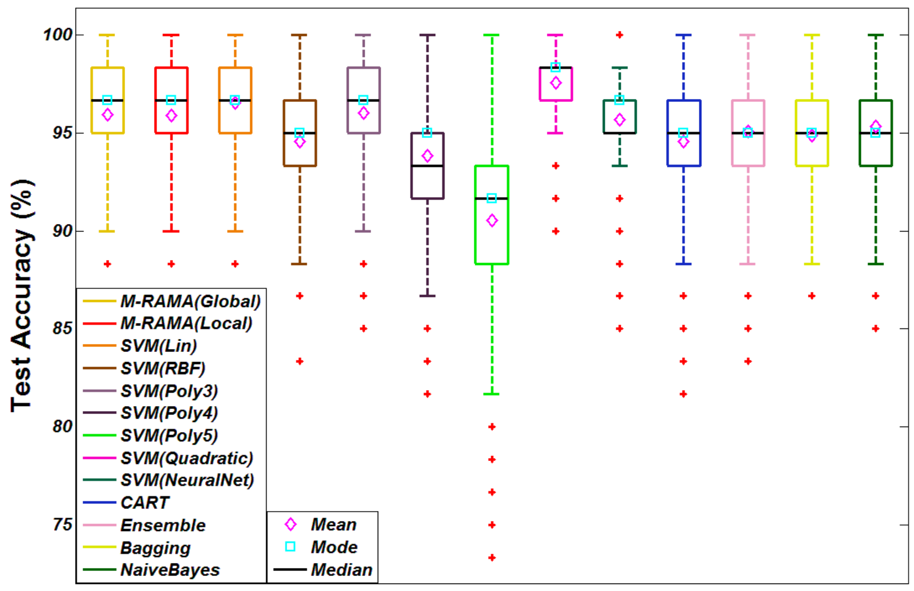

Figure 5.10: Performance distribution of selected algorithms on the Iris data set

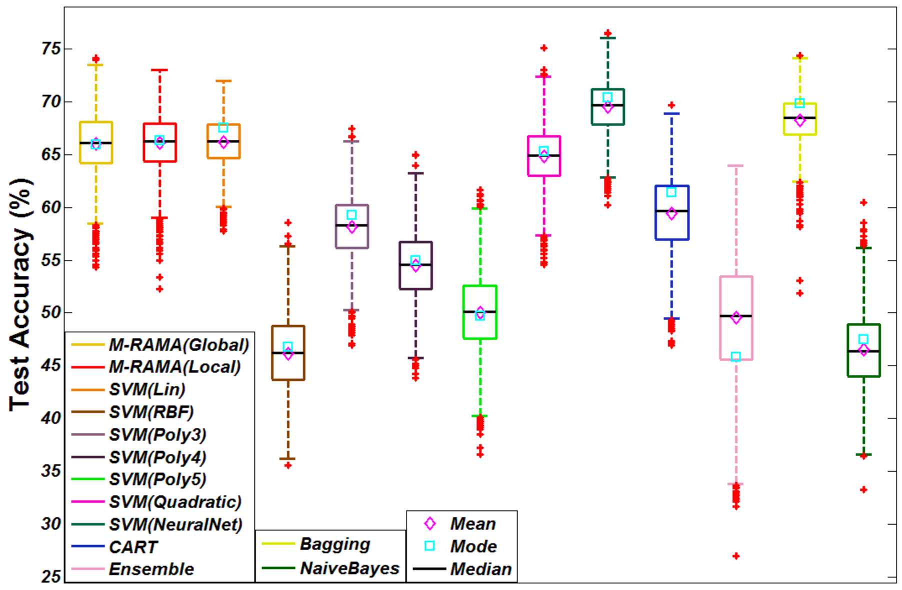

Figure 5.11: Performance distribution of selected algorithms on the Vehicle data set

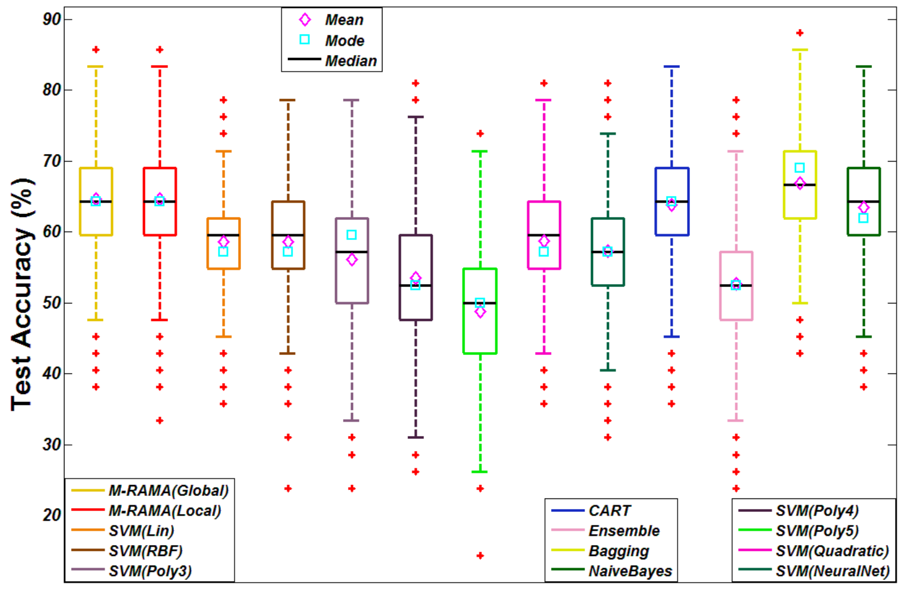

Figure 5.12: Performance distribution of selected algorithms on the Breast Tissue data set

Chapter 6

Summary and Conclusion

The focus of the thesis was to model a learning automata based classifier, capable of multi-class classification. In Chapter 4, two learning automata based algorithms have been proposed *viz.* M-RAMA(Local) and M-RAMA(Global). The former algorithm is capable of finding the local optimum and the latter has the ability to find the global optimum. The performance of the algorithms has been evaluated using data sets available from the UCI Machine Learning Repository. The test set accuracy of M-RAMA(Local) and M-RAMA(Global) are almost on par with the best performing algorithm over every data set considered here. The proposed algorithms are incremental algorithms and hence are suitable for operating in an online deployment. The proposed classification algorithms have been implemented in the three different programming platforms platforms namely Python, MATLAB and C++.

The research also highlighted a new methodology for representing the performance of the machine learning algorithms. The technique, though simple, still provides a method that circumvents the process of evaluating the performance of an algorithm under test, over all combinations of training and test sets of the available data set.

6.1 Future Work

Both M-RAMA(Local) and M-RAMA(Global) are yet to be tested with skewed data sets. For skewed data sets, representing the performance of machine learning algorithms using only the test set accuracy is insufficient. In such cases, generally, the performance is represented by the confusion matrix, precision, recall and F-Score. Further research can be carried out to observe the performance of M-RAMA(Local) and M-RAMA(Global) with skewed data sets and evaluate their performances using the parameters mentioned earlier.

Since both M-RAMA(Local) and M-RAMA(Global) are incremental algorithms and hence, can be used as classification tool for data sets that are noisy in nature. To evaluate the extent of the algorithms' ability to handle noise, further research is necessary.

For the present work, the approximation function is assumed to be linear, but this can be extended to any higher order. The evaluation of the resource requirements for the higher order function approximation can be carried out in future work. As the order of the approximation function increases, the computational complexity also increases. In that scenario, implementing the algorithms in the programming platforms capable of computing in a parallel/concurrent fashion is extremely useful. However, for parallel/concurrent computing, pure functional style of programming is more efficient and robust than both procedural and object oriented style of programming. Though Python, MATLAB and C++ supports functional style of programming, however, all of them have some limitations. For example, Python has limitations in evaluating functions via recursion. Hence, programming platforms such as

Haskell, which supports purely functional approach of implementation, is extremely helpful for parallel/concurrent computations. In future, the algorithms can be implemented in programming platforms like Haskell and others, that supports pure functional style of programming and investigate the performance of the proposed algorithms.

6.2 Probable applications of this research

The proposed multi-class classification techniques are based on stochastic automata. The algorithms are also incremental in nature. Being based on stochastic automata, both the techniques are capable of operating in environments comprising of noisy data. Also, being iterative algorithms, both the techniques are suitable for handling large data. Given the inherent qualities of Learning Automata and the nature of the modeled algorithms, the proposed techniques can be used as decision making agents in e-commerce sector, for classification in online data streaming and in several other areas. The resource requirement study of classifiers mentioned in Chapter 5 indicates that the learning automata bases algorithms require one of the least computational resources which make them suitable for deployment in electronic devices even with low processing strength.

Bibliography

[1] Keith A Quesenberry. *Social Media Strategy: Marketing and Advertising in the Consumer Revolution.* Rowman & Littlefield, 2015.

[2] Jeffrey Camm, James Cochran, Michael Fry, Jeffrey Ohlmann, and David Anderson. *Essentials of Business Analytics (Book Only).* Nelson Education, 2014.

[3] John A Parnell. *Strategic management: Theory and practice.* Sage Publications, 2013.

[4] Richard O Duda, Peter E Hart, David G Stork, et al. *Pattern classification*, volume 2. Wiley New York, 1973.

[5] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. Unsupervised learning. In *The elements of statistical learning*, pages 485–585. Springer, 2009.

[6] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[7] Charu C Aggarwal, Jiawei Han, Jianyong Wang, and Philip S Yu. On demand classification of data streams. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 503–508. ACM, 2004.

[8] Ludmila I Kuncheva and Indrė Žliobaitė. On the window size for classification in changing environments. *Intelligent Data Analysis*, 13(6):861–872, 2009.

[9] Mikhail TSetlin et al. Automaton theory and modeling of biological systems. 1973.

[10] Kumpati S Narendra and Mandayam AL Thathachar. *Learning automata: an introduction.* Courier Corporation, 2012.

[11] Andrew G Barto and P Anandan. Pattern-recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, (3):360–375, 1985.

[12] Ivan Petrovitch Pavlov and W Gantt. Lectures on conditioned reflexes: Twenty-five years of objective study of the higher nervous activity (behaviour) of animals. 1928.

[13] Clark Hull. Principles of behavior. 1943.

[14] W.K Estes. The statistical approach to learning theory. *S. Koch(Ed.), Psychology: A Study of a Science*, II, 1959.

[15] Cletus J Burke, William K Estes, and Sidney Hellyer. Rate of verbal conditioning in relation to stimulus variability. *Journal of Experimental Psychology*, 48(3):153, 1954.

[16] Robert R Bush and Frederick Mosteller. Stochastic models for learning. 1955.

[17] Richard Bellman. The theory of dynamic programming. Technical report, DTIC Document, 1954.

[18] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[19] A Harry Klopf. Brain function and adaptive systems: a heterostatic theory. Technical report, DTIC Document, 1972.

[20] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* MIT press, 1992.

[21] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[22] Ian H Witten. The apparent conflict between estimation and controla survey of the two-armed bandit problem. *Journal of the Franklin Institute*, 301(1-2):161–189, 1976.

[23] J Holland. Adaption in natural and artificial systemsmit press. *Cambridge, MA*, 1975.

[24] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards.* PhD thesis, University of Cambridge England, 1989.

[25] Ronald A Howard. Dynamic programming and markov processes.. 1960.

[26] Andrew G Barto and Michael Duff. Monte carlo matrix inversion and reinforcement learning. *Advances in Neural Information Processing Systems*, pages 687–687, 1994.

[27] Kaddour Najim and Alexander S Poznyak. *Learning automata: theory and applications.* Elsevier, 2014.

[28] Mandayam AL Thathachar and Pidaparty S Sastry. *Networks of learning automata: Techniques for online stochastic optimization.* Springer Science & Business Media, 2011.

[29] John Oommen and Sudip Misra. Cybernetics and learning automata. In *Springer Handbook of Automation*, pages 221–235. Springer, 2009.

[30] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[31] VV Phansalkar and MAL Thathachar. Local and global optimization algorithms for generalized learning automata. *Neural Computation*, 7(5):950–973, 1995.

[32] Thomas H Cormen. *Introduction to algorithms.* MIT press, 2009.

[33] William Nell and Loren Shure. Memory profiling, March 15 2011. US Patent 7,908,591.

[34] John Platt et al. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998.

[35] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[36] Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *Journal of machine learning research*, 5(Jan):101–141, 2004.

[37] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[38] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145, 1995.

[39] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems. *J. Mach. Learn. Res*, 15(1):3133–3181, 2014.

[40] Tony Van Gestel, Johan AK Suykens, Bart Baesens, Stijn Viaene, Jan Vanthienen, Guido Dedene, Bart De Moor, and Joos Vandewalle. Benchmarking least squares support vector machine classifiers. *Machine Learning*, 54(1):5–32, 2004.

[41] Henry C Thode. *Testing for normality*, volume 164. CRC press, 2002.

[42] Douglas Curran-Everett and Dale J Benos. Guidelines for reporting statistics in journals published by the american physiological society. *American Journal of Physiology-Regulatory, Integrative and Comparative Physiology*, 287(2):R247–R249, 2004.

[43] DN Joanes and CA Gill. Comparing measures of sample skewness and kurtosis. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 47(1):183–189, 1998.

[44] Carlos M Jarque and Anil K Bera. Efficient tests for normality, homoscedasticity and serial independence of regression residuals. *Economics letters*, 6(3):255–259, 1980.

[45] Jim Lewsey. Medical statistics: a guide to data analysis and critical appraisal. *Annals of The Royal College of Surgeons of England*, 88(6):603, 2006.

[46] Samuel S Samuel S Shapiro. How to test normality and other distributional assumptions. Technical report, 1990.

[47] Gregory W Corder and Dale I Foreman. *Nonparametric statistics: A step-by-step approach.* John Wiley & Sons, 2014.

[48] George Brinton Thomas, Maurice D Weir, Joel Hass, and Frank R Giordano. *Thomas' Calculus Early Transcendentals.* Pearson, 2010.

[49] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification.* John Wiley & Sons, 2012.