Classifying Streaming Data using Grammar-based Immune Programming

by

Jaspreet Kaur Bassan

Bachelor of Engineering, University of Mumbai, 2014

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Program of

Computer Science

Toronto, Ontario, Canada, 2016

Classifying Streaming Data using Grammar-based Immune Programming

Master of Science 2016

Jaspreet Kaur Bassan

Computer Science

Ryerson University

Abstract

This work proposes a technique for classifying unlabelled streaming data using grammar-based immune programming, a hybrid meta-heuristic where the space of grammar generated solutions is searched by an artificial immune system inspired algorithm. Data is labelled using an active learning technique and is buffered until the system trains adequately on the labelled data. The system is employed in static and in streaming data environments, and is tested and evaluated using synthetic and real-world data. The performances of the system employed in different data settings are compared with each other and with two benchmark problems. The proposed classification system adapted well to the changing nature of streaming data and the active learning technique made the process less computationally expensive by retaining only those instances which favoured the training process.

Acknowledgements

I would like to express my deepest gratitude to my mentor and supervisor, Dr. Marcus Santos. This work would not have been successful without your consistent guidance and feedback. Thank you for providing me with a scholarship, for taking your valuable time out for all the meetings, and for having the coolest conversations!

I would also like to thank Dr. Eric Harley, Dr. Alireza Sadeghian, and Dr. Andriy Miranksyy, the members of my defence committee for taking time out to read my thesis and providing me with valuable feedback. A special thanks to Dr. Eric Harley for providing recommendation letters for various scholarship applications.

A big thanks to the administrators of the computer science department for helping me out with setting up the environment for my system and for various other hardware issues. I also thank the department assistants of the computer science department and Norm for helping me out with many things.

I thank my family, especially my mom, for constantly supporting me throughout the span of my master's degree. I thank all of my friends, especially at ENG251, for making the time spent at Ryerson more enjoyable. I thank my best friends across the oceans for keeping me sane throughout this time. Finally, I thank Jeff for his interesting insights and help and for the encouragement that kept me going!

Jaspreet Kaur Bassan
(Waiting for the thunderstorm to pass)
July 27, 2016

Dedication

To my grandparents

Contents

List of Tables

List of Figures

List of Appendices

Chapter 1

Introduction

Can we automatically generate solutions to classify streaming data in real-time?

1.1 Preamble

Recent advancements in ubiquitous computing[1] and storage technology have facilitated fast and continuous collection of data using automated sources like sensors, social networking sites or credit cards. Information mined could be potentially applied to any application. One can mine information for predicting weather, classifying a lake into a polluted or a non-polluted category, recognizing a criminal, and optimizing sensor data among other things. The fundamental objective of this work is to mine information from streaming data for the purpose of classification. Streaming data is an ordered sequence of data records that can be accessed for a limited number of times from a limited storage. Credit card transactions and sensor data are examples of streaming data. Classification is a process of identifying or predicting in which category a newly observed datum belongs to. Considering the credit card example, a classification system may classify each transaction into a suspicious or non-suspicious category in real-time.

The problem of inducing classifiers in offline environments[2] is well understood and has a long history. However, this is not the case with streaming environments[3]. It is infeasible to employ the existing offline techniques to streaming environments for the following reasons:

1. Streaming data is continuous; thus one can not afford to store all the historical data.

2. Streaming data may not be representative which could lead to biased models and therefore to incorrect predictions affecting the system's performance.

[1]Computing which can be used anywhere at anytime.

[2]Offline environments are static in nature where all the data are available at once.

[3]Streaming environments are dynamic in nature, where the data streams in real-time.

3. Streaming data is unlabelled. Although one can inexpensively access it, unlabelled data is of little use because one needs to label the instances either manually or computationally to train a classification system, and this might prove expensive.

4. Another major issue of streaming data is "concept drift." Concept drift causes data which is being classified to change in various unpredictable ways.

Machine Learning (ML) has been quite successful in mining information from data. One could employ a neural network, support vector machine, and evolutionary algorithms (EAs) among others for classifying data. EAs induce a classifier by evaluating the population against all the training instances for a number of iterations. This is a well understood problem in the case of an offline environment where all the training cases are available at once. On the other hand, one cannot employ an iterative approach in a streaming environment as a streaming classification system should output a result as soon as it receives the data. It is argued that EC approaches perform better in dynamic environments[4] because of the availability of a diverse set of solutions which potentially makes it easier to detect change, or to tackle concept drift [2]. In spite of the apparent reasons for using evolutionary inspired ML techniques that are able to develop self adapting solutions to their ever changing environments, little research has been done in EC in dynamic environments [3, 4].

Inspired from the biological immune system, Immune Programming (IP) is a metaheuristic which generates solutions automatically. It is similar to Genetic Programming (GP) in the sense that it also automatically generates solutions but employs a different search engine. IP employs CLONALG (see Chapter 2) in its search space. A Grammar-based IP (GIP) system is an IP system that utilizes a grammar to generate and preserve syntactically correct programs. It has been quite successful in regression problems, but has not been applied yet to classification problems. Also, the built-in evolutionary properties of CLONALG and GIP make them amenable to streaming environments.

1.2 Thesis Statement

In light of the limitations (mentioned in the previous section) around the creation of models that learn from a data stream, this thesis puts forth the following central hypotheses:

1. The automatic creation of classifiers that label the data could be done in real-time.

2. *Evolvability*[5] can aid in maintaining diversity which in turn could address concept drift.

We propose a novel technique for labelling uncertain instances and evolving classifiers in real-time using GIP. In this thesis, we make use of the Immune Programming terminology where a population of candidate solutions for a problem is known as a repertoire of antibodies (*Abs*), where repertoire is the population and *Abs* are candidate solutions. Training and testing data are known as Antigens (*Ags*). A

[4]Streaming environments fall under dynamic environments. The differences between these environments are explained in Chapter 2.

[5]Evolvability, as defined by Altenberg, "(is) the ability of a population to produce variants fitter than any yet existing" [5].

fitness measure assigned to the *Abs* is known as *affinity* which is based on the number of correct classifications. Initially, the system is exposed to a small set of labelled *Ags*. The system then generates a repertoire of *Abs* using a defined grammar and the repertoire evolves over time by subjecting the *Abs* to affinity computation, selection, cloning, hypermutation, affinity computation, reselection, and curbing. In the first few generations, the system behaves like an offline system as it learns from the labelled *Ags*. New unlabelled *Ags* are accepted from the raw streaming data into a buffer at every iteration. They are then labelled by a committee of *Abs* at fixed intervals. The entire process is repeated as long as the *Ags* keep streaming. Testing is performed periodically on the *Ags* available in the buffer.

Three variants of the GIP system are created to be employed in static and streaming environments. Out of the three variants, two variants are employed in the streaming environment: one for the labelled streaming data and another for the unlabelled streaming data. The variants are trained and tested using the Adult and Synthetic datasets. Several tests are performed to determine the performance and calculate the diversity of the variants. The variants are compared with each other, and the static and labelled streaming variants are also compared with the benchmark problem presented by Atwater et al in his works [6, 7]. The benchmark problem performed better than our static variant but our labelled streaming variant performed better than the benchmark problem. The labelling procedure was accurate as the unlabelled streaming variant performed similar to its labelled counterpart. Our labelling method is compared with the minimum-variance method presented by Zhu et al in his work [8]. Our system performed better than the second benchmark problem when in the case of concept drift. The experiments performed indicate the systems achieved sufficient diversity to maintain satisfactory performance.

1.3 Contributions

1. **Grammar-based Immune Programming for Classification Problems:** As of this writing, this is the first work to show that GIP can be successfully employed to automatically generate and evolve classifiers in static and dynamic domains.

2. **Active Learning:** Many works in this field assume that the streaming data is already labelled. Our work attempts to label streaming data while simultaneously evolving classifiers. We label the instances which are the closest to the decision boundary, i.e., we label the least certain instances for the classifiers to learn the most difficult data for the ease of classification.

3. **Classification System:** The software was developed using the *Racket* programming language on the Ubuntu operating system on a personal computer. Various variants of this system were developed for classifying data:

   (a) Static classification system: A GIP-based system that learns from static data.

   (b) Classification system for labelled streaming data: A GIP-based system that learns from labelled streaming data.

   (c) Classification system for unlabelled streaming data: A GIP-based system that labels and learns from unlabelled streaming data.

1.4 Roadmap

This thesis is organized in the following manner:

Chapter 2 explains the background work relevant to our work. It provides an overview of the properties of streaming environments related to dynamic environments and explains EC techniques with respect to dynamic environments.

Chapter 3 presents the materials and methodology used in this work. Note, the data used for training and testing fall under materials.

Chapter 4 analyses the results obtained from the three different variants of our implementation, and provides a comparison with the benchmark problems.

Lastly, Chapter 5 provides the work to be undertaken in the future which is followed by a brief conclusion.

For the ease of reference, the Index includes references to the pages where key terms are defined in the text.

Chapter 2

Background

Dempsey et al, in [4], defines a dynamic problem as *"a problem in which some element under its domain varies with the progression of time"*. The two important elements of a problem are solution space (or objective function) and the constraints. A solution space is the initial set of candidate solutions which satisfy the constraints of the problem to some degree. Constraints are conditions imposed by a problem. The elements in a static domain remain constant with the progression of time. Streaming problem is a type of dynamic problem where the data flows in continuously, which makes the need to update the solution space frequently. Whereas, in a traditional dynamic problem, the need to update the solution space may not be that frequent. An adequate amount of research of evolutionary algorithms (EAs) in streaming environments is lacking. However, as streaming problems fall under dynamic domains, it is safe to assume that the methodology adopted by EAs in streaming environments should be similar to those adopted in dynamic environments.

This chapter provides the necessary background information required for the understanding of this work. The following review of the literature leads up to our proposed methodology. The review starts with an overview of streaming environments and their properties, it then provides an overview of EAs in dynamic environments which is followed by a review of artificial immune systems (AIS). Some amount of research has been carried out for EAs in dynamic environments. No research of AIS in streaming environments has been performed. This chapter also interrelates EAs and AIS and highlights their similarities and distinctions.

2.1 Streaming Environments

The following subsections explain in more detail the elements of streaming environments.

2.1.1 Dynamic Properties of Streaming Data

Streaming data is a set of continuous and rapid data records, and we assume that there is no start or end to it. Therefore, we must impose a single pass constraint on streaming data which makes learning a continuous and an online activity assuming solutions of a problem are available at any time interval. Some partitions of the stream may not be representative, making a model likely ineffective on those parts. This issue, also known as *class imbalance*, is easily handled in offline environments using *stratification*[1] [9]. In streaming environments, however, it is not possible to stratify the data because of its continuous nature [2]. Hence, class imbalance is responsible for a decrease in the performance of the model in some partitions of the data stream. Also, the sources generating streaming data could introduce noise to it. This work, however, does not address noise management.

2.1.2 The Need for Active Learning in Streaming Environments

Many streaming applications assume that streaming data is labelled. In real-world applications, however, data received from various sources will not have a label already assigned to it. Streaming data must be labelled in the case of supervised learning models, and this process should be error-free as erroneous labels would deteriorate the model's performance [10]. The task of labelling data using an oracle (e.g. human or machine) is usually expensive [11] and thus one must determine when and which instances to label as it is unaffordable to label all the instances in every partition of the data stream. This process is known as label budgeting [12].

One of the simplest approaches to label budgeting is random sampling, where randomly selected instances are labelled by the model [2]. But randomness does not necessarily guarantee correctness. A better way of label budgeting is determining a few important instances from which the model can learn. This is the approach used in *Active Learning* (AL), a sub-field of ML, whose task is to employ a learning model to (truly) label a few instances which then leads to effective learning of the system [13, 14]. In AL, the learning model examines all the incoming instances, analyses the properties of the instances, and then requests the label of the instances which are deemed important by the model. It has to be decided beforehand if the model should label the most certain or the least certain instances or a combination of both [15].

Many AL methods choose to label the least certain instances since they are usually difficult to learn from because of their closeness to the decision boundary. Learning from the least certain instances can potentially lead to an improvement in the classification accuracy of the difficult instances. This property of the instances can be determined from various statistics, such as variance [16, 8], entropy [11], or posterior probability [17] of the model. An instance is deemed uncertain by computing the model's (or a committee of models) measure of uncertainty. For example, an instance is labelled by the model and added to the set of labelled instances temporarily. The model's entropy is computed after training the model using that set. This entropy value is then assigned to the instance. The instance with the highest entropy value is deemed the most uncertain and added to the set of labelled instances permanently.

Similarly, Zhu et al use the measure of variance in their work [8] to deem an instance uncertain. Our labelling approach is inspired from his work (called minimum-variance AL (MV)) and is therefore used as a benchmark problem to compare our results. MV employs C4.5 [18] and Naive Bayes [19] methods to classify data. It begins with randomly labelling a small subset of unlabelled data and predicts labels

[1]Stratification is the process of randomly shuffling training and/or testing data for handling class imbalance.

using an ensemble classifier which includes 10 classifiers. The same approach as explained in the above example is followed in the MV method of AL.

True labels can also be requested by detecting *concept drift* (explained in the next subsection). Concept drift is usually detected by detecting a decrease in the model's performance accuracy [15]. This method is computationally expensive, as a model's performance (and concept drift) is computed by using each incoming input as soon as the system receives it.

2.1.3 Concept Drift

Concept drift is perhaps one of the biggest issues of streaming environments. Concept drift occurs when the underlying source responsible for generating data changes due to unpredictable reasons thus producing abnormal data [2, 8]. Abnormal data affects the model's performance, so tackling concept drift in streaming applications is a must.

Consider the following example, to improve the control system of a boiler, it is important to predict the actual mass flow of the fuel from the sensors [20]. But due to fluctuations in electricity demand, the mass flow changes because of fuel-reloading and changes in the consumption speed thus causing a drift in the mass flow which characterizes a concept drift. Notice, however, that noise should not be recognized as concept drift because a model adjusted according to noisy data will produce erroneous results [21]. For example, faulty sensors will produce noise. Learning algorithms should not detect this as concept drift.

Concept drift can be sudden or gradual [22]. For example: after watching a series of videos on quantum computing on YouTube, the recommender will most likely recommend videos related to this topic. The person may gradually lose interest in quantum computing, thereby causing a *gradual* drift in her choices. Conversely, after watching a video on quantum computing, the person has *suddenly* drifted from this topic to a completely different one.

Various techniques periodically re-train their models to handle changes irrespective of detecting concept drift [23]. Whereas, other techniques re-train their models by detecting concept drift which is usually detected after requesting true labels. Concept drift can be detected by monitoring certain properties of the model (such as performance). A decrease in performance usually acts like a change indicator. In both cases, learning should be adaptive in order to handle concept drift. Various AL strategies are also capable of handling concept drift [15].

2.2 Metaheuristics

A metaheuristic is a set of ideas, concepts, operators, and a (search) heuristic usually employed by an optimization problem. Metaheuristics are usually machine learning methodologies, and as such, can be expressed using the equation, *Learning = Representation + Evaluation + Optimization* [24]. We briefly discuss the elements involved in *learning*:

- **Representation:** The candidate solutions in the search space must be represented using a data structure which can be handled by the computer. The representation chosen must incorporate all the attributes and the decisions of the training exemplars. Decision trees, neural networks, and hyperplanes are some of the examples of various representation schemes. Some algorithms employ a binary scheme for evaluating and optimizing their solutions. For example, many grammar-based algorithms use binary strings for optimization and an arbitrary language for evaluation. A different representation scheme could also be used to execute the solutions for performing some tasks. For example, a solution represented using binary strings could be executed as an electrical circuit.

- **Evaluation:** Learning algorithms must employ a fitness function (also known as an objective function) to determine good solutions from the poor ones. Depending on the type of the problem and the representation scheme utilized, a learning algorithm could compute information gain, posterior probability, or squared error to evaluate the candidate solutions. It is not uncommon for a learning algorithm to use different methods of evaluation for the solutions represented using a binary scheme.

- **Optimization:** A heuristic, such as gradient descent or greedy search, must be employed by the learning algorithm in its search space to determine the best solution(s) in order to improve it by introducing some variations to it. Selection and variation are the two elements of optimization.

Lately, metaheuristics such as Evolutionary Algorithms (EAs), Artificial Immune Systems (AIS), and Ant Colony Optimization algorithms among others have been successfully employed in a range of problems from classification to pattern recognition to computer security. In this work, we focus on two metaheuristics, EAs and AIS, and on their underlying methodologies shown in the taxonomy depicted in Figure 2.1.

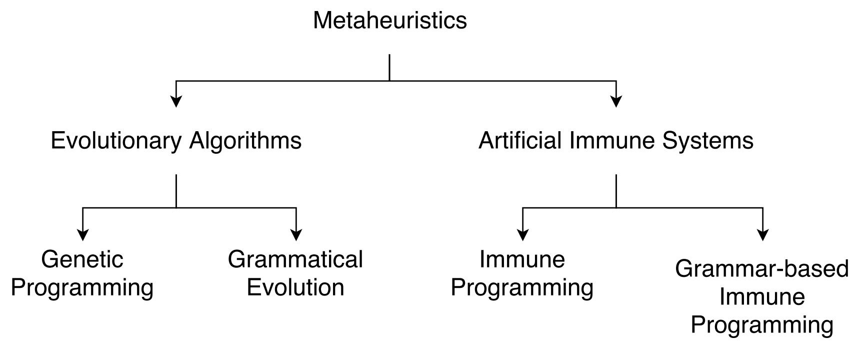


Figure 2.1: Metaheuristics

A comparison between EAs and AIS metaheuristics is shown in Figure 2.2. Both EAs and AIS are biologically-inspired population-based metaheuristics. EAs are inspired from Darwin's theory of evolution whereas AIS is inspired from the vertebrate immune system. EAs and AIS have an implicit memory, as the process of evolution retains some past information. The heuristic utilized in EAs is based on a global selection mechanism as the fittest individual is always selected for genetic variations.

In contrast, AIS utilizes a local selection mechanism whereby individuals are cloned and mutated based on their fitness with respect to the training dataset. An explanation of these heuristics can be found in Sections 2.3.2 and 2.4.2. Grammar-based metaheuristics have an advantage of generating evolvable solutions by the means of *degenerate genetic code* in which neutral mutations affect the genotype without affecting the phenotype of the solution. Section 2.3.3 provides more information about it.

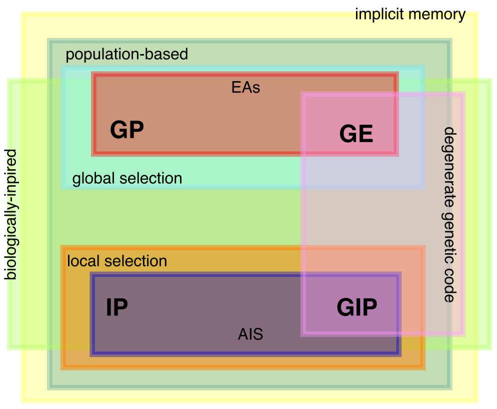


Figure 2.2: Distinctions between EAs and AIS

2.3 Evolutionary Algorithms

Humans have come a long way; from being capable of making fire to being capable of controlling drones with their brains. Evolution is intrinsic. Almost everything on Earth such as different species, their immune systems, religions, languages, or cultures is evolving constantly. Adaptation to external environments, natural selection, reproduction, and survival of the fittest are the fundamentals of evolution [25]. Evolutionary Computation (EC) encompasses many machine learning (ML) methodologies that apply the principles of Darwinian theory of evolution [26] and survival of the fittest theory [27] to produce solutions for various problems. These methodologies are known as Evolutionary Algorithms (EAs) and they follow a three-step iterative process of evaluation, selection, and random variation [28], as shown in Figure 2.3.

The methods employed for evaluation and/or optimization differentiate EAs from traditional ML techniques (see the learning equation from Section 2.2). The method of evaluation and selection in EC is usually "tournament-based", i.e., a candidate solution has to be better than another solution, unlike in traditional ML techniques, where the cost function is usually linear or gradient-based seeking a maxima/minima [28]. Also, EAs start from a diverse set of randomly generated solutions converging its way to the most optimum solution unlike traditional ML, where the algorithm starts off by adjusting the same solution to its optimum which was generated in the beginning [25]. Learning in EC techniques is generally supervised, although they can be applied to unsupervised learning domains [29].

The diagram shown in Figure 2.3 captures the essence of an EA. Many EAs such as Evolutionary

Strategies, Evolutionary Programming, Genetic Algorithms, Genetic Programming (GP), and Grammatical Evolution (GE) have gained popularity mostly because of their success in optimization problems [30, 31, 32]. Correct formulation of representation and evaluation of candidate solutions make them successfully employable to a plethora of applications such as data mining, computer vision, architecture, medicine, and robotics [31]. We briefly compare GP and GE systems using the learning equation as they form the core of our thesis; these systems are explained in depth in the next subsections.

- **Representation:** An EA, usually, randomly generates a population of candidate solutions only once. Candidate solutions in GP are usually represented by *syntax trees*. On the other hand, variable length binary strings are used in GE. The binary strings are mapped to an arbitrary language using a *genotype-to-phenotype mapping*.
- **Evaluation:** The generated candidate solutions must be evaluated. In GP, syntax trees are evaluated using the training dataset. On the other hand, GE evaluates the mapped output.
- **Optimization:** The best evaluated solutions must be optimized. In GP, the evaluated syntax trees are selected using a tournament selection to introduce variations to them using sexual genetic operators. The same process is carried out in GE, but instead of optimizing the arbitrary language, binary strings are optimized.

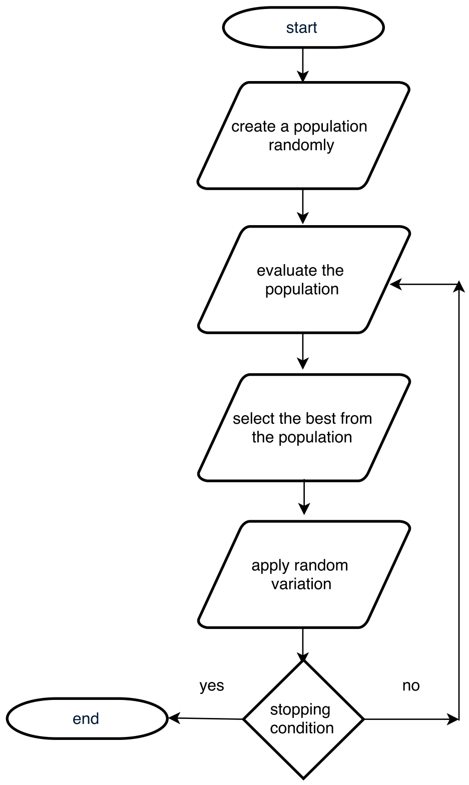


Figure 2.3: Generic flow followed in an EA.

2.3.1 EAs in Dynamic Environments

In a static environment, the converged solution should attain a (close to) optimal state [4]. However, for dynamic environments, instead of aiming for an optimal state, we must aim for an evolvable solution (which could be not the most optimum) to promote survivability [3]. A converged solution will inhibit the model to adjust to change which could cause the system to fail outright. The following five approaches can be adopted to overcome this issue but the focus of this thesis substantially lies in the first three approaches [4]:

1. Memory EAs should adopt mechanisms to recall important past models. This can be done explicitly or implicitly. Explicit methods directly store important models in an archive from which a population is re-initialized. On the other hand, implicit methods are based on *degenerate genetic code* explained in Section 2.3.3.

2. Diversity The availability of a diverse set of candidate solutions and their ability to adapt by evolving make EAs potentially amenable to dynamic environments [3]. Maintaining diversity while detecting change is of primary importance for EAs in dynamic environments [33]. One of the ways of maintaining diversity is by incorporating *Neutral Mutations* (explained in Section 2.3.3) in optimization process.

3. Evolvability Evolvability is the ability of the population to produce fitter offspring which can in turn introduce diversity. Some representation schemes can aid in achieving evolvability, thus one must carefully select the representation of the candidate solutions. Genetic variations can also be responsible for achieving evolvability because of neutrality and genotype-to-phenotype mappings (explained in Section 2.3.3) [2].

4. Multiple Populations: The population can be divided into smaller populations which can be evolved simultaneously. Maintaining sub-populations divide the search space and each sub-population will have its best solution. Thus, preserving multiple populations can prevent the system from converging to one optimum solution.

5. Problem Decomposition In dynamic environments, a problem is said to be decomposed whenever the environment changes. The models learned (until the detected changes) possess memory as they have been through the evolutionary process. Structural knowledge can be extracted from the learned models to adapt efficiently to the new decomposed problem. Some methods opt for generating a new population after detecting changes [4]. It is impossible to attain a converged state of optimality if a new population is generated every time after the environment changes.

We must change the existing approaches undertaken to employ EAs in dynamic environments. “Survival” should be the primary objective of EAs, so the solutions need not be the most optimum, they need to be better than the competing candidates. EAs should be effectively employable because of the dynamic properties of these environments and not despite of them [34].

2.3.2 Genetic Programming

Genetic Programming (GP) is an EA known for automatically generating computer programs [3, 29, 31, 32]. Computer programs could be (for example) classifiers whose task is to classify a datum. Given a high-level problem statement, the GP system determines the structure and the size of a program which is the most suitable to the environment [32]. GP often employs a genetic algorithm (GA) [35] as its search heuristic. Programs or parents are selected using a tournament selection to produce offspring using genetic operators (comprising of crossover and mutation.)

Using the equation mentioned in Section 2.2, learning in GP can be expressed as follows:

- **Representation:** In GP, candidate solutions to a problem are usually represented as *Syntax trees.* Other forms of representation are linear lists, graphs, linear chromosomes, or a combination of these. A *program* is usually constructed from a *function set* and a *terminal set.* The construction of function and terminal sets are application dependent; in a numerical application, the function set comprises of arithmetic operators whereas the terminal set comprises of variables and constants.

- **Evaluation:** Representation defines the search space of the GP system. The candidate solutions must be evaluated to determine their *fitness.* Computing fitness in a GP system is also application dependent. For example, in a classification system, the fitness of a candidate solution can be computed by evaluating it against a training set to determine the number of correct classifications. Evaluation paves the way for *Optimization.*

- **Optimization:** *Selection* and *Variation* processes optimize the candidate solutions, by producing evolved offspring. Genetic operators, crossover and mutation, are employed to produce "offspring". In crossover, two (or sometimes three) "parents" are probabilistically selected to engage in *subtree crossover* which is the most commonly employed crossover function. The subtree of parent 1 (selected randomly) is swapped with the subtree of parent 2 to create a new offspring. *Point mutation* changes the node randomly selected with the possible values from the function or the terminal set.

  It is not efficient to search the entire search space for producing offspring, therefore it is vital to use some kind of a selection process. *Tournament selection* is frequently employed by GP systems where one or more programs are selected randomly and evaluated to determine which is one is the fittest to be a parent(s).

Genetic Programming in Dynamic Environments

Enough research has not been carried out to analyse the behaviour of GP systems in dynamic environments. GP systems are population-based which is the key factor required to survive in changing environments as a diverse set of candidate solutions can solve multiple targets [3]. As discussed before in Section 2.3.1, GP systems should perform well because of the properties of these environments. Koza suggests the use of the mutation operation in GP systems to be useless, as the crossover operation can solely handle the system from prematurely converging to a local maxima [31]. But the mutation operation along with crossover in dynamic environments could provide more support for tackling premature convergence, as mutation (usually used in genetic algorithms) is responsible for restoring the diversity of the population.

Parameter setting plays a very important role in dynamic environments. Population size should be large enough to support a diverse set of candidate solutions and the probability values of crossover and mutation should be carefully selected to promote diversity. A decrease in fitness may require the system to increase its mutation rate [36]. An efficient way of parameter selection is adapting parameters dynamically during the runs rather than setting them statically which could be time consuming. Control parameters can be adapted deterministically, through a feedback from the system, or they could be evolved during the execution of the GP system [37].

Few methods [12, 6], use the concept of Symbiotic Genetic Programming where the training data and the learner population (candidate solutions) are co-evolved simultaneously and the past important training instances are retained in the archive. Control parameters used in this system are inherited in our implementation as we utilize this system as a benchmark system mostly because of its success in streaming environments.

2.3.3 Grammatical Evolution

In computing, grammars have been used to restrict the structure of languages thereby causing the production of syntactically correct programs. In GP, genetic variations could potentially destroy the structures of the programs. Various grammar-based GP systems have been introduced to handle this problem of "closure[2]" which results in the generation and preservation of valid programs [31, 38, 39, 40]

Grammatical Evolution (GE) is a grammar-based EA used to automatically generate and evolve programs in an arbitrary language using a Backus-Naur Form (BNF[3]) grammar [38]. GE is usually employed as a grammar-based GP system but it can utilize any EC technique in its search space. It is known for outperforming GP in many domains.

In GE, learning involves the following three elements:

- **Representation:** In GE, the candidate solutions are represented using variable length binary strings which are then mapped, through a grammar, to programs in a language.
- **Evaluation:** Although the candidate solutions are represented using binary strings, GE evaluates the mapped programs.
- **Optimization:** GE employs a GA in its search space. It optimizes its binary strings using crossover and mutation. It utilizes a tournament-based selection mechanism.

[2]The destruction of a program's structure because of variation is called the closure problem.
[3]According to Wikipedia, BNF grammar is a context-free grammar used to describe the syntax of languages used in computing.

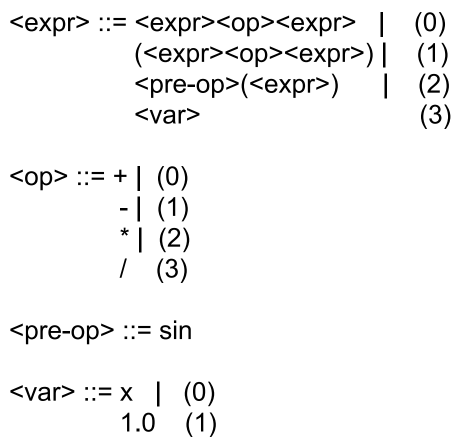
Genotype-to-Phenotype Mapping

GE is similar to GP, but it employs a different representation methodology and a mapping function to map a variable length binary string to an arbitrary language.

GE is largely based on the biological process of generating proteins from its genetic material [41]. In biology, the DNA of an organism (the genetic material) is called the *genotype* and the physical trait (e.g. eye color) is called the *phenotype*. Similarly, in GE, variable length binary strings are called genotypes whereas the actual programs are called phenotypes and the process of generating programs from binary strings is called *genotype-to-phenotype mapping*.

BNF grammar is expressed by the tuple $\langle\ N,\ T,\ P,\ S\ \rangle$, where N is a set of non-terminals, T is a set of terminals, P is the set of production rules, and S is the start symbol (also a non-terminal). The BNF grammar, for example, for a regression problem can be expressed as follows [38]:

$$
\begin{aligned}
&\mathrm{N} = \{\text{expr, op, pre-op}\}\\
&\mathrm{T} = \{+,\ -,\ *,\ /,\ \text{sin},\ 1.0,\ \text{x}\}\\
&\mathrm{S} = \{\text{expr}\}\\
&\mathrm{P} =
\end{aligned}
$$



```
<expr> ::= <expr><op><expr>   |   (0)
           (<expr><op><expr>) |   (1)
           <pre-op>(<expr>)   |   (2)
           <var>                  (3)

<op> ::= + |  (0)
         - |  (1)
         * |  (2)
         /    (3)

<pre-op> ::= sin

<var> ::= x   |  (0)
          1.0    (1)
```

Figure 2.4: An example of the BNF grammar.

Each option in P (delimited using '|') is numbered sequentially from 0. The genotype of each candidate solution is a variable length binary string, for example, 000101011011011101100000. A *codon* is consecutive 8 bits grouped together from the binary string. Equation 2.1 is used to select a production rule in the mapping procedure.

$$x \mod n \tag{2.1}$$

where,
x is the integer value of a the codon being read,
n is the total number of rules of the current non-terminal.

If the codon, 00010101, was being read while the start symbol was selected, rule number 1 '(⟨expr⟩⟨op⟩⟨expr⟩)' would be selected because:

$$00010101_2 = 21_{10}$$
$$21 \mod 4 = 1$$

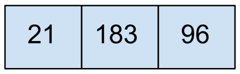


Figure 2.5: Codons of the individual 000101011011011101100000.

If the individuals run out of codons, the codons are re-used by wrapping the genome which is based on a process known in biology as *gene-overlapping* [41]. Revisiting the above example, ⟨expr⟩ was derived to (⟨expr⟩⟨op⟩⟨expr⟩). Always considering the left-most non-terminal, ⟨expr⟩ would be derived again, this time using the second codon (183) as shown in Fig 2.5. $183 \mod 4 = 3$, so the last rule ⟨var⟩ is selected. Now, ⟨var⟩ is derived using codon 96. $96 \mod 4 = 2$, so rule 2 (which is 1.0) is selected. After many derivations, the phenotype '(1.0 − 1.0)' is mapped from its genotype, '000101011011011101100000'.

In the above example, albeit same rule number was generated, the non-terminal under derivation was different, thus leading to different derivations. But there could be cases where the same rule under the same non-terminal is being selected over and over again, thus creating a long repeating branch in the parse tree. In extreme cases, the non-terminal under derivation is never derived which results into invalid phenotype structures. If these cases are true, the individual is deemed extremely unfit in order to purge it out eventually.

Point mutations on an individual's genotype may change the codon but may select the same production rule thereby producing the same phenotypes thus maintaining genetic diversity. This phenomenon is also observed in many organisms and is called *degenerate genetic code* [41]. Such mutations are called *neutral mutations*[4]. A neutral mutation on the third bit of the genotype (flipping 0 to 1), as shown in Fig 2.5, will lead to the codon 53. The modulus of 53 to 4 is also 1 (as opposed to the modulus of 21 to 4) thereby selecting the same rule (rule number 1). O'Neill and Ryan point out that the Kimura's *neutral theory of evolution*[42] suggests that neutral mutations are responsible for maintaining genetic diversity in natural populations which has been exhibited in GE [38].

[4]Neutral mutations do affect the search space but they do not affect the solution space.

Grammatical Evolution in Dynamic Environments

Genotype-to-phenotype mappings in EC may have beneficial properties under dynamic domains as it makes convergence on a genotypic level difficult to achieve [2, 4, 43]. GE encompasses features that address the first three approaches to handle the converging problem making (as noted in Section 2.3.1) it very suitable for dynamic environments. Implicit memory is incorporated in the individuals through degenerate genetic code in genotype-to-phenotype mappings [4, 38]. Diversity is promoted and maintained through genetic variations and neutral mutations. The separation of search and solution spaces may also aid in promoting diversity. Both neutral mutations and genotype-to-phenotype mappings aid in evolvability [2]. BNF grammars are also known to exploit the evolvability of the individuals [4]. GE can be made more powerful in dynamic environments by evolving the grammar which consequently evolves its genetic code thereby promoting adaptability [44].

2.4 Artificial Immune Systems

The vertebrate immune system is diverse, dynamic, and adaptive as it is evolving constantly to fight against a plethora of pathogens [45]. Artificial Immune System (AIS) is an interdisciplinary area of research that explores the commonality among the vertebrate immune system, computer science, and engineering [46, 47]. Four major AIS algorithms have been extensively being used for various computer security and machine learning applications: 1) Clonal Selection Algorithm [1], 2) Negative Selection Algorithms [48], 3) Artificial Immune Networks [49], and 4) The Danger Theory and Dendritic Cell Algorithms [50, 51]. Although being standalone computational research areas, EC and AIS are very similar to each other as they are both inspired from an evolutionary approach. The following advantages make AIS better than EAs:

1. AIS does not face the problem of "pre-convergence", unlike EC, as the absence of a termination-condition helps maintain diversity because of the inclusion of random individuals in the solution space.

2. The methods of evaluation and optimization are based on a local selection mechanism which has been deemed necessary to maintain diversity [52].

EAs and AIS have more similarities than distinctions. So, it is safe to assume that the process undertaken by AIS to solve dynamic problems should be similar to EC.

2.4.1 Immune Programming

Immune Programming (IP) is an area of research which bridges the gap between EC and AIS [53]. IP, like GP, automatically generates programs which in turn are capable of solving a wide area of problems. IP uses AIS as its search algorithm, unlike GP which uses an EA (usually GA). The clonal selection algorithm (CLONALG) is widely used in IP. CLONALG is based on the Clonal Selection Theory proposed by Leandro N. de Castro and Fernando in [1].

In that theory [54], Antibodies (*Abs*) are proteins produced by the B-cells (B lymphocytes) which bind to the surface of an antigen (*Ag*). *Abs* are found on the surface of a B-cell. *Ags* are molecules which intrude the body to cause harm. The primary task of an *Ab* is to identify an *Ag* and bind to it. When an *Ag* enters the blood stream, it will be attached to a lymphocyte causing the cell to activate and proliferate to produce descendants, or *Abs*. One type of clone called terminal cell, is selected to kill the *Ag*. Another type of clone, known as B memory cells, is selected to acquire immunity against it. B memory cells produce very high affinity *Abs* in case of a second antigenic invasion.

In IP, a repertoire of *Abs* is a set of candidate solutions, *Ags* are training and testing data. Like any ML methodology, learning in IP is achieved by a design that combines representation, evaluation, and optimization (see the learning equation stated in Section 2.2).

- **Representation:** Similar to GP, IP programs are usually represented using syntax trees. Other representations such as S-expressions and stack-based frameworks are also employed [53].
- **Evaluation:** *Affinity*, a measure of fitness, is employed by an IP system to evaluate its candidate programs. It is the measure of how well an *Ab* binds to the surface of an *Ag*. Usually, the *affinities* of *Abs* are calculated by exposing a randomly selected *Ag* to all the *Abs* in the repertoire. *Affinity* is usually derived from the raw fitness of an *Ab*. As noted in Section 2.3.1, survival should be the primary focus of dynamic algorithms. Therefore, this measure of fitness could prove beneficial in dynamic environments (and thus streaming environments) as *Abs* with a higher probability of survival have higher *affinities* than other *Abs* thereby increasing their chances of selection and variation.
- **Optimization:** As mentioned before, IP employs CLONALG in its search space. CLONALG in turn uses two asexual search operators to evolve *Abs*: *Cloning* and *Hypermutation* and its combined effect is known as *affinity maturation*.

  (a) **Cloning:** IP employs cloning so that the variations employed on *Abs* to produce offspring do not destroy the structure of the parents. An IP system selects a fixed number of *Abs* with the highest *affinity* for cloning. Cloning is made directly proportional to the *affinity* of the *Ab* in order to make more replicas of the fittest *Abs*.

  (b) **Hypermuation:** Clones generated undergo hypermutation. Hypermutation is a process of changing every gene (bit) in a genome (binary string) of an *Ab* using a probabilistic factor. Introducing too many changes in fitter *Abs* could decrease their *affinities*. Thus, the factor at which the clones are mutated is inversely proportional to its *affinity* value, so that more changes are introduced in less fitter *Abs* to possibly increase their *affinities*.

  In a converged population of a GP system, crossover between parents may not introduce diversity and may exploit the structure of the individuals. Hypermutation helps maintain a balance between exploration and exploitation, and aids in restoring diversity [4]. Recall that maintaining diversity is vital in dynamic environments. This built-in search operator of CLONALG can prove advantageous in dynamic environments.

After introducing variations, an IP system selects some mutated clones with the highest *affinity* and adds them to the repertoire of *Abs* and finally, *d Abs* (where *d* is usually a small number) with the *d* lowest affinity are replaced with newly and randomly generated *Abs*. Randomly generated *Abs* are added to the repertoire of the candidate solutions at the end of every generation to increase the search space of the solution. The above process can be understood clearly using the flowchart in Fig 2.6.

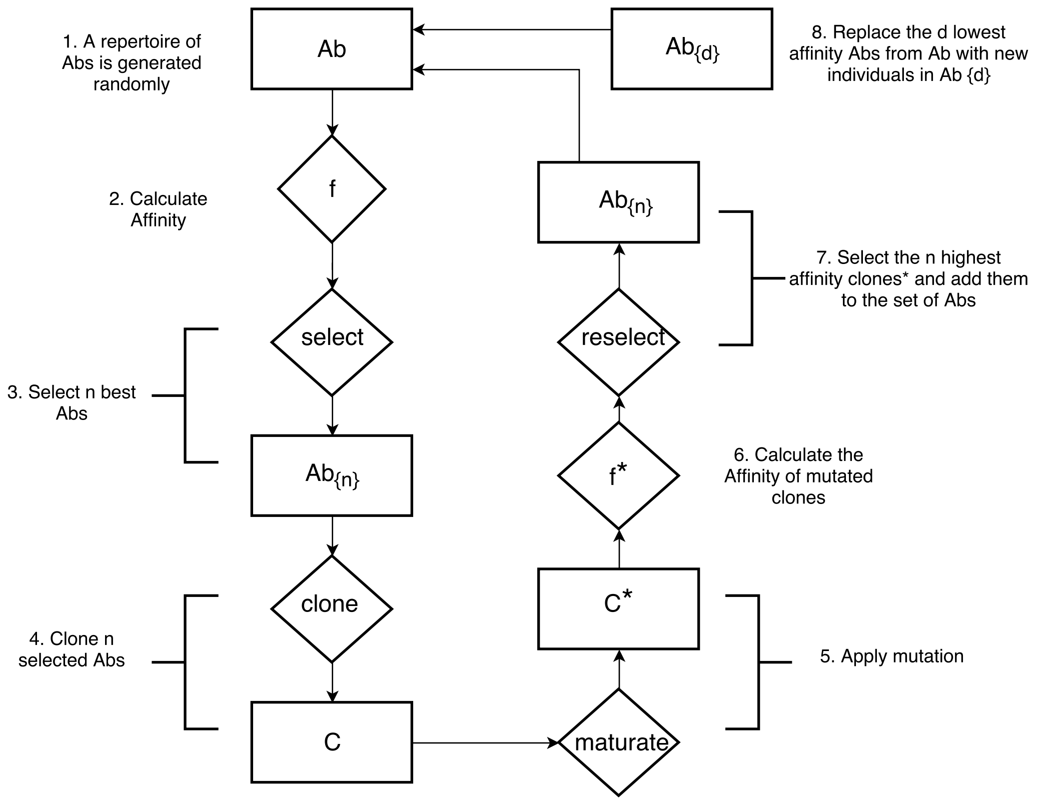

1. A repertoire of Abs is generated randomly
Ab
Ab{d}
8. Replace the d lowest affinity Abs from Ab with new individuals in Ab {d}
2. Calculate Affinity
f
Ab{n}
7. Select the n highest affinity clones* and add them to the set of Abs
select
reselect
3. Select n best Abs
Ab{n}
f*
6. Calculate the Affinity of mutated clones
clone
C*
4. Clone n selected Abs
5. Apply mutation
C
maturate

Figure 2.6: Generic flow of CLONALG [1].

2.4.2 CLONALG for classification

CLONALG was originally proposed for pattern matching tasks and was later adapted for optimization tasks [1]. The past decade has witnessed the implementation of many variants of CLONALG for the task of classification [55, 56, 57, 58]. Two of these methods are worth brief descriptions:

Clonal Selection Classifier Algorithm

Clonal Selection Classifier Algorithm (CSCA) [57] follows a very simple approach: like a standard EA and CLONALG algorithm, it begins with randomly initialising a repertoire of *Abs*. The entire repertoire is evaluated by exposing it to all the *Ags* using the measure of affinity which is the number of correct classifications and the number of misclassifications. Before introducing variation, the *Abs* are selected for pruning. *Abs* which classify all the *Ags* correctly and those whose affinity is less than a certain threshold are removed permanently from the population. Affinity of the *Abs* with zero correct classifications and the *Abs* with more than zero misclassifications is recomputed after adjusting their classes with the majority class or the first one in the case of a tie. *Abs* undergo cloning and hypermutation and hypermutated clones are added to the repertoire of *Abs*. The repertoire is pruned once again using the same procedure mentioned before. This entire process is repeated for a predetermined number of generations. And finally, the evolved classifiers are applied to the testing data, in the case of an unclassified *Ag*, some best classifiers determine its class value by a majority vote. CSCA (barely) outperformed CLONALG in four datasets out of five. An implementation of CSCA and CLONALG can be found in Weka.[5]

CLONAX

This variant of CLONALG has been developed for classifying data [58]. It is inspired from k Nearest Neighbour and its main task is to the detect the structure of the *Ags* by evolving its classifiers accordingly. A repertoire of Ab_m (generated randomly) consists of two sub-repertoires: Ab_c and Ab_r where c is a class value and r is the remaining *Abs* from Ab_m. The following process is carried out for a predetermined number of generations: The class of an Ag_i selected is determined and the Ag_i is exposed to Ab_c (of the same class) and Ab_r. Affinity of the Ag_i is computed with respect to Ab_m and Ab_r. Clones of n fittest *Abs* are generated and mutated. For the purpose of local search, k highest hypermutated clones are selected which are further evaluated using Ag_i. Their affinities are replaced by the averaged affinities of the p nearest *Ags* having the same class value for producing generalized classifiers to attract *Ags* of the same structure. Noise is filtered by discarding the clones whose average affinity is lower than at least two *Ags* of different classes and if one *Ag* has a higher affinity than any of the clones, then that *Ag* is discarded from the training set. Memory is updated by replacing the lowest affinity *Ab* of the existing Ab_c with the highest averaged affinity clone if the latter has a higher affinity than the former. Finally, d lowest affinity *Ab* are replaced from Ab_r with randomly generated *Ab*. CLONALG and CSCA achieved better classification accuracy than CLONAX in three datasets out of five.

[5]Weka is a data mining software in JAVA.

2.4.3 Grammar-based Immune Programming

Grammar-based Immune Programming is a metaheuristic comprising of GE and IP elements [59]. It is an IP technique, which makes use of a BNF grammar to map variable length binary strings (genotype) to an arbitrary language (phenotype). GIP utilizes the same *genotype-to-phenotype mapping* as introduced in GE (see Section 2.3.3). A GIP system can be summarily understood using the learning equation defined in Section 2.2 as follows:

- **Representation:** Analogous to GE, the genotype of the *Abs* are represented using variable length binary strings and the phenotype of the *Abs* are represented using an arbitrary language.
- **Evaluation:** A measure of affinity is used to evaluate the phenotype of the *Abs*.
- **Optimization:** GIP employs CLONALG as its search engine. The genotype of the *Abs* are optimized using asexual search operators of cloning and hypermutation (as explained before.) A few best hypermutated offspring are added permanently to the repertoire. The least fit *Abs* are purged out at every generation by replacing them with randomly generated *Abs* to maintain diversity.

GE does not have provisions to address the generation of invalid programs (see Section 2.3.3). GIP uses a *repair method* to prevent the mapping function from generating invalid programs. So far, GIP has been applied to regression problems and has successfully outperformed GE on various symbolic regression problems [60].

Chapter 3

Methodology

The primary focus of this work was to automatically generate solutions to classify streaming data in real-time. Streaming data needs an interface mechanism, is unlabelled, and has drifting concepts. We employed a *sliding window* as an interface that controls the continuous nature of the data stream, adopted an active learning approach to label the data, and assumed that achieving evolvability can sustain diversity which can result in addressing concept drift. Evolvability induces offspring that affine to the training data more than their parents and it promotes adaptable evolution resulting in the maintenance of diversity.

We employed GIP to automatically generate and optimize classifiers in real-time. GIP was deemed the fittest among all the other algorithms for the following reasons (as noted in Chapter 2):

1. An AIS algorithm is more capable (than EAs) of generating and sustaining a diverse set of solutions. Maintaining diversity is critical in dynamic problems, thus an AIS algorithm seems more appropriate than its counterpart.

2. Genotype-to-phenotype mappings, degenerate genetic code, and BNF grammar promote evolvability.

3. A separation of search and solution spaces also aids in promoting diversity.

This chapter provides a detailed description of the methodology undertaken to implement this work. Inspired from the learning equation defined in Section 2.2, this chapter is modularized into three main sections: Representation, Evaluation, and Optimization. We begin by depicting the high-level flowchart of our software (see Figure 3.1) and provide pseudo code for the key sub-routines highlighted in Figure 3.1 with a box with left and right margins. We also provide an overview of the data used for training and testing our system. Any process that relates to training or testing data falls under the Evaluation section, as the candidate solutions are evaluated against these datasets.

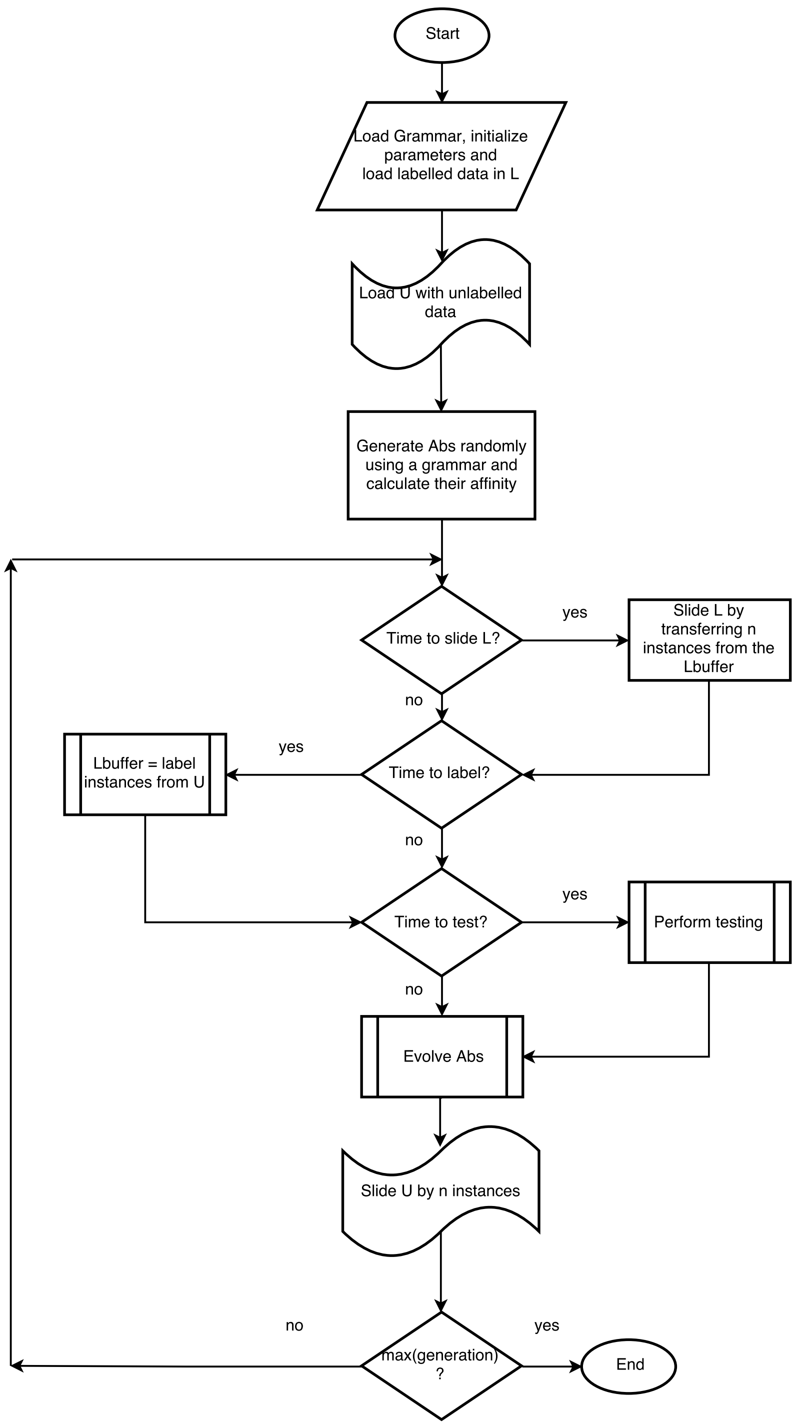

Start
Load Grammar, initialize parameters and load labelled data in L
Load U with unlabelled data
Generate Abs randomly using a grammar and calculate their affinity
Time to slide L?
yes
Slide L by transferring n instances from the Lbuffer
no
Time to label?
yes
Lbuffer = label instances from U
no
Time to test?
yes
Perform testing
no
Evolve Abs
Slide U by n instances
max(generation) ?
no
yes
End

Figure 3.1: Main method

The processes depicted in Figure 3.1 are not necessarily explained in the same order as shown in the figure, but are explained in their appropriate sections.

In a nutshell, the high-level algorithm depicted in Figure 3.1 works as follows: Assume L and U are buffers representing sliding windows that follow a FIFO policy where L is the labelled set of instances and U is the unlabelled set of instances. The first step was to load the grammar and initialize all the parameters (such as population size, mutation rate, number of clones, and so on.) Then we loaded the labelled data into the L set and unlabelled data (which is the streaming data) into U. We then generated a repertoire of *Abs* (candidate solutions) using the grammar and calculated their affinities. If *time to slide* L is true, then n new instances from the *Lbuffer* are transferred into L by replacing its oldest n instances. If *time to label* is true, we label some instances from U using an Active Learning method. If *time to test* is true, testing is performed. The *Abs* undergo evolution using the selection and genetic operators, and finally U slides at the end of every iteration. The entire process repeats as long as the instances are available in U.

3.1 Representation

An appropriate representation must be selected which can be easily handled by the computer. A correct representation is crucial to ensure that the models learn effectively from the environment, especially when the models have to undergo rigorous genetic variation.

3.1.1 Grammar

The first and foremost step in GIP is to decide the grammar to be used for generating the arbitrary language of the candidate solutions. GIP usually employs a BNF grammar which is known to promote evolvability. The BNF grammar can be expressed using a tuple $\langle\ N,\ T,\ P,\ S\ \rangle$, where N is a set of non-terminals, T is a set of terminals, P is a set of the production rules, and S is the start symbol. For each training dataset, we constructed a unique grammar. An example of this can be seen in Figure 3.2.

The production rules of the grammar ensure legality of the language's syntax. We also generated non-recursive rules from the grammar. This non-recursive grammar ensures that derivations can always terminate in a string of terminal symbols.

```
N = { Expr, Op, UnitOp, Var }

T = {x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, +, -, *, /, cos, sin, log, exp }

S = { Expr}

P =

Expr := Op Expr Expr l        (0)

            UnitOp Expr l       (1)

            Var                 (2)

Op ::= +  l  -  l  *  l  /

         (0) (1)  (2) (3)

UnitOp ::= cos l sin l log l exp

              (0)    (1)    (2)    (3)

Var ::= x1 l x2 l x3 l x4 l x5 l X6 l X7 l x8 l X9 l X10

          (0)  (1)  (2)  (3)  (4)   (5)   (6)   (7)   (8)   (9)
```

Figure 3.2: Grammar constructed for the synthetic dataset

Each *Ab* generated by our system have two parts, *genotype* and *phenotype*.

3.1.2 Genotype: Variable-length Binary Strings

The genotypes (also known as genomes) of the *Abs* are represented using a variable length binary string. The genotypes of each candidate solutions undergo *optimization*. The genotype generated is a complete set of genes, which is a factor of total number of genes and its size. An example of the genotype can be seen in Figure 3.5. The size of the genome was initialized to 240; which is the factor of 30 genes and each gene contains 8 bits. Although we initialize the genome using a fixed-length list, what makes them variable is their ability to *wrap* during the mapping procedure. This is similar to the operation of circular queues. The binary string was generated randomly using Racket's inbuilt random function which employs a 54-bit version of L'Ecuyer's MRG32k3a algorithm [61].

3.1.3 Phenotype: Decision Trees

The phenotype of each *Ab* is represented using decision trees (DTs). DTs are a powerful tool in supervised learning which employ the divide and conquer strategy on the given data [9]. Each internal terminal node of the DT represents an attribute, the node's branches represent the attribute values, and the terminal nodes represent the decisions to be taken which are the classes of the training dataset. The values of a data instance are plugged into the DT which are compared with the values in the DT. The route taken by an instance yields a decision or a class value and is deemed as a correct match (see Figure 3.3). Terminals 0 and 1 represent the class values.

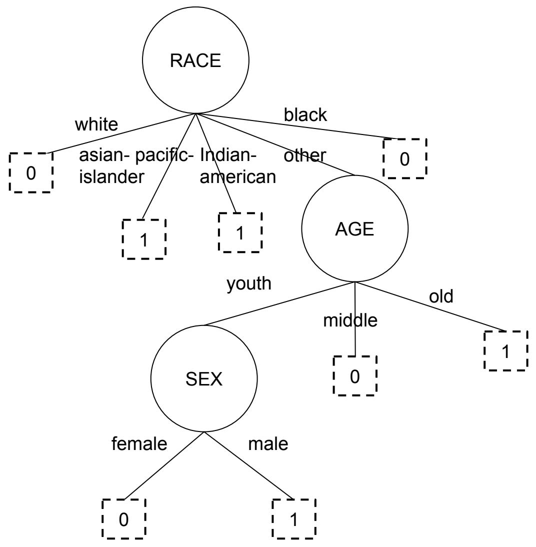


Figure 3.3: An example of the phenotype of an *Ab* generated randomly of adult-dataset explained in Section 3.2.4.

Consider the grammar defined for the adult dataset in Appendix A. In our defined grammar, the attributes and the classes of the training dataset are the terminals. The non-terminal DT is derived into an attribute which includes the attribute name and the non-terminal DTp whose number of occurrences indicate the total number of the possible values of that attribute. DTp is further derived into the non-terminal DT or the class 0 or 1 which are the terminals. For example, the first production rule AGE DTp DTp DTp indicates that the attribute AGE has three possible values which are youth, middle, and old. Figure 3.4 illustrates how the decision tree in Figure 3.3 is derived. The terms represented in dotted lines represent the terminals. Note that the branches (the possible values of an attribute) of the DT in Figure 3.3 are never really defined in the grammar because the system is programmed to know that the n^{th} class value represents the n^{th} value of the attribute.

Genetic programs and immune programs are usually represented by syntax trees [31]. Thus, using DTs as classifiers is an obvious choice as it simplifies and eases the implementation of the classification system. The profound results of GP implementations employing syntax trees inspired us to employ DTs as classifiers in our immune programming based classification system.

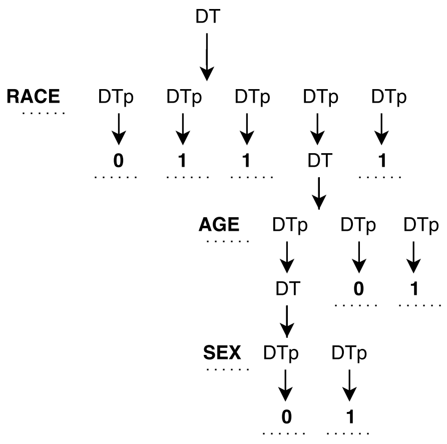


Figure 3.4: Derivation tree

3.1.4 Genotype to Phenotype Mapping

The genotype-to-phenotype mapping is similar to the one employed in GE and GIP with a small difference. Inspired by Hemberg's proposed method [62], a non-recursive version of the original grammar was used to inhibit the issue of non-terminals from never being derived (known as invalid generation). While mapping an *Ab* from its genotype to its phenotype, if the end of the genome is encountered and a non-terminal(s) is left to be derived, the genome is wrapped around and the process is repeated by loading the non-recursive grammar. The mapping process is depicted in Figure 3.5 and is explained in depth in Section 2.3.3. Note, the green boxes denote the non-terminal to be derived.

3.1.5 Generating the Population

A repertoire of *Abs* is randomly generated only once throughout the lifetime of the system. The repertoire is frequently updated by including offspring and by replacing unfit *Abs* with randomly generated *Abs*. This is carried out to increase the search space. Each individual in the population is a tuple of ⟨rank, genome, affinity⟩. The generation of genotype was explained in Section 3.1.2. Rank and Affinity, the fitness measures, are explained in Section 3.2.1. Initially, as the individuals are not yet evaluated, we assigned a poor fitness values (-100) to them.

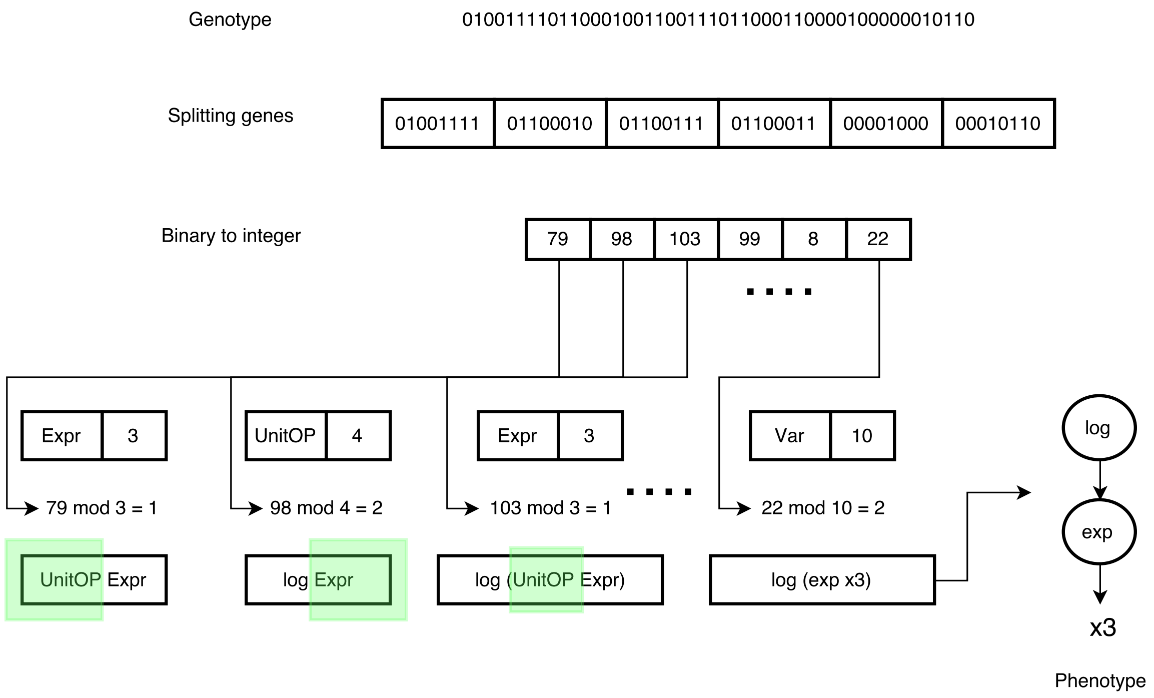


Figure 3.5: An example of genotype-to-phenotype mapping

3.2 Evaluation

The *Abs* generated or optimized have to be evaluated with respect to the training dataset. The training set comprises of *Ags*. The *Ags* streaming in through various sources are unlabelled and need to be buffered into an interface. Section 3.2 provides an overview of the methodologies undertaken to buffer the *Ags*, to evaluate the *Abs*, and to label the *Ags*. It also highlights the properties and the generation of the training datasets, and explains the testing procedure.

3.2.1 Fitness

Our classification system employs two measures of fitness *Rank* and *Affinity*. The latter is the primary measure of fitness which is used to train and evolve the candidate solutions. These measures are closely related to each other as *rank* is used to compute *affinity*. We evaluated the phenotype of the *Abs*.

Rank

Rank is the total number of correct classifications. This is the raw fitness of an *Ab* and is used to determine its affinity. Rank, also known as accuracy, is used as one of the test metrics to determine the models' performance.

Affinity

Affinity is the main measure of fitness used in CLONALG to evaluate and optimize the *Abs*. This measure of fitness is derived from the rank of an *Ab*. Equation 3.1 is used to calculate the affinity of *Ab*, where *rank* is the rank of an *Ab*, *min* is the lowest rank of the *Ab* in the repertoire, and *range* is $(max - min)$ where *max* is the highest rank of the *Ab* in the repertoire. If *range* is 0, affinity is set to 0.5.

$$affinity = \frac{rank - min}{range} \tag{3.1}$$

Affinity is in the range 0 to 1, implying that the best possible affinity to achieve is 1 and the worst is 0. Consider the following example: in iteration 1, the *min* is 20 and the *max* is 200 making the *range* 180. The *rank* of *Ab* is 200 and thus its *affinity* is 1. In iteration 2, the *min* is 1 and the *max* is 100 which makes the *range* 99. The *rank* is 100 and the *affinity* is 1. Although the raw fitness of these *Abs* are different, their corresponding *affinity* is 1 making these individuals the fittest. If *min* and *max* are both 200 (200 being the best), this means the population has converged. Hence, the affinity is set to 0.5. These phenomena aids in sustaining a diverse set of solutions. The affinities of the *Abs* change with respect to the current environment, which promotes adaptable evolution thereby promoting evolvability.

3.2.2 Streaming Interfaces

Streaming data can arrive from disparate sources. For example, monitoring a boiler in a power plant may involve processing continuous data arriving from disparate sensors. It is necessary to employ a control mechanism to pool the data from these sources into a buffer, called *sliding window* (SW), an interface between streaming data and the model [4, 6, 12, 21, 63, 64]. It is infeasible to store all the historical data as discussed before, hence employing a replacement policy to SW is inevitable. There are several replacement policies available such as first in-first out, least recently used, or not frequently used, but discussing all these policies are outside of the scope of this thesis.

First In-First Out (FIFO) is perhaps the cheapest replacement policy available as it follows a very simple rule: the least recent instances are replaced with the new instances. FIFO, in its pure form, is not usually employed for various operating system applications[65]. But in the case of streaming environments, the most recent data is of the utmost importance as it is the most relevant to the current scenario [66], making FIFO effectively employable to our work.

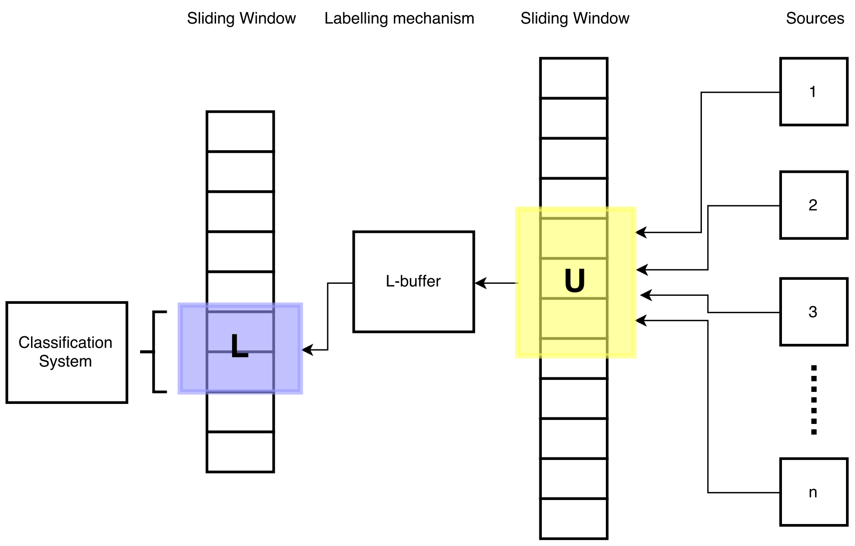


Figure 3.6: Streaming interfaces

Two sliding windows are employed, one for buffering unlabelled data (U) and one for storing labelled data (L) from which the classifiers learn. Interfaces U and L, as shown in the Figure 3.6, are slid whenever it is *time to slide* them. A labelling mechanism truly labels the data and buffers them in *Lbuffer* from where the instances are transferred to L. All the interfaces employ a FIFO replacement policy.

3.2.3 Labelling

We employed an Active Learning approach to label the data stream as it in infeasible to label all the incoming instances in real-time. Labelling was performed at regular intervals. Algorithms 1 and 2 depict the labelling procedure. If *time to label* is true, a *committee* of size c is created by selecting top c Abs from the repertoire. To make the system less computationally expensive, we randomly selected a subset (U_{temp}) of size r from U to participate in the labelling procedure.

Ags in U_{temp} are assigned an affinity, this process is shown in Algorithm 2. The *committee* temporarily labels the Ag by taking a majority vote of the labels predicted.

Labels are predicted as follows: Each genotype of the solution in the committee is first mapped to DTs using the genotype-to-phenotype mapping. After obtaining the DTs, the values of the data instance are plugged into the DTs one by one. Two datasets were used, synthetic and the adult dataset and their generation and properties are explained in the next subsection. In the case of the synthetic dataset, for example, if the DT is (+ (- x1 x2) (* x3 x4) (log x9)), the numeric values of the attributes of x1, x2, x3, x4, and x9 from the data instance are assigned to the attributes in the DT after which a result is obtained. The obtained result is compared with the thresholds obtained during the process of generation of the synthetic dataset and the respective label is assigned. In the case of the adult dataset, the label

```
    Data: Ab, a repertoire of Abs ;
    U, an unlabelled set of Ags ;
    r, length of the subset of U ;
    c, length of the committee ;
    bn, number of Ags to be permanently labelled ;
    Lbuffer, a set of permanently labelled Ags ;
    Result: bn labelled Ags
 1  committee ← select top c Abs from Ab ;
 2  $U_{temp}$ ← randomly select r Ags from U ;
 3  $U_{aff-temp}$ ← assign-affinity($U_{temp}$) ;
 4  $buf_{bn}$ ← select bn least certain Ags from $U_{temp}$ ;
 5  Lbuffer ← append $buf_{bn}$ ;
 6  U ← remove $buf_{bn}$ ;
```

Algorithm 1: Algorithm for permanently labelling the *Ags*

prediction process starts from the first node of the DT. We need to find the match of the values of the attributes of the data instance in the DT. Considering Figure 3.3 as the DT and *(middle State-gov low Bachelors 13 Never-married Adm-clerical Not-in-family White Male g2 g1 normal United-States)* as the data instance. When this data instance was plugged into the DT, the only match found was "white" and thus label "0" was predicted for this data instance by the DT in Figure 3.3. The entire process is repeated for all the *Abs* in the committee after which a majority vote was taken.

The *evaluation-set* is formed by appending the recently labelled *Ag* to L_{temp} which is the copy of the labelled set of instances. The affinity of each *Ab* in the repertoire is computed using the *evaluation-set* and an average is taken. The average affinity is then assigned to the *Ag*. This process repeats for all the *Ags* available in U_{temp}.

```
    Data: cmt, a committee of c best Abs ;
    $U_{temp}$, a randomly selected subset of U ;
    $L_{temp}$, the copy of labelled set of instances ;
    Ab, a repertoire of Abs ;
    Result: Label Ags from $U_{temp}$
 1  $i \leftarrow 0$ ;
 2  while $i < |U_{temp}|$ do
 3      Labels ← predict labels for $Ag_i$ using the cmt ;
 4      label ← select a label from Labels using a majority vote ;
 5      $Ag_i$ ← assign label ;
 6      evaluation-set ← $L_{temp} \cup Ag_i$ ;
 7      avg-aff ← calculate the average affinity of the Ab using the evaluation-set;
 8      $Ag_i$ ← assign avg-aff ;
 9      $i++$ ;
10  end
```

Algorithm 2: assign-affinity(U_{temp}):Algorithm for temporarily labelling the *Ags*

After assigning affinities to *Ags*, *bn Ags* with the lowest affinities are selected to permanently include them in *Lbuffer*. That is, the most uncertain *Ags* are permanently labelled so that the candidate solutions can learn from them. Changes can occur either on the boundary, near the boundary, or further away from the boundary of the search space. Labelling uncertain instances captures the changes occurring on or near the boundary of the search space. Another reason for randomly selecting a subset of U is to potentially increase the search space in order to capture both near and remote changes.

3.2.4 Datasets

The distribution of the classes in a dataset could have an unusual effect on a classifier's performance. A biased class distribution could improve the accuracy of the system but may fail to classify crucial data. Hence, we made use of two datasets for training and testing our system: a real-world dataset and synthetic dataset whose properties are well-known in various data mining (DM) and machine learning (ML) applications.

Real-world Data:

We employed the adult dataset for training and testing our system. It is freely available in the UCI Machine Learning repository [67][1]. This dataset has been widely used in various DM and ML applications and thus is a benchmark for evaluating our system with other classification systems. Numeric attributes were discretized and the missing values were regarded as noise and were thus discarded making the total number of the instances as 45,222. All the 14 attributes in this dataset are nominal, for example,
(middle Private medium 9th 5 Married-spouse-absent Other-service Not-in-family Black Female g1 g1 low Jamaica 1) and
(middle Self-emp-not-inc high HS-grad 9 Married-civ-spouse Exec-managerial Husband White Male g1 g1 normal United-States 0)
are the two data instances of the adult dataset where 1 and 0 represent the classes ≤50K and >50K respectively where 50K is a person's income per year. Each value in a data instance indicates a person's attributes' values. The distribution of the classes of this dataset can be found in Table 3.1.

classes	Adult	Synthetic (no CD)	Synthetic (CD)
class 1	75%	25.74%	3.3%
class 2	25%	67.84%	4%
class 3	-	6.42%	92.7%
no. of attributes	14	10	10

Table 3.1: Properties of the datasets used in training and testing. CD stands for concept drift.

[1]Interested readers may visit this website https://archive.ics.uci.edu/ml/datasets/Adult.

Synthetic Data:

For training and testing the proposed system, we need a large volume of ordered data whose temporal and behavioural properties are known. Synthetic data was generated using a similar process described in Atwater's thesis [68]. For the sake of clarity, we describe the process for generating synthetic data.

Equation 3.2 produces an instance of the streaming data where x is a vector of non-class attributes, vector a controls concept drift, and d is the size or dimension of vectors x and a. The vectors, x and a are initialized randomly (a at the beginning) in the range $[0, 1]$. The value generated by this equation is compared with the threshold value(s) to assign a label. The threshold value(s) is generated by taking a sum of randomly initialized vector with d dimensions. We wanted to generate three labels (0,1, and 2) and thus we generated two threshold values randomly. The first threshold value was generated by taking the sum of the vector a and the second threshold value was generated by taking the sum of a randomly generated vector of dimension d. We multiplied the two threshold values by the factors of 1/2 and 2/3 for evenly distributing our concepts (or three classes.) Examples of the data instances are:
(0.8501184873340292 0.5571873406635055 0.515485903532493 0.3137504903739556 0.6820560921606765 0.3306239589047114 0.19985539549261386 0.5939804137563162 0.5416601765121606 0.7627175673016473 0) and
(0.29730702839788564 0.9051495716606992 0.26345939557057674 0.6136015217353397 0.38147359791828983 0.30094685442674574 0.08803661943218133 0.4086165917087931 0.42523013135601473 0.09542764277405798 1)
where 0 and 1 are the labels.

$$\sum_{i=1}^{d} x_i * a_i = a_o \tag{3.2}$$

Concept drift can cause two instances of the data with the same attribute values to have different classes. The process of simulating concept drift (CD) is shown in Equation 3.3: where p is the number of attributes in a involved in causing change, t controls the magnitude of change, and s controls the direction of change which is randomly initialized to $\{-1, +1\}^d$. The value a_i is adjusted after every generation. After n instances, p attributes involved in change, change their direction using $h\%$. CD is introduced at every n instances. Inspired from the minimum-variance method of labelling introduced in [8] and Atwater's thesis [68], The parameters involved in simulating CD were initialized to the values listed in Table 3.2. A total of 2,000,000 instances of the synthetic dataset were generated using Equation 3.2.

$$a_i = a_i + s_i * \frac{a_i * t}{n} \tag{3.3}$$

The properties of this dataset can be found in Table 3.1.

parameters	values
d	10
p	5
n	1000
t	0.1
h	0.2

Table 3.2: Values of the parameters involved in concept drift.

3.3 Optimization

The genotypes of the *Abs* are optimized after the process of affinity calculation, labelling, and testing (only when their *time to slide*s are true). The evolve process in Figure 3.1 is shown in Algorithm 3. After calculating the affinity of the *Abs*, n best *Abs* are selected for *cloning*. The cloned *Abs* undergo *hypermutation*. The mutated clones are added to the repertoire of *Abs* after computing their affinities. We then replace d worst *Abs* from the *reselected* set with d randomly generated *Abs*. This is performed to potentially increase the search space of the solutions. The *selection*, *cloning*, and *hypermutation* methods are explained in the following subsections.

```
  Data: $Ab$, a repertoire of $Abs$ ;
  $n$, a parameter for selecting $Abs$ for genetic variation ;
  $d$, a parameter for purging and randomly generating $Abs$ ;
  Result: evolved $Abs$
1 $Ab \leftarrow$ calculate affinity ;
2 $Ab_n \leftarrow$ select $n$ best $Ab$ ;
3 $Ab_{nc} \leftarrow$ clone $Ab_n$ ;
4 $Ab^*_{nc} \leftarrow$ hypermutate $Ab_{nc}$ ;
5 $Ab^*_{nc} \leftarrow$ calculate affinity ;
6 $Ab_{reselect} \leftarrow Ab \cup Ab^*_{nc}$ ;
7 $Ab^*_{reselect} \leftarrow$ remove $d$ worst $Abs$ ;
8 $Ab_d \leftarrow$ randomly generate $d$ $Abs$ and calculate affinity ;
9 $Ab \leftarrow Ab^*_{reselect} \cup Ab_d$ ;
```

Algorithm 3: Algorithm for Evolving *Abs*.

3.3.1 Selection

The best or worst *Abs* are selected using the measure of affinity. In the case of *selection*, we selected n best *Abs* and d *Abs* in the case of *reselection*.

3.3.2 Cloning

Cloning is one of the asexual operators employed by GIP. The selection method selects n best *Abs* and makes replicas of them called as clones. CLONALG, traditionally, employs a cloning rate which is directly proportional to a selected *Ab*'s affinity. That is, higher the affinity, more clones are generated. Our system employs a cloning rate (*beta*), see Equation 3.4, which is based on the selection value n and the population size, s:

$$beta = \frac{1}{n} * s \tag{3.4}$$

3.3.3 Hypermutation

Cloned *Abs* undergo hypermutation. Each bit in the genome is flipped using a probabilistic factor. Clones with higher affinities have a lower chance of being mutated, since we do not want to introduce potential disruptions in the most fit clones. On the other hand, clones with lower affinities have a higher chance of being mutated. The *rate* at which the clones are mutated is computed using Equation 4.4, where ρ is the mutation factor.

$$rate = \exp[(-\rho) * affinity] \tag{3.5}$$

Chapter 4

Empirical Analysis

In this work, we proposed and implemented three variants of a GIP-based classification system: a static system, and two streaming systems for labelled and unlabelled data respectively. Two datasets were utilized for training and testing purposes: Synthetic and Adult datasets (see Section 3.2.4). For the static system, test was performed only once that is after the training. In the case of streaming systems, as the data streams in continuously, it is not possible to test the models' performance after the entire training process. So, tests were performed at regular intervals midst training. In all the cases, the systems were analysed using the same test metrics. The results of the first two variants are compared with the results of Atwater et al works presented in [6, 7]. The last variant, which performs labelling, is compared with Zhu et al work presented in [8]. The systems under comparison are called benchmark problems.

4.1 Test Metrics

Four metrics were used to determine classification performance.

1. **Accuracy:** It is the total number of correct classifications, also known the rank (see Section 3.2.1). In the case of static and labelled-data streaming systems, the accuracy of the best *Ab* is computed. For the unlabelled-data system, a committee was formed which are the top *c Abs* and its average accuracy was computed. In both cases, the best *Ab*(*s*) was determined using the measure of affinity and then its rank is computed.

2. **Average Detection Rate:** Average Detection Rate (DR) is another test metric used while performing tests to evaluate the models' performance. DR is usually used in various streaming problems (such as [68]) as it aids in maintaining the accuracy in the case of class imbalance. The average DR of each *Ab* is calculated using Equation 4.1.

$$Avgerage\, DR = \frac{\sum_{i=1}^{C} DR_{C_i}(Ab)}{|C|} \tag{4.1}$$

Where, C is the class-set consisting of labels. DR_{c_i} is the rank of *Ab* for class C_i divided by the total number of instances of C_i. The average DR of each *Ab* is calculated and the best one was recorded.

3. **Average Accuracy:** Diversity is a by-product of evolvability [2]. Thus, evolvability can be determined by computing a repertoire's diversity. To determine the diversity of the repertoire of *Abs*, we compute its average accuracy. This is an implicit way of measuring diversity [4]. Larger average accuracy implies more diversity. We computed the diversity of the repertoire using Equation 4.2 where s is the size of the repertoire of *Abs*.

$$Average\,Accuracy = \frac{\sum_{i=1}^{s} Rank(Ab_i)}{|repertoire|} \tag{4.2}$$

4. **Variance:** Variance is another metric which was used to measure the diversity of the repertoire of *Ab*. Variance was the only metric used to measure the diversity of *Ab* using training data. Thus, these statistical measures depicted how the diversity of the trained models evolve over the period of t generations. It was used to determine how far the accuracies of the *Abs* were spread out from its average accuracy. The measure of variance was used to determine the diversity of the repertoire on a genotypic and phenotypic level.

   (a) *Phenotypic Diversity:* Phenotypic diversity determines the diversity of the phenotypes of the solutions. An average of the variance was computed using Equation 4.3.

$$Variance = \frac{\sum_{i=1}^{s}(Rank(Ab_i) - Average\,Accuracy)^2}{|repertoire|} \tag{4.3}$$

   (b) *Genotypic Diversity:* One of the aims of this thesis was to achieve evolvability in order to sustain diversity. The GIP system generates and evolves a repertoire of genotypes, so it is of our main interest to analyse the diversity of the genotypes. Genotypic diversity was calculated by taking an average of the variances of all the integers of the genomes at locus i using Equation 4.4, where L=240 is the number of locus, N=120 is the the size of the repertoire, X_i is the set of integers at locus i, and μ is the mean of X_i.

$$mean\,of\,variances[69] = \frac{\sum_{i=1}^{L} \frac{\sum_{i=1}^{N}(X_i - \mu)^2}{N}}{L} \tag{4.4}$$

   The highest possible diversity for the genotypes to achieve is 0.25, and thus the values closer to 0.25 imply that a high diversity was attained [69].

4.2 Static System

Until now, there has not been any research which employs GIP for classification problems. In this section we present a GIP variant we developed to automatically generate and evolve classifiers for static data.

4.2.1 Parameterization

1. **Number of generations, t:** This defines the total number of iterations used to train the system. This parameter should not be initialized to a low value as the solutions may not have achieved a converged state. Similarly, it should not be initialized to a very high value in order to make the system less computationally expensive. No terminating condition was defined for our system. Thus, the training is conducted for all the generations. We swept the parameter space, doing runs with a variety of number of iterations to find the best value which was **1000**.

2. **Population Size, s:** This parameter controls the size of the solution and search spaces. The larger the size of the population, the better the chances of finding an optimal solution faster, i.e., attaining a state of convergence. But a large population size makes the system computationally expensive. Thus, a trade-off needs to be attained between convergence and computational expense. We initialized s to **120**, a parameter inspired from the benchmark problem [68] for the purpose of comparison.

3. **Total number of genes (ng) and gene length (gl):** An *Ab*'s genome size is the factor of the total number of genes and its length. Each bit in the genome could potentially be mutated (based on a probability). A larger genome size could potentially introduce more mutations which could lead to better *Abs*. The gene length is used in genotype-to-phenotype mappings where the gene-length specifies the number of bits to use while converting the binary bits to its decimal value. Total number of genes was initialized to **30** and the gene length was initialized to **8**.

4. **Number of *Abs* to select, n:** The system selects n *Abs* with the highest affinities for optimizing them. Only n selected *Abs* undergo optimization. Similar to s, a trade-off is required between optimization and computational expense. Introducing optimizations to more *Abs* could promote evolvability. Only 33.33% of the population undergoes optimization, that is, n was initialized to **40**. Varying only 33.33% of the population was sufficient to achieve good performance.

5. **Cloning rate, β:** β is defined as $\beta = \frac{1}{n} * s$. This parameter determines the number of clones to be generated for each selected *Ab*. Thus, for each selected *Ab*, β which is $\frac{1}{40} * 120$, **3** clones are generated.

6. **Mutation rate, ρ:** Each bit in the genome is flipped based on the probabilistic factor of ρ. A high mutation rate must be initialized to sustain diversity. The implemented system negates this value, thus the value 1 for ρ is the highest possible mutation value. Too much mutation could exploit the genomes. Therefore, this parameter was initialized to **3.5** based on trial and error.

7. **Purging rate, d:** This parameter determines the percentage of the population to be replaced with the same amount of randomly generated *Abs*. The parameter, d should not be very high or very low. A high d could remove evovlable *Abs* and very low d could slow down the evolution process. This parameter was initialized to **10**%.

8. **Grammar, g:** A unique grammar was defined for each dataset, adult and synthetic. The same grammar defined for the static system was used for the streaming systems. Figure 3.2 depicts the grammar defined for the synthetic dataset. Grammar for the adult dataset can be found in Appendix A.

9. **Number of runs, r:** A run consists of t generations. We analysed the test data of the system over **30** runs. An average of the test metrics over these 30 runs was taken.

10. **Size of training data:** To inhibit the issues of over fitting and under fitting, and to reduce the cost of computation, static datasets of size 6000 were used, out of which **80%** was used for training the static classification system. Thus, the size of the training dataset was initialized to 5000.

11. **Size of testing data: 20%** of the total dataset (6000 cases) was utilized to test the system, that is 1000 test cases were used.

    The properties of the datasets are explained in Sub-section 3.2.4 of Chapter 3.

4.2.2 Results and Discussion

The results of the static classification system using the Adult and Synthetic datasets are shown in Table 4.1. Concept Drift (CD) is not relevant to static datasets as it is a property of changing environments.

Metrics	Adult	Synthetic
Accuracy%	77.9% (779)	78.04% (780)
Average Accuracy	761.82	691.43
Average Detection Rate	0.66	0.58
Genotypic Diversity	0.23	0.09
Phenotypic Diversity	129984.9	959307.99

Table 4.1: Test results of the static classification system. The value in parenthesis shows the number of hits

In spite of having only two classes, the GIP system was unsuccessful to achieve a satisfactory performance using the static adult dataset. The average DR was 20% lower than a *bat* case of the benchmark problem presented by Atwater et al in [6], indicating the inability of this variant to classify most of the instances of a class. Training the system with all the training cases of the adult dataset proved very computationally expensive. Therefore, "under-fitting" could be one of the reasons behind the low average DR, as only 16.5% of the total training cases were used to train the system. While testing, the system maintained a small amount of diverse solutions in the repertoire, as the average accuracy was 2% lower than the total number of hits[1]. The high values of genotypic and phenotypic diversities (as

[1]The average value of a dataset can not be higher than a value of an instance in that dataset. Equal average and maximum values indicate that all the instances in the dataset are of the same values. Unequal values indicate that some instances are of different values implying that the dataset consists of different instances. Hence, this measure of diversity is known as an "implied" diversity measure.

shown in Table 4.1) also imply that the classifiers evolved are in fact diverse, as the value 0 for variance indicates the same fitness of the classifiers.

The performance achieved by the ternary-class synthetic dataset was somewhat better than the binary-class adult dataset. More classes imply a more difficult scenario for the system to learn from. The contrasting accuracy and DR make the system satisfactory and unsatisfactory at the same time. The low DR value, again, indicates the necessity to train the system using more training cases. The solutions retained in the repertoire were more diverse than that of the adult dataset, as the average accuracy was 12% higher than the average number of hits. This is also indicated by the phenotypic diversity where the variance of the synthetic dataset was 638% higher than the phenotypic diversity of the adult dataset. In spite of achieving a very high phenotypic diversity, the genotypic diversity of the synthetic dataset was 60% lower than that of the adult dataset.

4.3 Labelled Streaming System

In the case of streaming problems, periodic tests were performed every 200 generations for the synthetic dataset and (due to the scarcity of test cases) every 500 generations for the adult dataset. Typically, in streaming problems, tests are performed on the cases available in the window. After evolving the candidate solutions, the window was slid and if the *time to test* were true, a test was performed using the cases in the window.

4.3.1 Parameterization

The values of the parameters used in the static system produced satisfactory results in the case of achieving diversity. Recall that one of our hypotheses requires the streaming system to sustain diversity in order to address concept drift. Therefore, we initialized the parameters to the same values as used in the static system with the addition of the following parameters that supported the dynamic properties of the data stream. The properties of the datasets used for training and testing are stated and explained in Section 3.2.4.

1. **Number of generations, t:** The total number of generations for the GIP system trained using synthetic dataset with concept drift was **30000**. Due to a small number of instances (45,222), t was initialized to **750** for the GIP system trained using the adult dataset.

2. **Size of the window, w:** The size of the window was initialized to **200** to make the training process faster.

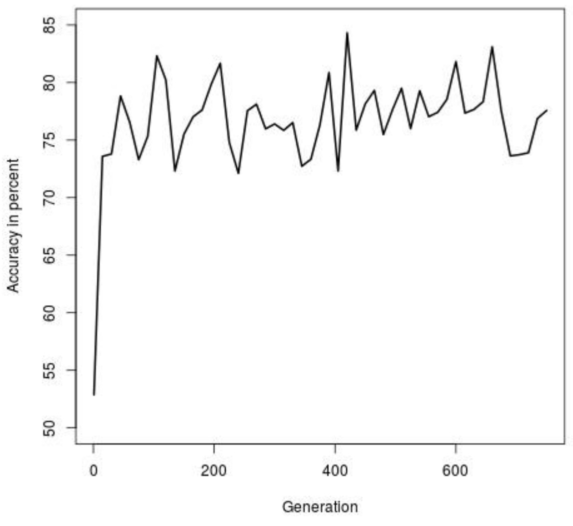
3. **Sliding parameter, sp:** The sliding parameter was initialized to **60**, similar to the benchmark problem.

4. **Time to slide, tts:** For our system, we assumed that the streaming data flows in rapidly at regular intervals. Thus, our system accepted data at every generation. After accepting 200 instances at generation 1, the sliding window accepted sp new instances at every generation. This is done in

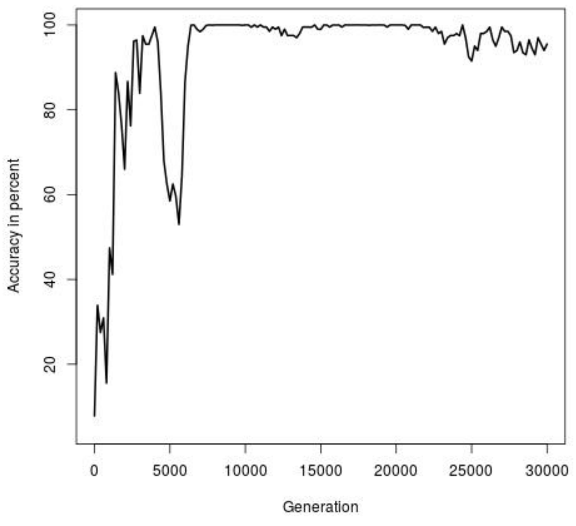
lieu of accepting w new instances to avoid drastic environmental changes for the system to adjust efficiently to the small changes.

4.3.2 Results and Discussion

At generation 1, tests were performed on the untrained candidate solutions with the default affinity of -100. Thus, the first randomly generated Ab was selected to test the Ags in the sliding window. Therefore, generally for the all streaming experiments (labelled and unlabelled), a steep growth or drop was observed after generation 1. Recall that in synthetic dataset, concept drift occurred every 1000 generations.

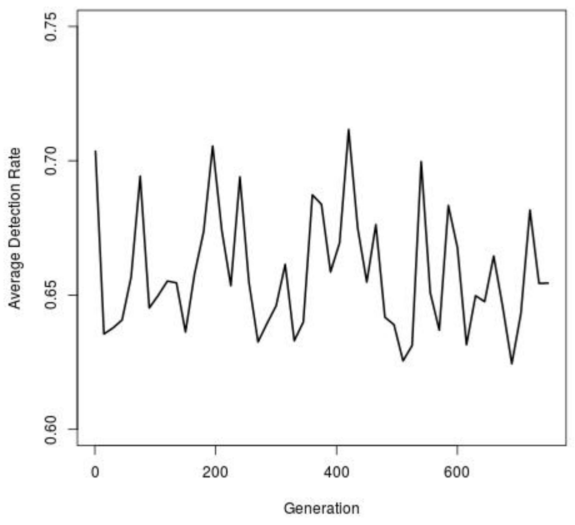


(a) Adult dataset

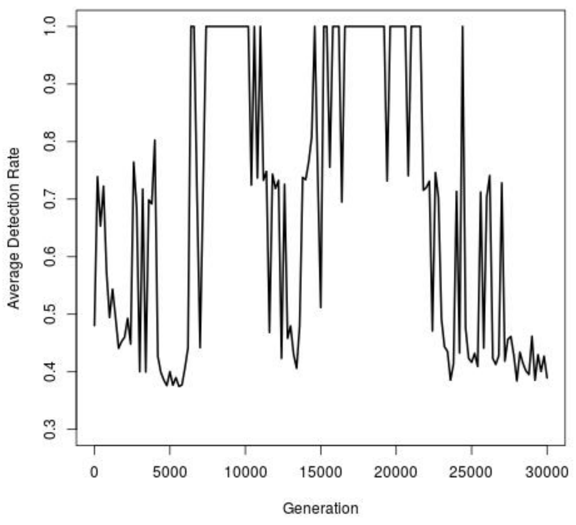


(b) Synthetic dataset with CD

Figure 4.1: Labelled streaming system: Accuracy% of the best Ab at each generation during *training*.



(a) Adult dataset



(b) Synthetic dataset with CD

Figure 4.2: Labelled streaming system: Average Detection Rate during *training*.

Accuracy

The streaming system trained using the labelled adult dataset achieved an average accuracy of approximately 77% which is similar to its static counterpart. The performance of the best *Ab* varied a lot at every generation which depicts environmental changes. As mentioned in Chapter 1, the data available in the window may not be representative which explains the variation in the accuracy. Also, if the repertoire maintained diverse solutions, the best *Ab* could be different at every generation which explains the variation in performance. After generation 400, the graph (mostly) follows an upward trend until generation 690 after which it experiences a steep drop (see Figure 4.1a).

The average accuracy achieved by the system trained using synthetic dataset with concept drift was approximately 93% (see Figure 4.1b), which was 19% better than its static counterpart. Unlike the best *Ab* in the adult dataset which never performs equivalently, the accuracy in Figure 4.1b appears to be stable after generation 7,000. One of the reasons of this could be the convergence of the solutions in the repertoire because of which the same *Ab* was selected to test the data. Convergence in streaming problems is not desirable, unless the data never drifts which is impractical. The graph of the synthetic dataset followed two major growths and only one major drop. The fitness started increasing after generation 2,000 with a steep drop at generation 5,000, after which a stability in the fitness was achieved. Minute variations after generation 24,800 are observed which displays the ability of the GIP system to introduce variation in the candidate solutions in accordance to its environment. A very high accuracy ensures that the system was able to handle concept drift well. The system which handles concept drift well while maintaining satisfactory performance will also perform well without the presence of concept drift and thus we did not the test the labelled streaming system using synthetic dataset without concept drift.

Average Detection Rate

Similar to the static system, the mean of average DR for the adult dataset was also 0.66, see Figure 4.2a. A high average DR at generation 1 displays the power of randomness. However, randomness does not always guarantee satisfactory results. No patterns were found on the graph of Figure 4.2a as the variations seem quite random, making it almost impossible to comprehend environmental changes from it. The average DR of the *w01P* case[2] of the benchmark problem introduced by Atwater et al in his work [6] varied from (approximately) 45% to 68% with distinct fluctuations. On the other hand, our average DR varied from 65% to 71% implying that in spite of having a low sliding window size and less number of generations, our streaming system achieved a better performance. The mean of the average DR for the synthetic dataset was approximately 0.70 achieving a 20% increase from the static system. Contrasting average DRs were achieved at different generations maintaining irregular sharp increases and drops. One of the reasons behind obtaining poor average DR could be the absence of one or more classes of the test cases available in the window. The absence of one class affects the average DR by 33.33%. Both, the *planar* dataset[3] of the benchmark problem and our synthetic dataset were produced through the same process. The average DR of the planar dataset of [7] varied from approximately 30% to 90% whereas the average DR of our synthetic dataset (as shown in Figure 4.2b) varied from 30% to 100% confirming that the GIP system performed better in the dynamic environment than the GP system under comparison while using a very small window size of 200.

Diversity

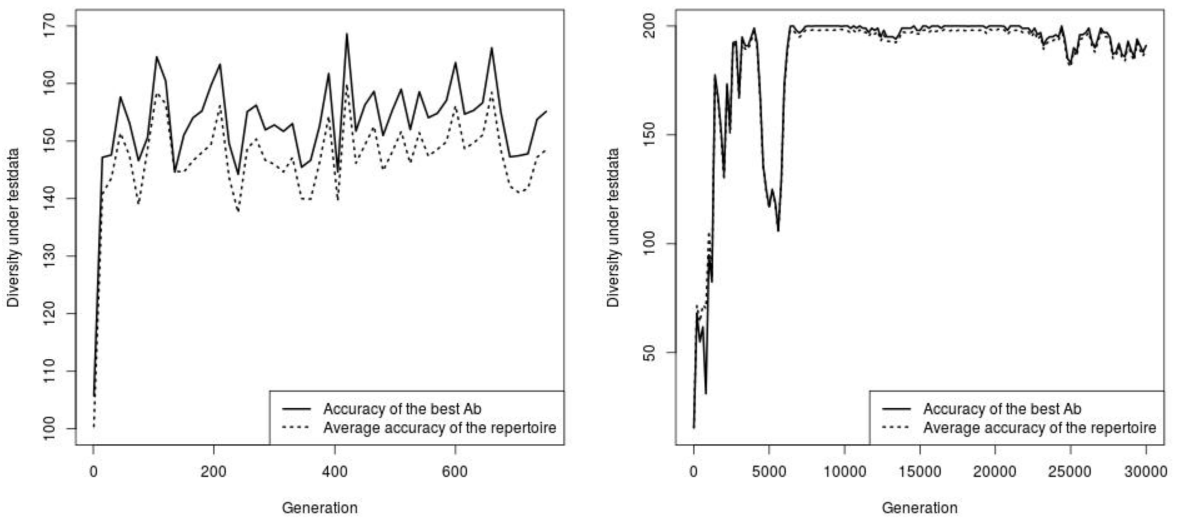
As noted earlier, ensuring diversity while maintaining accuracy is essential for streaming problems. We computed the average accuracy of all the candidate solutions in the repertoire to determine its diversity implicitly with respect to the test cases available in the window. Figure 4.3 depicts the diversity of the two datasets where the solid line represents the number of true positives of the best *Ab* and the dotted line represents the average number of true positives of 120 *Abs*. Similar to the static system, the streaming system for the adult dataset was able to preserve more diversity than the streaming system for the synthetic dataset. At generation 135, the best and the average accuracy as shown in Figure 4.3a was the same implying that all the solutions of the repertoire were the same or performed similarly. But the solutions after generation 135 were diverse and a steady diversity was maintained after generation 400. On the other hand, synthetic dataset (see Figure 4.3b) managed to reach an almost-converged state after generation 2000. A very small amount of diversity was maintained by the system which was enough to achieve a good performance. However, the amount of diversity preserved may not be sufficient for the system to perform well in the long run. Despite the convergence for a short period of time, from generations 4,800 to 5,600, the system was able to introduce a small amount of variations after generation

[2]The *w01P* case of Atwater et al works [6] employed a sliding window of size 1% of the dataset. Thus for the adult dataset, the size of the sliding window was 480 and the number of generations was initialized to 30,000.

[3]The size for the planar dataset also used in [7] was initialized to 7500.

5,600. An interesting pattern was observed from generations 200 to 1200, where the average accuracy of the repertoire is better than the accuracy of the best *Ab*. This is the result of determining the best solution using the measure of affinity and then plot its rank. Although, the affinities of the best solutions generated until 1200 were high, their ranks were not the best. Over the time, the GIP system learned to link affinity with rank and the best affinity *Ab* always had the best rank.

Measuring the diversity using the training dataset provides more information about evolvability or survivability. We computed the genotypic and phenotypic diversities of the repertoire under *training* dataset using the measure of variance. Figures 4.4 and 4.5 depict the genotypic and phenotypic diversities of the adult dataset and the synthetic dataset respectively. As shown in Figure 4.4, the system managed to maintain a stable genotypic diversity after generation 150. In spite of achieving a stable genotypic diversity, we achieved a gradual increase in the phenotypic diversity which was what we aimed for. The system maintained a diverse repertoire of genotypes throughout its life time. The lack of convergence for both genotypic and phenotypic variances represent the absence of the most optimal solution, thus ensuring survivability. Recall that in Chapter 2, we pointed out why we need to pursue survivability rather than the most optimal solution. Thus, the solutions of the adult dataset were capable of achieving evolvability.



(a) Adult dataset (b) Synthetic dataset with CD

Figure 4.3: Labelled streaming system: Measuring Diversity using test cases. The dotted line represents the average accuracy of the population and the solid line represents the number of true positives of the best *Ab*.

The graph in Figure 4.5 depicts the diversity achieved by the candidate solutions of the synthetic dataset during training. This is the only experiment where the genomes are the most diverse and the phenotypes reached an almost converged state after generation 5000. Neutral mutations can cause the system to behave in such a way. In spite of maintaining a very diverse set of genomes, neutral mutations always select the same production rule to generate its corresponding phenotype. The most interesting part is the spikes around generation 5000 observed in Figures 4.1a and 4.5 respectively. A reduced accuracy around generation 5000 caused a sudden increase in the phenotypic diversity. The genotypic diversity had no effect with the reduced performance. An increase in the genotypic diversity around generation 14,000 minutely increased the phenotypic diversity. In spite of the reduced genotypic diversity at generation 25,000, the system maintained a stable phenotypic diversity. Overall, the labelled streaming system for synthetic dataset did not quite achieve a satisfactory phenotypic diversity and diversity under test data. This is not desired for streaming systems, as a converged solution may not incorporate environmental changes which could affect the system's performance.

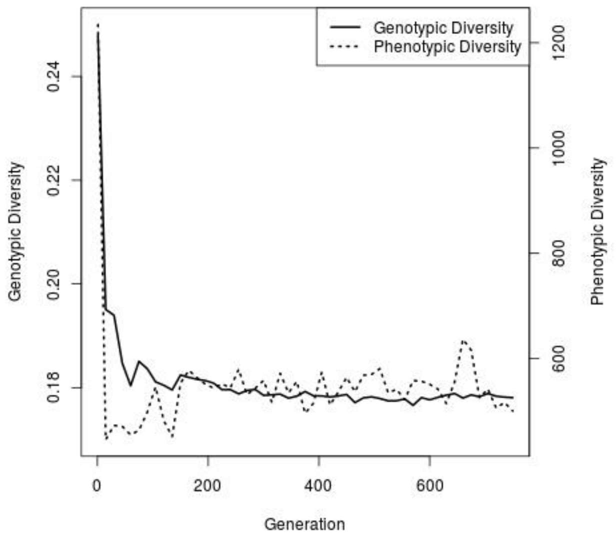


Figure 4.4: Labelled streaming system: Measuring genotypic versus phenotypic diversity for adult dataset during *training*.

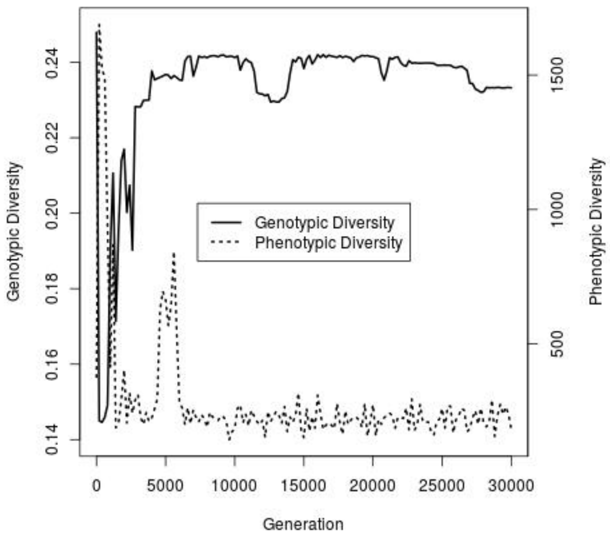


Figure 4.5: Labelled streaming system: Measuring genotypic versus phenotypic diversity for synthetic dataset with concept drift *training*.

4.4 Unlabelled Streaming System

Analogous to the labelled streaming system, testing was performed every 200 generations for the Synthetic dataset and every 500 generations for the Adult dataset. It is impractical to use unlabelled data to perform tests. Therefore, we made use of another data stream of the labelled instances just for the testing purpose. No test exemplars were used to train the system.

4.4.1 Parameterization

All the GIP parameters were initialized to the same values as the ones used in the static and the labelled streaming systems. But as the unlabelled streaming system contains more streaming interfaces than the labelled streaming system, the streaming and labelling parameters were initialized to the following values (Note that some experiments used different values which will be mentioned while discussing their results):

1. **Size of the unlabelled window, $|U|$:** The system labels the data available in U. A large U will potentially increase the chances of having diverse instances making U representative. Permanently labelling instances while maintaining a class balance is desirable as the models learned using a dominant-class instances will affect the performance. But more data instances increase the cost of labelling. Thus, this value was initialized to **1000**, as a trade-off between performance and cost.

2. **Size of the labelled window, $|L|$:** The system learns from the instances available in L. As labelling is not performed every generation, more data should be used for training the system. The size of L for synthetic dataset was initialized to **500** and to **1000** for the adult dataset. A larger size is preferred for the adult dataset because labelling is performed even later than the synthetic dataset.

3. **Time to slide U, ttU:** Similar to the labelled streaming system, the window U, slid every **1** generation for the synthetic dataset and every **100** for the adult dataset as it does not have enough training exemplars.

4. **Sliding parameter, sp:** Analogous to the labelled streaming system, this parameter was initialized to **60**. Both the interfaces, U and L, are slid by sp. In the case of U, 60 old instances are replaced by 60 new instances from the raw data at every ttU generation. Whereas, for L, 60 instances from the *Lbuffer* are transferred to L by replacing 60 old instances at every ttL generation.

5. **Number of instances to randomly select from U, r:** After trial and error, we learned that labelling all the instances available in U is very expensive. Thus, we randomly select **500** instances from 1000 instances for labelling.

6. **Size of the committee, sc:** More members in the committee can increase the labelling accuracy thereby increasing the cost of labelling. Therefore, we initialized sc to **10**.

7. **Time to label, lab:** Labelling was the most expensive task of the unlabelled streaming system. It was proved infeasible to perform true labelling or temporary labelling for detecting concept drift at every generation. Thus, labelling was performed periodically at every **500** generations for the synthetic dataset and every **1000** generation for the adult dataset due to the limited number of instances in the adult dataset.

8. **Number of instances to truly label from U_{temp}, bn:** The committee at generation t, may not accurately predict labels for a data instance. Therefore, instead of truly labelling all the instances in U_{temp}, we truly label only **50** instances.

9. **Time to slide L, ttL:** Since labelling was performed at every 500 generations, L like U, cannot be slid at every generation. Thus, L was slid as soon as we performed labelling. We initialized ttL to **501** for the synthetic dataset and **1001** for the adult dataset.

4.4.2 Results and Discussion

The unlabelled streaming system forms the core of this work. Therefore, we performed multiple experiments to evaluate the streaming system. The first experiment was performed using the adult dataset. The properties of the synthetic dataset can be easily varied using various control parameters. Hence, we tested the unlabelled streaming system using synthetic dataset with and without concept drift, known as second and fourth experiments respectively. We also tested the system by randomly initializing 20%

of the unlabelled instances at the beginning instead of using a labelled set known which was our third experiment.

These experiments are not compared with the benchmark problem or any other problems, as we have not come across any GP, GE, IP, or GIP system that evolves solutions while simultaneously labelling data for streaming problems.

Accuracy

The same testing data was used to test the labelled and the unlabelled streaming systems. The average accuracy of the committee of the adult dataset (as shown in Figure 4.6a) was approximately 80%, which is 3.8% better than the labelled streaming system. The similar accuracies of the static, labelled, and the unlabelled systems prove that the labelling process was accurate. Unlike the labelled streaming system which learned from the new data at every generation, the unlabelled streaming system learned from the new data every 1001 generations and achieved slightly better accuracy. An interesting trend was observed every 10,000 generation, the accuracy seems to gradually grow after a sudden drop. The accuracy never converged which was essential to achieve evolvability which in turn aided in sustaining diversity. The accuracy of Zhu et al's system [8] under comparison which employed C4.5 algorithm varied from 81 to 84% making the average 82.5% which was 3% better than our accuracy making our performance a tad unsatisfactory.

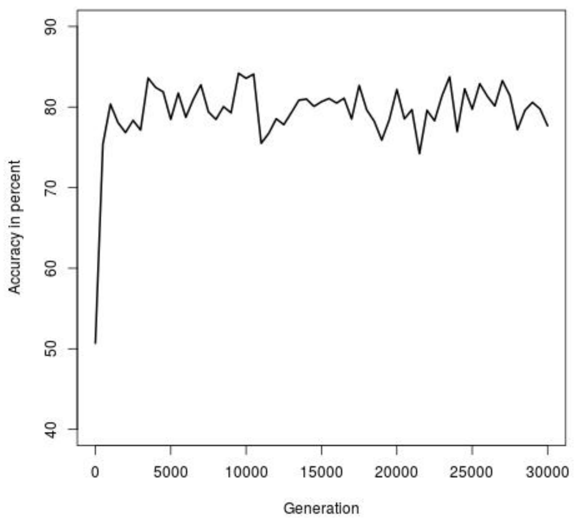
The second experiment achieved an average accuracy (of the committee) of 91%. As shown in Figure 4.6b, the accuracy of the committee across 30,000 generations remained moderately stable. No noticeable trends were found, making it very difficult to discern environmental changes. The committee of solutions seems to be in an almost-converged state, thus it seems that low evolvability was achieved. Figure 4.6c depicts (the third experiment) the same synthetic dataset where no off the shelf labelled data was used at the beginning. Instead, 20% of the data instances in U were labelled randomly using probabilistic factors. The probability of occurrence of each label was determined by analysing past labelled instances. Each data instance was thus labelled using its probabilistic factor only once at the beginning. We also experimented by truly labelling 100 instances instead of 50, and labelling every 200 generations instead of every 500. Coming back to Figure 4.6c, the average accuracy achieved was 89% which is similar to Figure 4.6b. Both the methods achieved similar performances which displays the power of randomness.

The fourth experiment, labelling synthetic data with concept drift, is perhaps the most important experiment of this work. The average accuracy achieved by the fourth experiment (as shown in Figure 4.6d) was approximately 86%, which is 9% less than its labelled counterpart. The accuracy of our system was 9% better than the benchmark problem which employed C4.5 and 32% better than the benchmark problem which employed Naive Bayes (NB) [8]. Thus, our evolutionary system was more capable of addressing concept drift than its non-evolutionary counterpart. The accuracy of the committee remains stable until generation 10,000 which started declining thereafter. The accuracy seems to increase around generation 15,000 but it continues to decrease after that. The accuracy seems to rise around generation 30,000 but it is unlikely that new genetic code will be introduced in the population. Recall that, among the datasets used for testing, synthetic dataset for the labelled streaming system was known to maintain low diversity. Thus, if the population was converged around generation 10,000 (similar to Figure 4.1b), it is highly unlikely for the system to re-introduce new genetic code or diversity. One of the reasons behind a linear decrease in the accuracy could be the fact that two separate windows were used for training and testing purposes. As concept drift was introduced every 10,000 instances, testing data may have had different changes to it as opposed to the training dataset. Thus, it is plausible for the performance to decrease gradually if the system were trained and tested using different datasets with different concept drifts. Also, the system was trained using new labelled data every 501 generations and the system was tested every 200 generations. 60 new unlabelled instances are accepted at every generation. Thus, 300 generations will accept 18,000 new unlabelled instances which includes a new concept drift as changes were introduced to the data at every 1,000 instances. The duration before testing the data is enough to introduce concept drift which was not taken into account by the system which thereby decreases the performance of the models. The labelled streaming system was able to handle concept drift because it was trained with new data at every generation. Training the unlabelled system with new data at every generation is infeasible and impractical because of the cost of labelling.

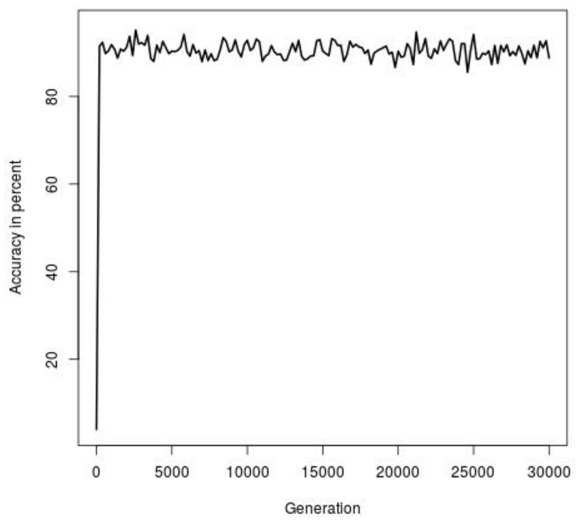
Average Detection Rate

Analogous to the labelled streaming system, the mean of average DR in Figure 4.7a of the adult dataset was approximately 0.68. Similar accuracy and average DR prove that the labelling process was effective. As the models of the unlabelled and labelled streaming systems were trained with new data at different generations, dissimilar trends (see Figures 4.7a and 4.2a) in their average DR were obtained. Also, the models in the unlabelled streaming system were trained for 30,000 generations unlike the labelled system where the models were trained only for 750 generations.

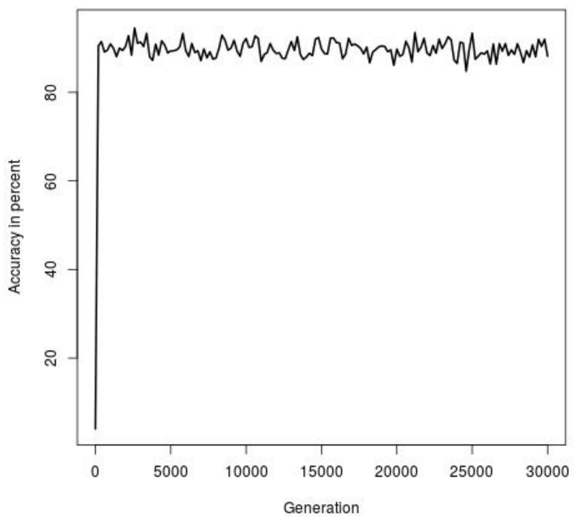
The mean of average DR, 0.42, of the second experiment was highly unsatisfactory. The steep drop after generation 1, as depicted in the Figure 4.7b, depicts that the best solution which was generated randomly in fact had a better average DR than the solutions which underwent learning. The average DR depends highly on the distribution of the class labels of the testing exemplars. Recall that each label contributes 33.33% of the average DR value. In the case of synthetic dataset without CD generated for the unlabelled system, class 1 dominated the exemplars. Thus, if the data stream consisted small or no amount of classes 2 and 3 or if the system failed to classify the instances of classes 2 and 3, the value of the average DR will affect a drop of 66.66% which was experienced by our system. The fact that the mean of the average DR of the third experiment as shown in Figure 4.7c was also 0.41 proves that the unlabelled system can perform equally without using past labelled instances. The trends achieved by the average DR in the Figures 4.7b and 4.7c were quite dissimilar, as the amount of the true labels requested and the generations at which labelling was performed were different.



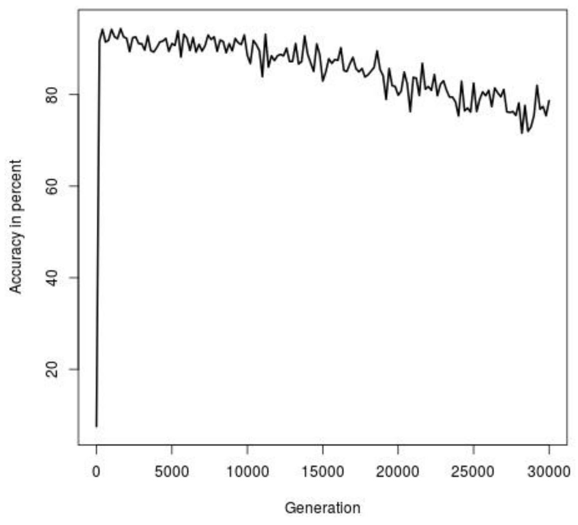
(a) Adult dataset



(b) Synthetic dataset without CD

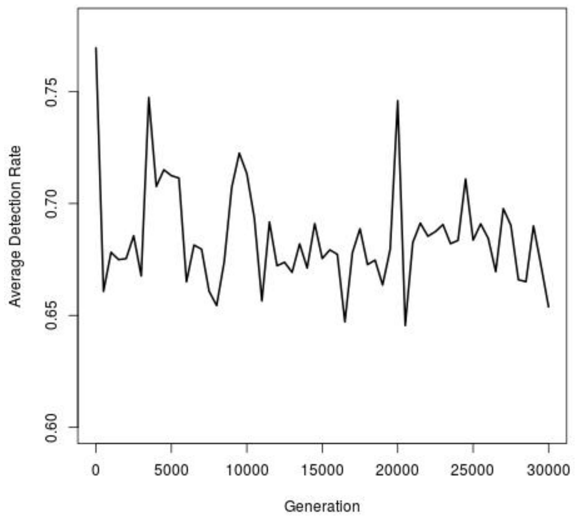


(c) Synthetic dataset with initial random labelling

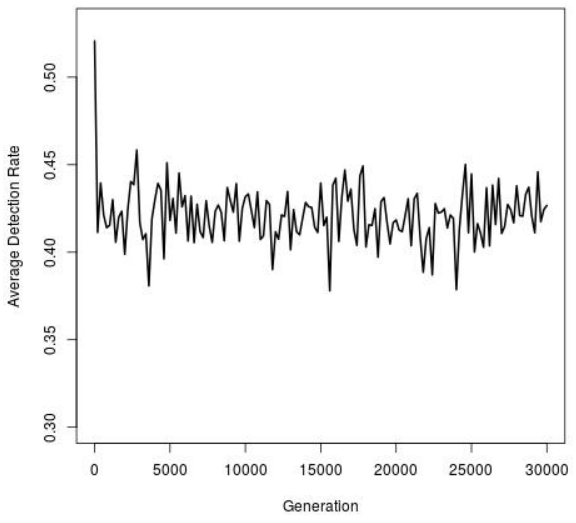


(d) Synthetic dataset with concept drift

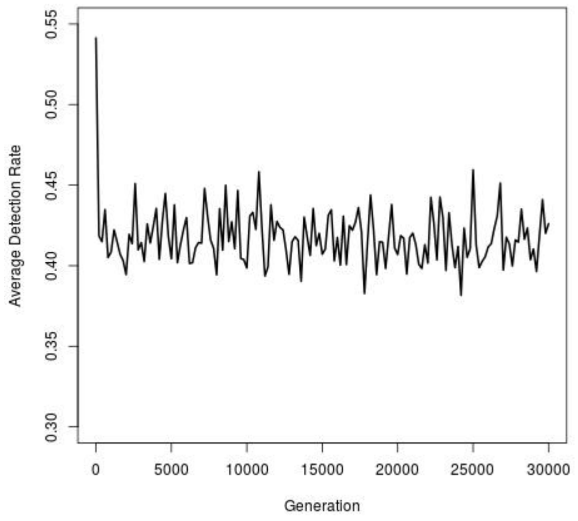
Figure 4.6: Unlabelled streaming system: Accuracy% of the committee of *Abs* at each generation during *training*.



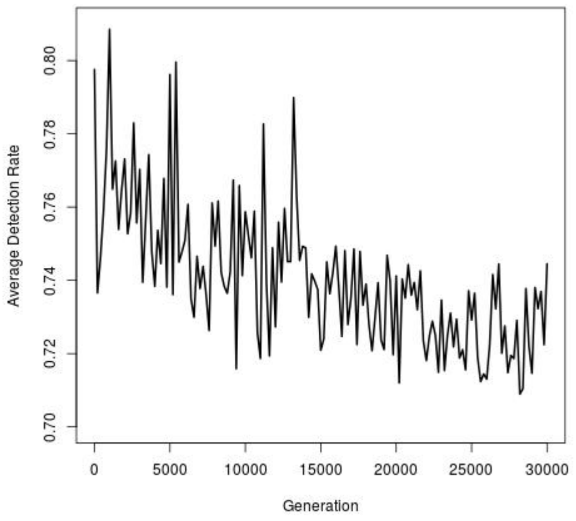
(a) Adult dataset



(b) Synthetic dataset without CD



(c) Synthetic dataset with initial random labelling



(d) Synthetic dataset with concept drift

Figure 4.7: Unlabelled streaming system: Average Detection Rate of the best *Ab* during *training*.

Similar to the accuracy, the graph of average DR of the fourth experiment as shown in Figure 4.7d experienced a linear decrease. The mean of average DR was 0.74 which was 6% better than the average DR of the labelled streaming system. This does not imply that unlabelled system performed better than its labelled counterpart as the labelled system achieved fluctuating data but without a gradual decrease in its value. An interesting observation is the differences in the average DRs of the synthetic datasets with and without concept drift. A better average DR of the synthetic data with concept drift was obtained than the synthetic dataset without concept drift. An abstract explanation to this is the distribution of the classes in the testing window which causes the average DR to be high or low.

Diversity

Figure 4.8a depicts the average accuracy of the population v/s the accuracy of the committee. Solutions of the adult dataset were slightly more diverse than its labelled counterpart (see Figure 4.3a). Diversity was low whenever the system experienced a drop in the accuracy. Low diversity from generations 1 to 10,000 caused the system to perform poorly from generations 10,000 to 20,000 thereby slightly reducing the performance. During this period, the presence of less fit solutions in the repertoire caused the probability of mutation to increase thereby introducing more diversity. The presence of a more diverse solution caused a slight overall increase in the performance which in turn resulted in the reduction of the rate of mutation leading to the reduced performance.

Experiments 2 and 3, both, maintained a low and stable diversity across all the generations (see Figures 4.8b and 4.8c). High accuracy inhibited the system from introducing more variations to it. Due to the absence of concept drift, it was expected of the unlabelled system to sustain a stable diversity across the course of its execution. The amount of diversity achieved by these datasets was sufficient enough to maintain high performance. As depicted in Figure 4.8d, the diversity of experiment 4 using testing data was found to gradually decrease after generation 10,000. The gradual decrease in the diversity was expected as the accuracy of the system also decreased gradually. The sparse differences in the average accuracies of the committee and the repertoire from generation 17,500 led to an overall decrease in the system's performance. Although the system experienced a gradual decrease in the diversity, the unlabelled streaming system achieved more diversity than its labelled counterpart (see Figure 4.3b).

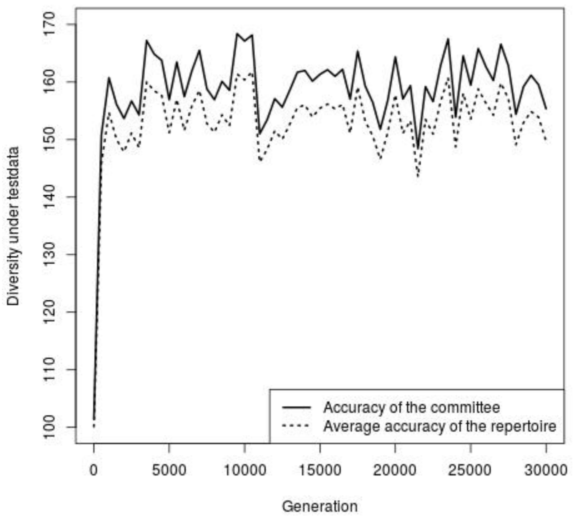
The diversity of the repertoire measured using the training exemplars is of more importance in this context as the evolvability of the solutions during training affects its performance during testing. The linear growth in the phenotypic diversity of the repertoire as shown in Figure 4.9 while maintaining genotypic diversity demonstrates the success of the unlabelled streaming system (using the adult dataset) in achieving evolvability. The phenotypic diversity increased linearly until generation 25,000 after which it appears to attain stability. It is very interesting to note the distinctions obtained in the phenotypic diversities of the labelled and the unlabelled streaming systems (see Figures 4.4 and 4.9). Although, the genotypic diversity for both the systems remain stable after generation 1, the phenotypic diversity increased at different rates. It appears that the unlabelled streaming system for the adult dataset achieved more evolvability than its labelled counterpart.

As shown in Figure 4.10, experiment 2 also experienced a linear increase in its phenotypic diversity.

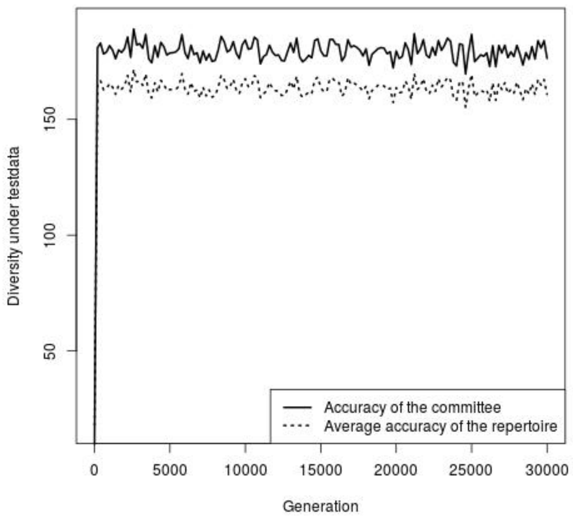
Genotypic diversity varied like a waveform between 0.12 and 0.14 values of variances and it appears to converge after generation 25,000. The variations in the diversity of the genomes did not affect the performance of the phenotypes which is a desirable trait to achieve in dynamic environments. Experiment 2 was also able to achieve a satisfactory evolvability. Experiment 3 attained the highest phenotypic diversity, around 80,000, out of all the experiments performed which is depicted by Figure 4.11. Like experiment 1, the genotypic diversity was sustained around 0.10. The phenotypic diversity linearly increased until generation 10,000 and the solutions in the repertoire achieved an almost-converged state. Similarly, the genotypic diversity linearly decreased until generation 10,000 after which it attained an almost-converged state. Although experiments 2 and 3 were performed using the same dataset, experiment 3 therefore randomness achieved more phenotypic diversity than experiment 2 which used a labelled dataset for training at the beginning. Similar to experiments 1 and 2, experiment 3 was also capable of attaining evolvable solutions.

A very interesting trend was observed in the diversity of experiment 4 using training data (see Figure 4.12). A linear decrease in its accuracy, average DR, and diversity using testing data fabricated an assumption that the results of the statistical measures will also follow the same pattern. But instead, similar to experiment 1, a linear growth was obtained in the phenotypic diversity of the system while sustaining genotypic diversity using training data. Different diversities for experiment 4 using training and testing were achieved. The unlabelled streaming system acts like a static system as it was trained using the same dataset for 500 generations until new data with concept drift is introduced to it. A gradual increase in the phenotypic diversity is proof enough that the system did in fact achieve evolvability. Unlike its labelled counterpart where the system received no or very low evolvability (see Figure 4.5), the solutions generated by the unlabelled system were satisfactory evolvable. Unfortunately, the best solutions obtained (during training) for testing failed to represent the current scenario of the environment since the data had drifted a lot by then. Analysing the statistics of the training process is self-assuring that the system did not fail outright while learning and there is a need to test the solutions using a more relevant environment.

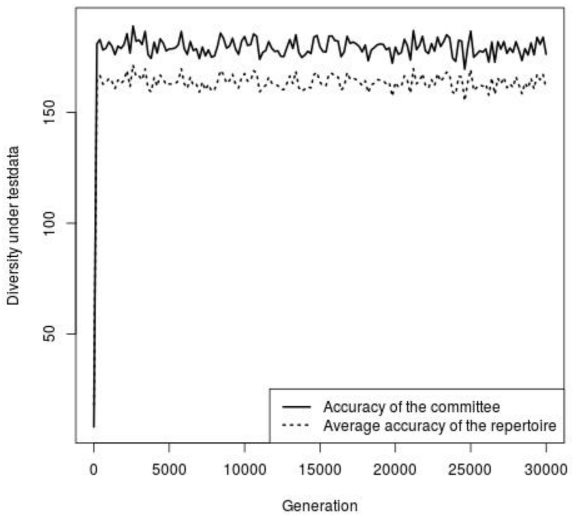
In summary, the labelled and unlabelled streaming systems performed similarly to the static system which proves that the GIP-based system was capable of handling fast environmental changes and the labelling process was accurate. The phenotypic diversities of all the experiments of the unlabelled streaming system were better than their labelled counterparts. A gradually increasing phenotypic diversity in the presence of a sustained genotypic diversity made the systems achieve evolvability which is desirable to achieve good performance for streaming problems. The GIP-based systems seem to perform better in the presence of concept drift than the traditional ML classification algorithms. The declining performance of experiment 4 of the unlabelled streaming system is an indication to increase the *time to label* and the training iterations in order to incorporate all the environmental changes which may improve the testing performance but will result into a significant increase in the cost of computation.



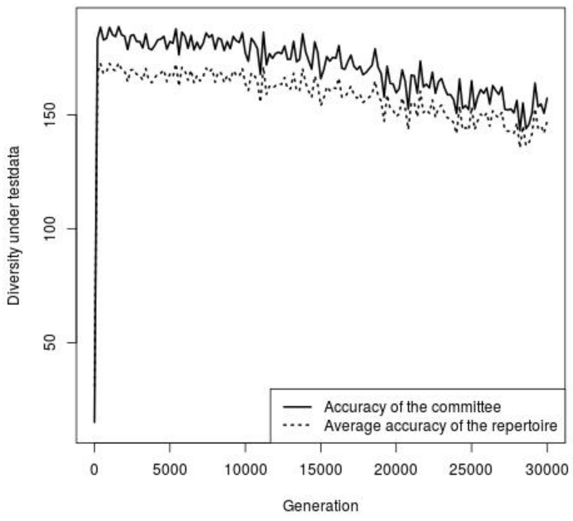
(a) Adult dataset



(b) Synthetic dataset without CD



(c) Synthetic dataset with initial random labelling



(d) Synthetic dataset with concept drift

Figure 4.8: Unlabelled streaming system: Measuring Diversity using test cases. The dotted line represents the average accuracy of the population and the solid line represents the number of true positives of the committee of *Abs*.

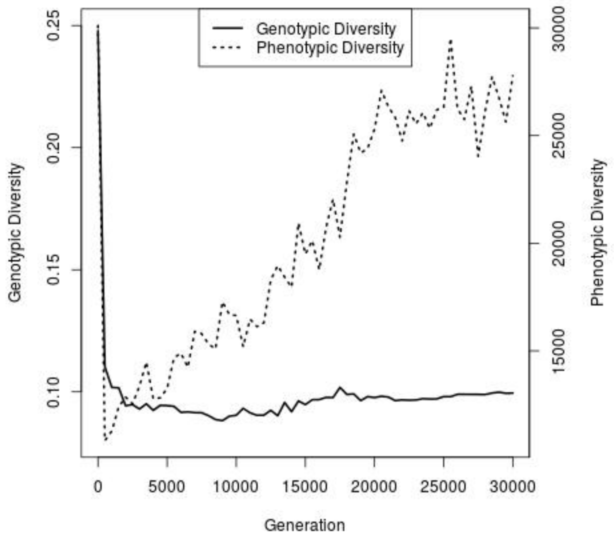


Figure 4.9: Unlabelled streaming system: Measuring genotypic versus phenotypic diversity for adult dataset during *training*.

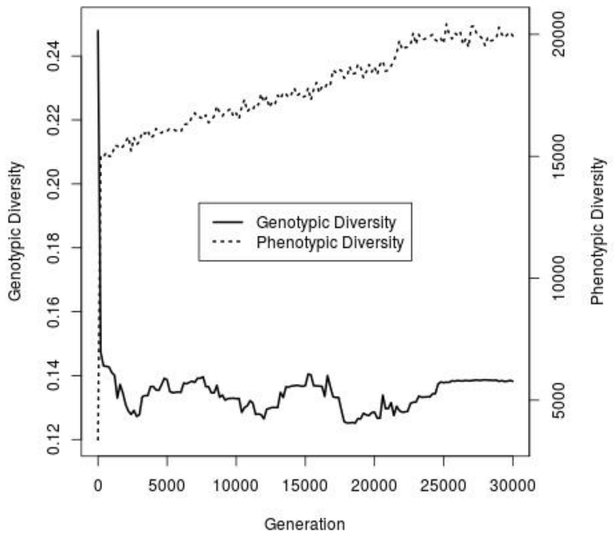


Figure 4.10: Unlabelled streaming system: Measuring genotypic versus phenotypic diversity for Synthetic dataset without concept drift during *training*.

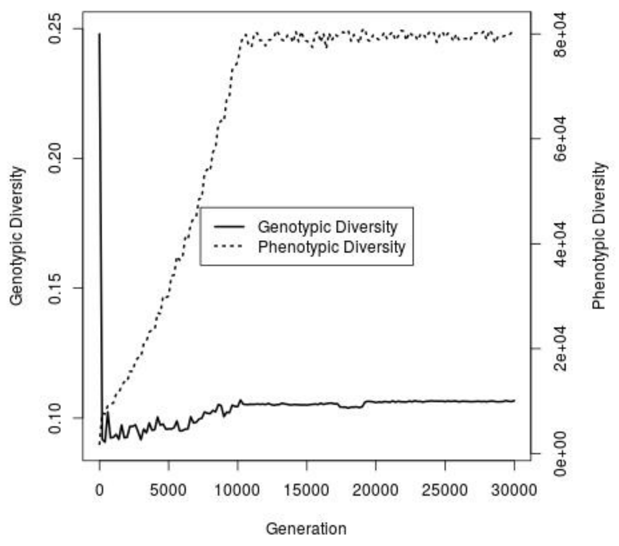


Figure 4.11: Unlabelled streaming system with initial random labelling: Measuring genotypic versus phenotypic diversity for synthetic dataset without concept drift during *training*.

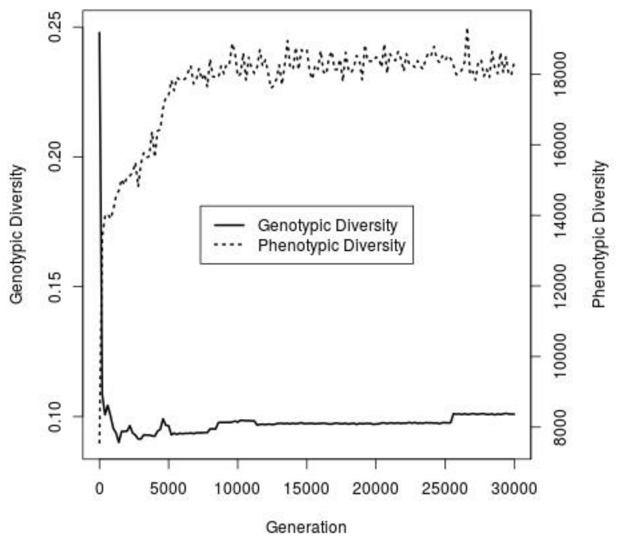


Figure 4.12: Unlabelled streaming system: Measuring genotypic versus phenotypic diversity for synthetic dataset with concept drift *training*.

Chapter 5

Conclusions and Future Work

This work successfully accomplished its primary aim of automatically generating classifiers for streaming data in real-time using GIP. This chapter provides a concluding summary of the question presented, *"How successfully did we automatically generate solutions to classify streaming data in real-time?"* in Section 5.1. This chapter also dwells on the possible applications for this study and extensions to the work to potentially improve the performance of the GIP variants in Section 5.2.

5.1 Conclusions

Let us start by revisiting the proposed hypotheses:

1. The automatic creation of classifiers that label the data could be done in real-time.
2. *Evolvability* could aid in maintaining diversity which in turn could address concept drift.

No research was found which employs GIP for classification problems. Also, to the best of our knowledge none of the EA-based systems simultaneously evolve classifiers while labelling streaming data in real-time. A static GIP variant was first introduced in order to automatically generate classifiers for static data. Although the results of the static system could have been made better by training the system using more data, they were satisfactory enough to build a streaming system on it for classifying labelled data in real-time. The labelled streaming system produced the most satisfactory results as it performed better than its static counterpart and than the benchmark problem. The labelled streaming system trained using the adult dataset achieved the most diverse solutions which is a by product of evolvability. At every time of its execution, the system maintained diverse solutions which was required to explore the ever-changing search space. The labelled system trained using the synthetic dataset with concept drift did not maintain a diverse set of solutions at every time of its execution implying that it did not achieve much evolvability. Typically, in an EA, it is infeasible to introduce variations to a converged solution. But our AIS-based system was capable of introducing variations to it in small amounts whenever the environment changed. This proves that hypermutation does in fact aid in restoring diversity by maintaining a balance between exploration and exploitation, thereby verifying the hypothesis of Dempsey et al stated in [4]. Genotype-to-phenotype mappings and neutral mutations may also have aided in achieving evolvability. The amount of diversity achieved for both the datasets was plenty for addressing concept drift and achieving satisfactory performance. Our work also proved that a grammar-based AIS does in fact perform better than a non-grammar based system as our streaming system performed better than GP-based benchmark problem. It also proves, rather abstractly, that an AIS-based system performs better in dynamic environments than static environments.

The satisfactory results achieved by the labelled streaming system inspired us to continue our research in streaming environments using unlabelled data. Similar results achieved by the adult dataset for labelled and unlabelled streaming systems provided us a proof of concept that the active learning process accurately labelled the streaming data. This verifies our first hypothesis since the unlabelled streaming system successfully generated classifiers automatically while labelling the data in real-time.

We noticed that when a dynamic system is set to operate as a static system for a certain amount of generations, it achieves a linear growth in its phenotypic diversity while sustaining genotypic diversity, which is the most essential in dynamic environments as maintaining different solutions can easily address environmental changes. Although it was not one of our hypotheses, we also experimented with random labelling to observe the performance of the system using the synthetic dataset without concept drift. The results achieved by the systems using a labelled set and random labelling respectively were quite similar thereby obsoleting the need to use past labelled instances in the future. The linearly decreasing performance and the linearly increasing diversity of the models of the unlabelled system using synthetic data with concept drift demands the need of training the system at every generation. Training the system at every generation requires labelling the streaming data at every generation which was infeasible because of its considerable computational expense and our limited access to a powerful computer. We also proved that our GIP system which is evolutionary in nature is more capable of addressing concept drift than the non-evolutionary ML algorithms of C4.5 and NB.

5.2 Future Work

1. **Test data and adaptive control parameters** Our systems were tested using only one real-world dataset. Thus, there is a lot of room for experimenting the implemented systems using different types of real-world datasets employed in different settings, static and dynamic. Also, more experiments are required to test the system using synthetic data with varying intensities of concept drift. It would be interesting to analyse the performance of the GIP system where concept drift occurs every 10 instances. This thesis proved that the AIS-based system performs better than the GP system in dynamic environments. Due to the limitation of time, we could not compare our results with various ML techniques such as neural networks, SVMs, and so on and thus we have left this as a future work.

   Due to the limitation of time and resources, we could not analyse the performance of the systems using different population sizes, cloning rates, and mutation rates. Varying control parameters could potentially result in a decreased or increased diversity which is particularly important in dynamic environments. Also, many EAs and AIS algorithms employ self-adaptive parameters [32]. Self-adaptation or evolving the values of the control parameters can potentially prove beneficial in dynamic environments. For example, a detected concept drift can signal the system to increase the mutation rate in order to introduce more changes in the population. As noted in Chapter 2, control parameters can be dynamically adapted either deterministically through a feedback loop, or they could be evolved during the execution of a system [37].

2. **Maintaining an archive** The success of the symbiotic-based GP system [6] in streaming environments is an inspiration to employ an archive for retaining past important exemplars. So far, we have maintained a buffer to store the labelled instances which are replaced by a FIFO policy. A huge bias in the classes, for example, can cause the streaming window to have unrepresentative data. A system frequently trained using unrepresentative data can result into misclassification thereby reducing the performance or the average DR of the system. Thus retaining past important exemplars can increase the average DR of the system thereby increasing its performance.

   One can also maintain an archive of past important models. During the execution of the system, especially in streaming environments, it is plausible to have different best models during different generations. Maintaining past models may be reused in the future if the system encounters similar data. Consider an example, the electricity demand in Canada follows a predictable trend. The demand of electricity reduces in the summer. In this scenario, it is sensible to store the past models which can be reused in the future. One can also employ a replacement policy to replace the models which have been never used. Models can be retained in an archive either periodically or by identifying the best models by analysing its performance.

3. **Population** A converged population will prevent the system from introducing variations to it in spite of the environmental changes which leads to a decreased performance. Many EAs are known to instantiate a new population after detecting a decrease in the model(s)' performance below a certain threshold [4]. The new population can be seeded from the past best model or from the models retained in an archive. The new population can also be generated randomly. The new population generated and evolved represents the current environment which is more effective than maintaining a converged population which represents some past environment.

   Maintaining sub-populations on different parts of the training exemplars and co-evolving them thereby maintaining sub-optimal solutions could potentially aid in sustaining diversity. Our system could benefit from this as the main goal of this work was to achieve evolvability.

4. **Detecting concept drift** Concept drift can occur at any point of a system's lifetime. Thus, periodically labelling data in spite of drifting concepts utilizes a lot of resources which could be easily avoided. An optimal way of labelling data is by detecting concept drift. Concept drift can be detected with or without labels. Detecting concept drift without labels is more optimal as the system does not have to temporarily label streaming data solely for detecting changes. Conditional probability can be used to detect changes using unlabelled data [2].

5. **Evolvable grammar** An effective way of improving evolvability by achieving more diversity and survivability is by the use of an evolvable grammar. Dempsey et al introduced this concept in his study, Grammatical Evolution by Grammatical Evolution (GE^2) [4] which proved more successful than the traditional GE in dynamic environments. It was employed for symbolic regression problems where the target function changed every 20 generations. Evolving genetic code is important in the cases where one has no or little information about the environmental changes or where achieving adaptability is crucial for ensuring survivability. Evolving the representation potentially provides the population the ability to survive. A single change in the grammar can re-introduce lost genetic code. In GE^2, the grammar is evolved to change the mappings of the codons to different production rules. Two types of grammars are maintained for this purpose - meta grammar and solution grammar and thus each solution is represented by two variable length binary strings for meta grammar and solution grammar respectively. The meta grammar is responsible for evolving solution grammar. It is not necessary to evolve all the non-terminals of a grammar. Crossovers occur between homologous binary strings of the solution. That is, crossover between a meta grammar and a solution grammar can not occur.

6. **Streaming Interface** The implemented systems accepted streaming data in a linear fashion. But in real-world, streaming data can stream in asynchronously at irregular intervals. This also implies that for some duration, the system may not receive any data from the source(s). Provisions could be made to accept data from multiple sources. The replacement policy for the interfaces could be changed to a more optimal one in the future. For example, instead of flushing out the least recent instances, the least important instances in terms of where it lies in the search space could be flushed out. Likewise, several such replacement policies can be employed.

7. **Applications** Flexible streaming interfaces and various parameters for controlling the duration of training make our system deployable to various real-world dynamic applications such as sensor networks in a power plant, analysing electricity demand, and recommender systems to name a few. Also, defining grammar for each problem is an easy task which also makes our systems practically deployable. The static variant of the classification system can be deployed for various classification problems such as classifying medical data or classifying environmental data.

Appendix A

Grammar defined for the adult dataset

The grammar for the adult dataset is represented using the following equation:

$G = \langle\ N, T, P, S\ \rangle$

where,

$N = \langle$ DT, DTp $\rangle$

$T = \langle$ 0, 1, AGE, WORKCLASS, FNLWGT, EDUCATION, EDUCATION-NUM, MARITAL-STATUS, OCCUPATION, RELATIONSHIP, RACE, SEX, CAPITAL-GAIN, CAPITAL-LOSS, HOURS-PER-WEEK, NATIVE-COUNTRY $\rangle$

$S = \langle$ DT $\rangle$

$P =$

$\langle DT \rangle$ ::= AGE DTp DTp DTp |
WORKCLASS DTp DTp DTp DTp DTp DTp DTp DTp |
FNLWGT DTp DTp DTp DTp DTp |
EDUCATION DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp |
EDUCATION-NUM DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp |
MARITAL-STATUS DTp DTp DTp DTp DTp DTp DTp |
OCCUPATION DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp

DTp DTp DTp |
RELATIONSHIP DTp DTp DTp DTp DTp DTp |
RACE DTp DTp DTp DTp DTp |
SEX DTp DTp |
CAPITAL-GAIN DTp DTp |
CAPITAL-LOSS DTp DTp |
HOURS-PER-WEEK DTp DTp DTp DTp |
NATIVE-COUNTRY DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp DTp

$\langle DTp \rangle$ ::= DT | 0 | 1

Bibliography

[1] Leandro N De Castro and Fernando J Von Zuben. Learning and optimization using the clonal selection principle. *Evolutionary Computation, IEEE Transactions on*, 6(3):239–251, 2002.

[2] Malcolm I Heywood. Evolutionary model building under streaming data for classification tasks: opportunities and challenges. *Genetic Programming and Evolvable Machines*, 16(3):283–326, 2015.

[3] Michael O'Neill, Leonardo Vanneschi, Steven Gustafson, and Wolfgang Banzhaf. Open issues in genetic programming. *Genetic Programming and Evolvable Machines*, 11(3-4):339–363, 2010.

[4] Ian Dempsey, Michael O'Neill, and Anthony Brabazon. *Foundations in grammatical evolution for dynamic environments*, volume 194. Springer, 2009.

[5] Lee Altenberg. The evolution of evolvability in genetic programming. *Advances in genetic programming*, 3:47–74, 1994.

[6] Aaron Atwater, Malcolm I Heywood, and Nur Zincir-Heywood. Gp under streaming data constraints: A case for pareto archiving? In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, GECCO '12, pages 703–710, New York, NY, USA, 2012. ACM.

[7] Aaron Atwater and Malcolm I Heywood. Benchmarking pareto archiving heuristics in the presence of concept drift: diversity versus age. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 885–892. ACM, 2013.

[8] Xingquan Zhu, Peng Zhang, Xiaodong Lin, and Yong Shi. Active learning from stream data using optimal weight classifier ensemble. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 40(6):1607–1621, Dec 2010.

[9] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2005.

[10] Xingquan Zhu, Xindong Wu, and Qijun Chen. Eliminating class noise in large datasets. In *ICML*, volume 3, pages 920–927, 2003.

[11] Shayok Chakraborty, Vineeth Balasubramanian, and Sethuraman Panchanathan. Dynamic batch mode active learning. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 2649–2656, June 2011.

[12] Ali Vahdat, Aaron Atwater, Andrew R McIntyre, and Malcolm I Heywood. On the application of gp to streaming data classification tasks with label budgets. In *Proceedings of the 2014 Conference Companion on Genetic and Evolutionary Computation Companion*, GECCO Comp '14, pages 1287–1294, New York, NY, USA, 2014. ACM.

[13] Burr Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52(55-66):11, 2010.

[14] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *The Journal of Machine Learning Research*, 2:45–66, 2002.

[15] Indre Zliobaite, Albert Bifet, Bernhard Pfahringer, and Graham Holmes. Active learning with drifting streaming data. *Neural Networks and Learning Systems, IEEE Transactions on*, 25(1):27–39, 2014.

[16] David A Cohn, Zoubin Ghahramani, and Michael I Jordan. Active learning with statistical models. *Journal of artificial intelligence research*, 1996.

[17] David D Lewis and William A Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–12. Springer-Verlag New York, Inc., 1994.

[18] John Ross Quinlan. *C4. 5: programs for machine learning.* Morgan Kaufmann, 1993.

[19] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine learning*, 29(2-3):103–130, 1997.

[20] Jorn Bakker, Mykola Pechenizkiy, Indrė Žliobaitė, Andriy Ivannikov, and Tommi Kärkkäinen. Handling outliers and concept drift in online mass flow prediction in cfb boilers. In *Proceedings of the Third International Workshop on Knowledge Discovery from Sensor Data*, SensorKDD '09, pages 13–22, New York, NY, USA, 2009. ACM.

[21] Indrė Žliobaitė. Learning under concept drift: an overview. *arXiv preprint arXiv:1010.4784*, 2010.

[22] Patrick Lindstrom, Sarah Jane Delany, and Brian Mac Namee. Handling concept drift in text data stream constrained by high labelling cost. 2010.

[23] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Advances in artificial intelligence–SBIA 2004*, pages 286–295. Springer, 2004.

[24] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

[25] Andries P. Engelbrecht. *Computational Intelligence: An Introduction.* Wiley Publishing, 2nd edition, 2007.

[26] Charles Darwin and George Gaylord Simpson. *The origin of species by means of natural selection: Or, The preservation of favoured races in the struggle for life*. Collier Books New York, 1962.

[27] Charles Darwin. *The variation of animals and plants under domestication*, volume 2. O. Judd, 1868.

[28] David B Fogel. What is evolutionary computation? *IEEE Spectrum*, 37(2):26, 28–32, Feb 2000.

[29] Wolfgang Banzhaf, Peter Nordin, Robert E Keller, and Frank D Francone. *Genetic programming: an introduction*, volume 270. Morgan Kaufmann San Francisco, 1998.

[30] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1(1):1–23, 1993.

[31] John R Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[32] Riccardo Poli, William B. Langdon, and Nicholas F. McPhee. *A field guide to Genetic Programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[33] Ronald W Morrison. *Designing evolutionary algorithms for dynamic environments*. Springer Science & Business Media, 2013.

[34] Nadav Kashtan, Elad Noor, and Uri Alon. Varying environments can speed up evolution. *Proceedings of the National Academy of Sciences*, 104(34):13711–13716, 2007.

[35] David E Goldberg et al. *Genetic algorithms in search optimization and machine learning*, volume 412. Addison-wesley Reading Menlo Park, 1989.

[36] Helen G Cobb. An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Technical report, DTIC Document, 1990.

[37] Zheng Yin, Anthony Brabazon, Conall O'Sullivan, and M O'Neil. Genetic programming for dynamic environments. In *Proceedings of the International Multiconference on Computer Science and Information Technology*, pages 437–446, 2007.

[38] Michael O'Neill and Conor Ryan. Grammatical evolution. *Evolutionary Computation, IEEE Transactions on*, 5(4):349–358, Aug 2001.

[39] Jennifer J Freeman. A linear representation for gp using context free grammars. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 72–77, 1998.

[40] R Keller and W Banzhaf. Gp using mutation, reproduction and genotype-phenotype mapping from linear binary genomes into linear lalr phenotypes. In *Genetic Programming 1996: Proc. 1st Annu. Conf*, pages 116–122. Stanford, CA: MIT Press, 1996.

[41] Gerald D Elseth and Kandy D Baumgardner. *Principles of modern genetics.* Brooks/Cole Publishing Company, 1995.

[42] Motoo Kimura. *The neutral theory of molecular evolution.* Cambridge University Press, 1984.

[43] Marc Ebner, Mark Shackleton, and Rob Shipman. How neutral networks influence evolvability. *Complexity*, 7(2):19–33, 2001.

[44] Michael O'Neill and Conor Ryan. Grammatical evolution by grammatical evolution: The evolution of grammar and genetic code. In *Genetic Programming*, pages 138–149. Springer, 2004.

[45] Steven A Hofmeyr and Stephanie Forrest. Immunity by design: An artificial immune system. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1289–1296. Citeseer, 1999.

[46] Leandro Nunes De Castro and Jonathan Timmis. *Artificial immune systems: a new computational intelligence approach.* Springer Science & Business Media, 2002.

[47] Dipankar Dasgupta, Senhua Yu, and Fernando Nino. Recent advances in artificial immune systems: Models and applications. *Applied Soft Computing*, 11(2):1574 – 1587, 2011. The Impact of Soft Computing for the Progress of Artificial Intelligence.

[48] Zhou Ji and Dipankar Dasgupta. Revisiting negative selection algorithms. *Evolutionary Computation*, 15(2):223–251, 2007.

[49] José Pacheco and José Félix Costa. The abstract immune system algorithm. In *Unconventional Computation*, pages 137–149. Springer, 2007.

[50] Uwe Aickelin and Steve Cayzer. The danger theory and its application to artificial immune systems. *arXiv preprint arXiv:0801.3549*, 2008.

[51] Julie Greensmith, Uwe Aickelin, and Steve Cayzer. Introducing dendritic cells as a novel immune-inspired algorithm for anomaly detection. In *Artificial Immune Systems*, pages 153–167. Springer, 2005.

[52] Nikolaos Nanas and Anne De Roeck. Multimodal dynamic optimization: from evolutionary algorithms to artificial immune systems. In *Artificial Immune Systems*, pages 13–24. Springer, 2007.

[53] Petr Musilek, Adriel Lau, Marek Reformat, and Loren Wyard-Scott. Immune programming. *Information Sciences*, 176(8):972–1002, 2006.

[54] Sir Frank Macfarlane Burnet et al. *The clonal selection theory of acquired immunity*, volume 3. Vanderbilt University Press Nashville, 1959.

[55] Jennifer A White and Simon M Garrett. Improved pattern recognition with artificial clonal selection? In *Artificial Immune Systems*, pages 181–193. Springer, 2003.

[56] Andrew Watkins, Jon Timmis, and Lois Boggess. Artificial immune recognition system (airs): An immune-inspired supervised learning algorithm. *Genetic Programming and Evolvable Machines*, 5(3):291–317, 2004.

[57] Jason Brownlee. Clonal selection theory & clonalg-the clonal selection classification algorithm (csca). *Swinburne University of Technology*, 2005.

[58] Anurag Sharma and Dharmendra Sharma. Clonal selection algorithm for classification. In *Artificial Immune Systems*, pages 361–370. Springer, 2011.

[59] Heder S Bernardino and Helio J C Barbosa. Grammar-based immune programming. *Natural Computing*, 10(1):209–241, 2011.

[60] Heder S Bernardino and Helio J C Barbosa. Grammar-based immune programming for symbolic regression. In *Artificial Immune Systems*, pages 274–287. Springer, 2009.

[61] Pierre L'ecuyer, Richard Simard, E Jack Chen, and W David Kelton. An object-oriented random-number package with many long streams and substreams. *Operations research*, 50(6):1073–1075, 2002.

[62] Martin Hemberg and Una-May O'Reilly. Extending grammatical evolution to evolve digital surfaces with genr8. In *Genetic Programming*, pages 299–308. Springer, 2004.

[63] Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. Mining data streams: A review. *SIGMOD Rec.*, 34(2):18–26, June 2005.

[64] Alexey Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 106:2, 2004.

[65] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2007.

[66] Charu C Aggarwal. *Data streams: models and algorithms*, volume 31. Springer Science & Business Media, 2007.

[67] M. Lichman. UCI machine learning repository, 2013.

[68] Aaron Atwater. *Towards coevolutionary genetic programming with Pareto archiving under streaming data*. PhD thesis, Dalhousie University Halifax, 2013.

[69] Michael O'Neill and Conor Ryan. Under the hood of grammatical evolution. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 2*, pages 1143–1148. Morgan Kaufmann Publishers Inc., 1999.

Index