LABEL FREE CHANGE DETECTION ON STREAMING DATA WITH COOPERATIVE MULTI-OBJECTIVE GENETIC PROGRAMMING

by

Sara Rahimi

Submitted in partial fulfillment of the
requirements for the degree of
Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
August 2013

To Dr. Andrew McIntyre,
for making it possible for me.

Table of Contents

List of Tables

List of Figures

Abstract

Classification under streaming data conditions requires that the machine learning approach operate interactively with the stream content. Thus, given some initial machine learning classification capability, it is not possible to assume that the process 'generating' stream content will be stationary. It is therefore necessary to first detect when the stream content changes. Only after detecting a change, can classifier retraining be triggered. Current methods for change detection tend to assume an entropy *filter* approach, where class labels are necessary. In practice, labelling the stream would be extremely expensive. This work proposes an approach in which the behaviour of GP individuals is used to detect change *without* the use of labels. Only after detecting a change is label information requested. Benchmarking under three computer network traffic analysis scenarios demonstrates that the proposed approach performs at least as well as the filter method, while retaining the advantage of requiring no labels.

Acknowledgements

I would like to express my deepest appreciation to Dr. Andrew McIntyre and Dr. Malcolm Heywood for guiding me throughout this journey and showing so much patience. Likewise, I would like to thank Dr. Nur Zincir-Heywood for her guidance . Without the help and nurturing that each of you provided, this thesis would not have been possible.

A big thank you goes to my lovely parents and other family members, especially my sisters who have listened to and supported me through all the ups and downs of my life.

Chapter 1

Introduction

The problem of growth in size of the data generated in the real world is not a new challenge. These constantly increasing data volumes, known as "data streams", need to be stored, classified, and analyzed for further usage. One way to do this is through conventional offline classification algorithms. These algorithms assume that independent training and test sets exist, and that the content of the training set can be revisited without penalty. However, there is no guarantee that the structure of a data stream will always remain constant and never change. i.e., the process generating stream content is said to vary over time (non-stationary). Thus, by sampling data streams and forming a limited and separated amount of them as *training set*, we are only able to classify data for a short period of time. As a result, we need classification algorithms that are not static/linear and able to adapt accordingly to such changes, and can therefore be used in more continuous and streaming contexts.

Real world datasets, such as financial markets, computer network traffic analysis, power utility management and autonomous systems in general, are frequently of a constantly changing nature. Thus, data has an explicitly temporal property, such that the validity of any given classifier has a finite lifetime. Under such conditions, it is no longer possible to partition data into training and test sets. Thus, the fundamental assumption of off-line training is no longer true. As a result of these changes, algorithms should never stop learning and revising what they have learnt. In the most general case, streaming data or online learning requires that classifiers are constructed on a continuous basis (online mode). Online algorithms keep evolving as new data gets introduced into the stream, and evaluation is done based on the most recently learned patterns. MOA:*Massive Online Analysis* [10], provides a hands-on collection of these online algorithms. In addition, Figure 1.1 shows an application example of online algorithm for identifying spam emails [1].

Classification under the Genetic Programming (GP) domain has been studied

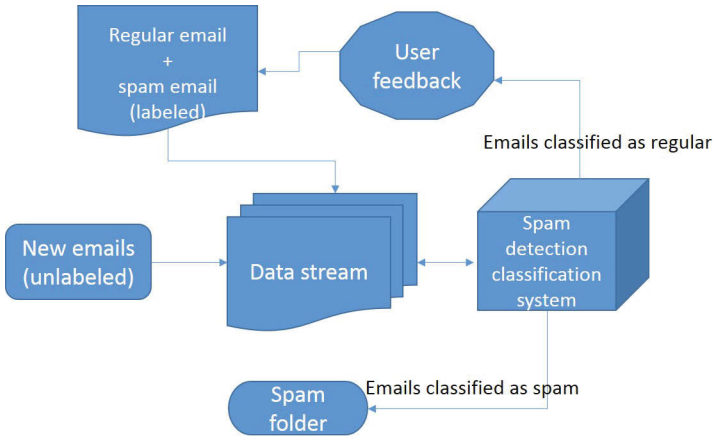


Figure 1.1: An example of online algorithm for spam email detection [1]

extensively in machine learning. Some solutions utilize separate classifiers for each class and others tend to decompose the problem into smaller subsets and classify data as an "ensemble behavior." All these approaches have an essential feature in common: they are canonical. In each case, a population of candidate classifiers evolve against the content of the training set. Post training evaluation is performed relative to the test set. In the advent of larger data sets, it becomes increasingly necessary to decouple the cost of fitness evaluation from the cardinality of the training set. Thus, different forms of re-sampling are typically employed. The underlying objective of such schemes is to identify training exemplars that promote the most development of the classifiers at any particular generation. In the case of an online GP classifier, the first training set exemplars might provide the basis for identifying initial champion GP classifier. The GP champion then provide labels for the exemplars appearing in the stream.

This online method of constantly learning and revising is potentially time- and resource-consuming, thus conflicting with the goal of efficiency and optimization. Moreover, it is important to note that data streams are not necessarily changing drastically. The variations occurring in the generated data streams depends on the nature of the data being collected and used. A general example is the data collected from computer networks. Adding a new machine might add new data in the stream, for instance, with the widespread usage of wireless devices and their mobility may cause data to abruptly change. One appropriate solution might be identifying these

changes and then revisiting and updating previously learned values afterward. This approach requires some sort of change detection mechanism, such as what has been introduced in [33, 37, 40]. Moreover, in most cases, the pairwise sliding windows technique is deployed on the stream after which a class-specific entropy comparison is made between the two sliding windows (i.e., using a classifier independent filter). Naturally, this implies that each exemplar requires a class label; in addition, such approaches generally fail to identify recurrence. Thus, a change detection method that is independent of labels and capable of handling recurrence is needed. Such a method has been introduced in this research using a GP classification framework.

This research begins by introducing background information and literature review in Chapter 2. Then it continues, in Chapters 3 and 4, by explaining the framework and methodology used in this thesis. To be more specific, Chapter 4 starts by applying the standard entropy-based framework for change detection [37]. Pareto archiving will then be introduced in the context of a recent study which concluded that, under a labeled stream, GP classifiers can be built that approximate the strength of classifiers constructed from fitness evaluated against the entire stream [6]. We will first demonstrate that information from the Pareto archive can be used to provide an alternative basis for defining sliding window content under the entropy formulation for change detection; albeit under the labeled stream requirement. Finally, in order to support change detection without a prior labeling of the stream, change detection is conducted on the output space of a team of cooperatively coevolved champion GP classifiers. This provides the benefit of being able to incrementally change the collection of champion classifiers. Thus, rather than relying on a single champion – whose behavior might be re-encountered under cyclic non-stationary processes – classifiers act as a team of non-overlapping behaviors. The starting point for developing such a capability in streaming data will be a previously proposed framework for Co-evolutionary Multi-objective GP (CMGP) [30, 31], explained in Chapter 3. The proposed scheme for unlabeled change detection is shown to at least match that of those requiring all exemplars to be labeled. As such, this represents a significant improvement, as the proposed approach does so by decoupling the labeling requirement from the stream itself. An overview of the algorithm can be seen in Figure 1.2.

In Chapter 5, results are reported under three network traffic analysis data sets

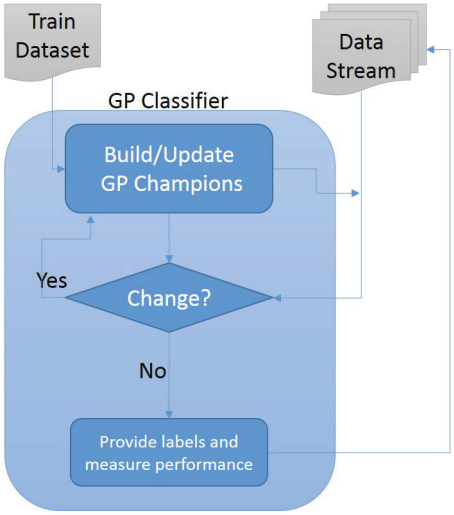


Figure 1.2: An overview of applying On-line Genetic programming classifier on data streams with change detection

in which changes have been explicitly applied into the stream. There has been some analysis of encrypted traffic using various machine learning methods without IP addresses, port number and payload information [2, 3, 8]. A similar dataset called UCIS [3] has been used as the first data set. The NIMS is another dataset which was artificially produced by Arndt [3] and it has been used in combination with UCIS as the second dataset. The third dataset is KDD'99 [15]. It is a collection of different network attacks in an artificial set. Further information about these data sets are provided in Section 5.1. At the end, the conclusion and future work is provided in Chapter 6.

Chapter 2

Background & Literature Review

2.1 Problem Definition

In the following section, three concepts - data stream, concept change, and on-line algorithms - are described. These terms are used throughout this research, and can be applied to diverse application domains such as weather prediction, customer preferences, spam categorization, user modeling, monitoring in biomedicine, monitoring industrial processes, fault detection and diagnosis, safety of complex system, financial markets, computer network traffic analysis, and power utility management.

2.1.1 Data Stream

A data stream is a collection of sequential instances where each instance includes a set of features or attributes. Each data instance is associated with a class label i.e., data instances are self-contained. Figure 2.1 shows an example of such a data stream. Prediction methods try to analyze the labeled instances and their features in order to predict the next unlabeled instance in the stream. These instances have an unlimited growth rate, such that it is almost impossible for classification algorithms to learn based on the whole collection of previously visited data. Specifically, the context where these data streams are generated and collected is not stable. Thus, underlying process 'generating' the stream content might alter over time, so the learned model needs to be revised occasionally.

In this thesis, the stream data will take the form of "flows" – where this represents a collection of attributes / statistics used to summarize packet data from a computer network e.g., [3]. Each flow is differentiated from others by source IP, source port, destination IP, destination port, and transport protocol. It also includes various characteristics or features such as standard deviation of packet length and mean inter-arrival time. Some recent papers – such as [2, 3, 8, 16] – demonstrate that flows

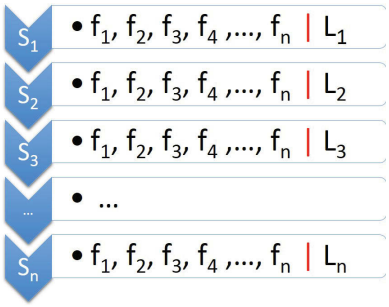


Figure 2.1: An example of data stream. Each flow is shown by "S", each feature in flow by "f" and respective labels by "L"

without IP addresses, port numbers and payload information can be used for distinguishing between different types of network traffic using machine learning algorithms constructed under an off-line batch training framework. Arndt [3], Alshammari [2], Bacquet [8], and Erman [16] use 46, 22, 38 and 11 features, repectively, for every flow in the data stream. They have reached these numbers of features based on trial and error. One of the datasets used in this research (UCIS) is based on that collected by Arndt [3, 4], where the number of flows and features (40) have already been characterized. This dataset consists of Dalhousie University network traffic traces (called UCIS); it is described in greater detail in Section 5.1 (i.e., it was necessary to modify the data set in order to introduce non-stationary properties.)

2.1.2 Concept Change

As discussed earlier, data streams consist of collections of data accumulated on a continuous basis. The underlying context of this data collection process may change over time. Moreover, the distribution of data may also vary over time. Considering the increasing amounts of data that we are faced with, these differences are inevitable. Thus, relying on a model that is built according to the first batch of data is not practical since the model needs to be updated regularly in order to be responsive and adaptive to such changes. With this in mind, it is important to identify changes and their types accurately in order to apply a proper approach. Any kind of variation in the underlying context of generating data streams could result in a dramatic or slight change in the main concept, and these concept changes could be either new

or repeated. This repeat pattern in changes includes seasonal and irregular cycles. According to Tsymbal's survey [36], the ultimate goal is to establish a system that is able to adapt quickly, differentiates noise from real drift, and handles recurring changes.

Different types of changes can be categorized into two main groups: gradual vs abrupt changes and real vs virtual changes. In the following section we will describe each one of these change types. At the end considering real and artificially produced datasets, an overview of existing changes in famous datasets will be given.

- Gradual vs Abrupt Change:

  Sudden changes in the underlying context can cause abrupt changes in target concept, which is called *concept shift* [41]. For example, a customers preferences might vary drastically after graduation from college. On the other hand, gradual changes, or so called *concept drift* [41], are those that happen incrementally over time. For instance, approaching the autumn season causes a gradual decrease in weather temperature prediction models. Vorburger [37] introduced a change detection method for such abrupt concept drifts which is explained in detail later, in Section4.2.

- Real vs Virtual Change:

  Changes in the distribution of data are referred to as *virtual concept shift* [40], and changes in the underlying concept are referred to as *real concept drift* [39]. Salganicoff [33] states that *virtual concept shift* is the same as *sampling shift*, and Yang [41] refers to it as *sampling change*. *Virtual concept shift* is most commonly studied under the spam categorization domain. In fact, it emphasizes that, even though the underlying concept might be the same, learning models need to be revised due to high error rates.

- Concept Change in Real vs Artificial Datasets:

  A widely used artificial dataset in change detection literature is the Stagger, which was introduced by Schlimmer [34] and used in [19, 26, 39]. Another popular dataset is the Moving Hyperplane, used in [22, 26, 38]. These types of artificial datasets allow control over type, rate, concept drift, recurrence,

appearance of noise, and existing irrelevant attributes. Their major downside is lack of scalability [36]. Some other used artificial datasets includes: Flight simulator [19], Webpage access [22], and Credit card fraud [38]. Real datasets, such as Electricity [19], were also investigated in some studies (e.g. [17]). The main disadvantage of a real dataset is the fact that abrupt changes happen very rarely. To address this shortcoming, abrupt changes should be introduced manually into the set.

For the purposes of this research, one real (UCIS [3]) and two artificial (KDD'99 [15] and NIMS [4]) datasets have been used. In both cases, concept shifts are introduced manually to facilitate change detection performance evaluation. Further details about these dataset are described in Section 5.1.

2.1.3 Change Detection Methods

Methods investigating the change detection problem generally take the form of a window-based approach (e.g. [17, 25, 33, 37, 40]) . There have been other methods, as well; for instance, Klinkenberg [24, 25] uses SVM for establishing a weight-based approach. He assumes that the importance of learning exemplars changes over time and lose significance in proportion to exemplar age. Thus, their effect upon the final decision gets reduced over time. Nonetheless, after applying this theory to text data, he concludes that using a window technique with adaptive size works better than a weighting scheme [24].

The *sliding window* is a technique of instance selection where the emphasis is on the most recent data rather than historical data. The term "Sliding" refers to the fact that the window moves forward on the stream. Prediction models based on the current window location attempt to predict the label of the upcoming flow in the data stream. These window sizes could be variable, as Widmer and Klinkenberg [25, 40] use adaptive window sizes. The possibility of using this technique was first introduced by Helmbold [20]. Babcock [7] states that the size of the window could be dependent on the time variable. As a result, all flows within specific time stamps are considered as a window regardless of their size. These methods apply on the stream, whereas Zhu [42] takes a behavioral ensemble approach in which different classifiers receive a different weighting. Both window-based techniques and behavioral ensemble

approach motivate the approach adopted in this research and are described in Chapter 4.

2.1.4 Streaming Algorithms

In offline algorithms, it is generally assumed that the data set is stationary cf., the "independent and identically distributed" assumption [19, 25, 39]. Thus, it is fair to partition the data set into independent training and test partitions i.e., the training partition is suitably representative of the underlying task. As soon as the task (i.e. process creating the data) is non-stationary, this will not hold true. Indeed, if the offline batch approach were adopted for non-stationary data the labels could even be explicitly contradictory. Conversely, under online algorithms, data is expected to arrive continuously. Thus, it is expected that the algorithm adapts to the new incoming data on a continuous basis. In some cases, e.g. [2, 3], an offline algorithm is supposed to update periodically, whereas in other studies , e.g.[1, 22, 38], a change detection method is used to reduce the cost of unnecessary periodical updates. In both cases, user feedback is needed in order to provide the necessary class label information.

Atwater [6] shows how classification on data streams could benefit from genetic programming (GP) concepts such as Pareto archiving. Specifically, Pareto archives provide a useful scheme for deciding which data points should be retrained from the sliding window for identifying particularly 'good' GP classifiers. The caveat being that for computational reasons the archive needs to be of a finite size, thus requiring the use of appropriate secondary selection heuristics when the archive limit is encountered [5]. Similarly, in this research, a GP classification framework has has been pursued in order to satisfy the goal of label free change detection. The Pareto GP framework and approach to change detection are explained in Chapters 3 and 4 respectively.

2.2 Literature Review

This research is focused on the concept of change detection as applied to GP models of classification. Thus rather than attempt to perform evolution on a continuous basis – something that we wish to decouple from on account of the labelling cost – we are

first interested in detecting when a change occurs. If we can do this without reference to labels, then we will only trigger retraining under specific circumstances.

As described in Section 2.1.2, change detection to date as assumed one of two generic approaches, either an entropy based measure as applied to the input stream followed by some form of threshold comparison (e.g., [23, 25, 37, 39]) or a behavioral analysis of a team (or ensemble) of classifiers (e.g., [42]) is assumed with a variance test (e.g., [18, 26, 28, 33, 38]).

Yang [41] assumes that changes have one of two forms: *virtual* concept drift caused by variation in data distribution and *real* concept drift caused by variation in context. In addition, changes could also be divided into two categories of concept drift(gradual change) and concept shift(abrupt change). Concept shift is a known issue in terms of dealing with data streams [36]. Moreover, recurring / cyclic reintroduction might also appear in real world streaming data; such as sensor networks, recommender systems, customer care or weather predictions. Algorithms that have memory of previously employed models could therefore potentially react without requiring the complete reconstruction of a previously identified solution.

Vorburger [37] assumes a common window-based approach but introduces a thresholding metric based on Shannon's entropy [35], that is able to detect both real and virtual concept shifts. The major issue of window-based approaches problem is the fact that it is purely reactive i.e., has no memory beyond the current window content. Conversely, ensemble learning methods [18, 22, 38, 39, 41] deal with this problem by keeping a history mechanism to cope with the recurring situations (e.g. [18] builds an ensemble learners from the existing classifiers pool). In this work, we will consider both scenarios (of detecting change on the stream and also incorporate an ensemble metaphor), the former providing the performance baseline that the latter should ideally maintain without the former's labelling requirement.

Various monographs have discussed evolutionary computation as applied under non-stationary or 'dynamic' environments[1] e.g., [14, 32]. In this work, we are particularly interested in decoupling the need for supplying class labels at the rate of the stream. From the perspective of applications in network traffic analysis, this is a very important requirement because the cost of providing labels is high. Conversely,

[1]Hereafter non-stationary and dynamic will be employed interchangeably.

under other applications often associated with non-stationary environments, such as trading agents in the financial services, there is no concept of a labelling cost. Instead, fitness is generally associated with maximizing the return on an investment [14] Thus, change detection may even be associated with the measurement of behavioral properties, such as loss of wealth.

Many authors have provided taxonomies for the types of variation that result in an environment being considered dynamic; for a survey see Chapter 3 in [14]. In the case of this thesis, we consider the case of an environment described in terms of traffic flow data. As such each exemplar from the stream summarizes a set of packets associated with the communication between the same pair of IP addresses. Thus, each exemplar potentially describes a behavior and requires independent classification. The goal of the classifier is to categorize the type of application / service associated with each network traffic flow. Tasks of this form are central to the management of computer networks and historically have been performed using port information or deep packet inspection. However, services are increasingly becoming hidden – care of the wide spread use of encryption or dynamic port allocation – making deep packet inspection or port based categorization ineffective. Previous works have demonstrated that machine learning methods can successfully classify encrypted traffic from flow data, but always assume the classical batch model of classification (e.g., [2]) and is therefore not appropriate for non-stationary data. The stream associated with this work is therefore representative of a task with abrupt random changes that in some cases involve revisiting previously encountered behaviors. The initial period of training necessary to construct the first set of champion classifiers is only conducted relative to a sample of data that is specific to a subset of the potential set of services, with other services potentially appearing later in the flow.

In this research change detection is done with respect to the class label space. Since change detection is relative to the input space, it might be possible to address this through the case of parameterized models as applied to the input (attribute) space. Similarly, Holst [21] has introduced a statistical anomaly detection component that aids opertors to identify deviations from normal as a sign of potential problems as early as possible. Previous research with GP in non-stationary environments often emphasize the characteristics of a specific application domain. Indeed, the only task

domain considered in [14] is that of financial services trading. In this case, evolution is undertaken on a continuous basis with the goal of tracking any change to the underlying dynamic. As noted above, this is fine as long as there is no labeling cost. However, we explicitly need such a decoupling in the network traffic classification task pursued here, and it is this characteristic that sets the principle distinction in the contribution of this work. More general recommendations made for applying evolutionary computation to non-stationary environments include [14]:

- Memory: the ability to refer to previously evolved champion solutions;
- Diversity: maintaining population diversity;
- decomposition: where modularity potentially leads to faster reconfiguration; and,
- Evolvability: the capacity to continuously develop new solutions.

Finally, several previous works illustrate the potential application of machine learning algorithms to the encrypted classification task. Arndt [3] studies three different machine learning algorithms to identify SSH traffic as a case scenario in network traffic analysis: C4.5 (supervised learning similar to [2]), K-means (semi-supervised learning similar to [16]) and Multi-Objective Genetic Algorithm (unsupervised-learning similar to [8] where a Multi-objective GA is used to select K value for K-means and clustering feature selection). In each case, however, a batch offline training framework was assumed.

Chapter 3

Co-evolutionary Multi-Objective GP(CMGP) Framework

A generic goal of *machine learning* is to identify patterns amongst data. The idea is to learn and form models of classification based on input data quickly and efficiently. This research is focused on classification under the evolutionary computation paradigm of machine learning. In classification algorithms, data(input) is given to the algorithm as a set of instances where each instance could be composed of several *features*.

In general, machine learning algorithms can be divided into two categories: *supervised* and *unsupervised*. In supervised learning, each instance is accompanied by a label or target output. In unsupervised learning, these target outputs are not provided. Thus, in the latter case, the focus is more on grouping the input data and building models based on similarities. Classification falls under supervised learning algorithms given the condition that the provided labels are discrete.

In the classification framework proposed by McIntyre [30], multiple individuals are evolved to cooperatively represent each class in the classification task. Regardless of the specific domain where these classification algorithms are applied, there are three important design issues that need to be addressed: representation, cost function, and credit assignment [9]. The representation refers to the form used to describe a candidate solution; where this establishes the possible set of models that can be described, or the *representation space*. The cost function provides performance information which in the case of a classification task is relative to some sample of data. Credit assignment determines how feedback from the cost function results in modifications to the representation and essentially boils down to answering how the exploration versus exploitation tradeoff is addressed.

Genetic Programming (GP) represents a form of evolutionary computation in which the representation takes the form of computer programs, in this case described in terms of a very simple "reduced instruction set" or register based transfer language.

One and two argument operands[1] are defined a priori to operate on a predefined set of general purpose registers. Two addressing modes are typically supported: Register–Register $R[x] \leftarrow R[x] < op > R[y]$ or Register–Attribute $R[x] \leftarrow R[x] < op > A[y]$; where $R[x]$ is a reference to general purpose register x, $< op >$ is an opcode, and $A[y]$ is a reference to attribute y. Credit assignment is addressed through *selection* and *variation* operators. Specifically, selection operators define who gets to 'reproduce' (parental selection) as well as who is replaced (survival bias) at any given training epoch. Selection therefore potentially makes use of information from the cost function. Variation operators define what material from a (pair of parents) is used to define an child individual. As such, variation operators act on the representation space. Typical examples include adding or deleting instructions (cf., mutation) or 'inheriting' different code fragments from a pair of parent individuals (cf., crossover). Credit assignment again occurs through the code inherited from parents. However, both selection and variation generally assume stochastic models of application. Specifically, the representation space is not ordered. As such gradient information is precluded. This makes it difficult to 'hill climb' to the 'nearest' local solution as the concept of locality is defined relative to all possible combinations of the search operators, relative to any two parents. Given that there are many candidate solutions (the population) present at any given training epoch, then conducting an exhaustive search for the single 'best' modification is generally not feasible. Instead, selection and variation operators are applied probabilistically, with survival, say, being determined deterministically i.e., a fixed number of the weakest individuals are always replaced at each generation; as in a breeding metaphor. All of the above represents the case of canonical GP [9]. In the following we elaborate on the specific approach taken to support task decomposition, multi-class classification and cardinality decoupling as embodied in the Coevolutionary Multi-objective GP framework(CMGP) [31]. Before doing so, however, we discuss the specific case of canonical GP as applied to the (batch offline) classification task.

[1]For example, single argument operands might take the form of trigonometric and logarithmic operators whereas two argument operands might be arithmetic or logical operators. In addition, tests on conditional statements might also be utilized, but are not considered here.

3.1 Canonical GP, the batch classification task and a case for CMGP

Applying GP to a classification task implies that for each GP individual, the program is executed for each training exemplar. This results in each exemplar being mapped from the input space to a 1-d output space, or number line (GPout). The goal of a GP individual is therefore to find a mapping that results in exemplars from one class appearing on (a continuous region of) half of the number line, and exemplars from the second class appearing on the other half of the number line.

Naturally, it is also useful to maximize the separation between exemplars associated with each region of GPout. To do so, a wrapper operator is employed. This takes the form of a sigmoid operator that maps points from one half of GPout to a suitable binary value, say 0; while the second region of GPout is always mapped to the second binary value, say 1. At the mid point of GPout there is a transition region in which the sigmoid operator smoothly switches between 0 and 1. Given that the labels for each class will be 0 or 1, then mapping points to the transition region always results in a non-zero error. Likewise, mapping an exemplar with a label of 1 (0) to the region of GPout that the sigmoid operator associates with outcomes of 0 (1) results in a maximum error. Figure 3.1 demonstrates a canonical GP classifier that uses sigmoid operator.

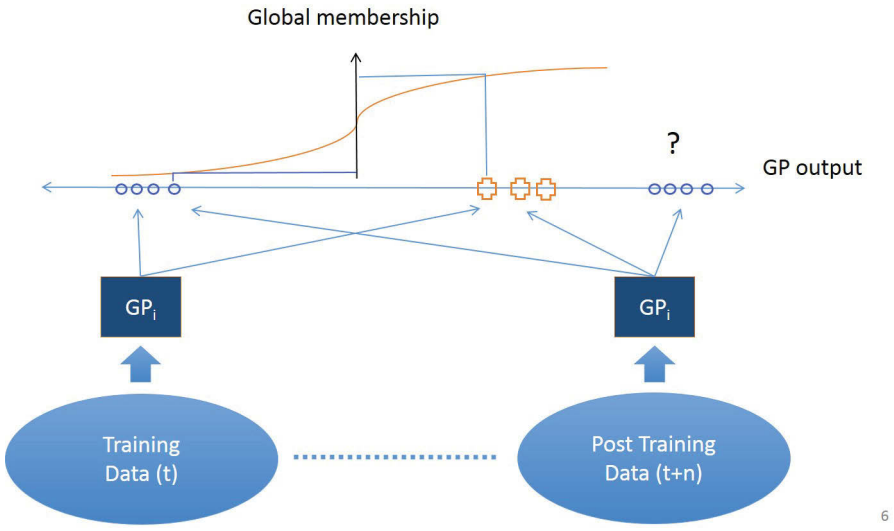


Figure 3.1: Canonical GP classifier with sigmoid operator

There are various well known limitations associated with this canonical framework for GP classification [31], summarized as follows:

1. Binary classification limitation: There are only two regions that a sigmoid operator may define under a minimum bias assumption. Attempting to divide GPout into more regions (to reflect more classes) forces an increasing number of assumptions. For example, under a three class scenario why should exemplars be a priori mapped to regions ordered as classes 0–1–2, as opposed to say regions ordered as 2–1–0 or 1–0–2? Such biases already exist in the binary scenario, however, it becomes increasingly difficult to avoid significant negative impacts – the learning task is made artificially more difficult – as the number of classes increase.

2. Monolithic classifiers: The canonical GP framework assumes that a single GP individual is responsible for classifying all the data. This is equivalent to requiring a single universal program neatly map all the data into class specific regions of GPout. Although theoretically possible, it may be much easier to find say, three programs, that map different subsets of the data set to their respective (binary) regions of GPout. In effect this is an argument for modular programming capabilities. The catch is that we cannot a priori define what subset of exemplars should be associated with each of the three different GP individuals.

3. Lack of robustness: When sigmoid operators are used to define the GPout number line up into two regions, and evolution successfully identifies such an individual i.e., exemplars from each class are successfully mapped to each of the binary regions of GPout, it transpires that the classifier can still fail in practice. Specifically, if the training data misses 'rare events' associated with say, a fault condition, there is no fail safe. The classifier will label the fault condition as one of the two classes. In a sense, the program has no way of indicating when it encounters something that it has never "seen before", even if all cases of such data were successfully mapped to a unique region of GPout.

4. Lack of scalability: Training of canonical GP models is performed against the entire training data set. This in itself might run to the tens if not hundreds

of thousands of exemplars. Even if we are looking at a binary classification task (separate skype from non-skype data), a data set of 200,000 exemplars, and GP with 500 individuals, evolved for 10,000 generations would require 1,000,000,000,000 program evaluations. Clearly this does not scale – especially when you note that the stochastic nature of GP also requires that multiple (say 50) runs are performed. Conversely, if training could be decoupled from the size of the training partition (cardinality) then the number of program evaluations could potentially be dramatically decreased. In effect this amounts to solving a dual learning task, identify and retain the training exemplars which distinguish between the performance of an archived set of GP classifiers.

Points 1 through 3 represent a requirement for reducing the constraints on the type of mapping that GP performs when mapping from input space to GPout. The CMGP algorithm addresses this assuming the following process with respect to each GP individual:

1. Map all training exemplars to GPout;
2. Cluster the 1-d distribution on GPout;
3. Select the most dense cluster, where this typically (although not necessarily) represents some subset of the training exemplars;
4. Use the selected subset of data to parameterize a Gaussian operator i.e., mean (μ) and variance (σ);
5. At this point class label data is introduced for the subset of points identified in Step 3. The point at the center of the Gaussian parameterized in Step 4 defines the class label this GP individual will assume. Error can now be estimated with respect to the subset of points. Only individuals that find subsets of point with the same class label will have a low error. This also implies that multiple performance properties need to be measured to avoid trivial / degenerate solutions. Thus, as well as error minimization, maximizing the subset of exemplars labelled is also rewarded as well as minimizing the overlap between the exemplars labelled by other GP individuals. The process of simultaneously

optimizing multiple objectives is referred to as Evolutionary Multi-objective Optimization (EMO) [13];

Addressing the scalability issue, as pointed out above, is approached by adopting a competitive coevolutionary framework referred to as Pareto archiving [11]. Thus, for each evolutionary cycle the training partition is stochastically sampled to construct a point population. Certain sampling biases are assumed such as equal representation of each class. The aforementioned process for evolving GP individuals with Gaussian operators is assumed to identify the fittest individuals, thus support the application of selection and variation operators. After this Pareto archiving is applied to identify the GP individuals for which "distinctions" are formed. That is to say, we only wish to retain exemplars that identify GP individuals as unique. Pareto archiving achieves this through the use of Pareto dominance, which implies that only the non-dominated GP individuals are retained relative to the sets of exemplars correctly classified. Thus, given a common subset of exemplars correctly classified, say set A, then an individual that classifies these plus an additional exemplar a would dominate all other GP individuals. Moreover, only exemplar a would need to be retained to correctly identify the GP individual as unique. Various potential pathologies can potentially exist – in particular forgetting, where this corresponds to a 'rock–paper–scissors' interaction; however, the significance of this in practice remains an open question.

3.2 CMGP and batch offline classification

The basis for the CMGP framework was established in the discussion of the preceding section, thus CMGP consists of two high level components: task decomposition (Gaussian wrapper operator and EMO fitness evaluation) and Pareto archiving (cardinality decoupling). The pseudocode proposed in Figure 3.2 [31] describes the *co-evolutionary Multi-objective GP (CMGP)* framework in more detail.

In total there are three components that make the CMGP framework of particular interest to the task of streaming data analysis in general:

I **Pareto archiving**: The cost of fitness evaluation is decoupled from the cardinality of the *training set*. Thus, fitness evaluation is performed relative to the

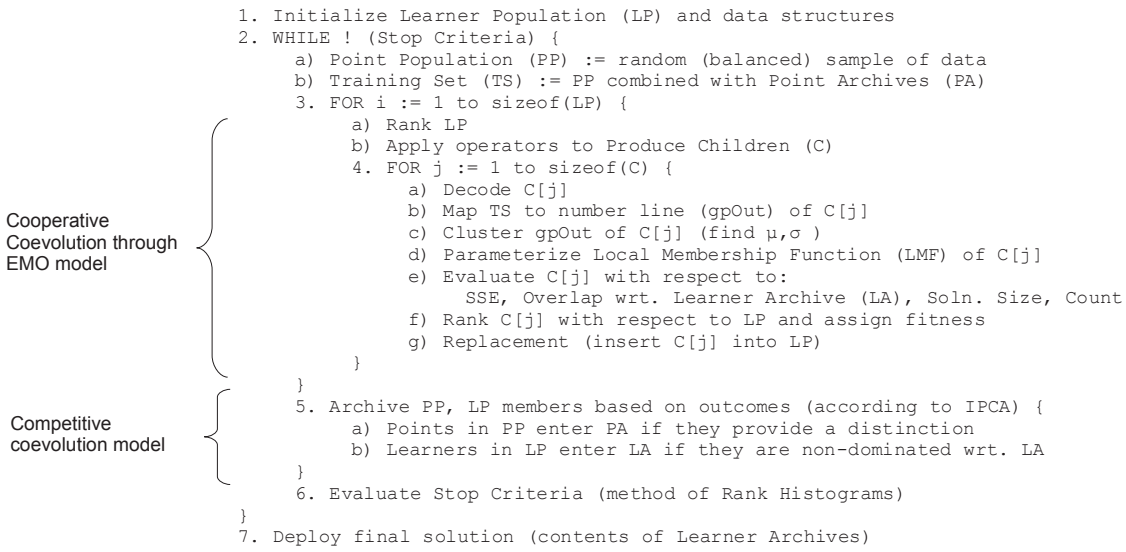


Figure 3.2: Pseudocode of CMGP framework [31]. Besides *Archiving*, other significant features are: 1) In *step2(a)* random balance sampling which gives equal presence of classes in the point population. 2) Gaussian wrapper operator divides the gpOut axis into a unique in class regio after applying a clustering method on the existing individuals (*steps 2.3.4.c and d*). 3) Class label is assigned to the individual and the fitness evaluated over multiple objectives for a subset of exemplars that were mapped to the Gaussian (*2.3.4.e*). Additionally, non-overlapping GP behavior is identified relative to the current Learner Archive(LA) content . 4) The best individuals from the point and learner population are identified using Pareto archiving *step 2.5*. 5) A Pareto rank histogram is used in *step 2.6* to evaluate stop criteria that is able to identify convergence class-by-class.

content of a *Point Population (PP)* and *Point Archive(PA)*, where each class has an independent Point Archive. The *Point Population* is sampled from the *training set (TS)*, typically while enforcing a bias to ensure equal representation of each class. A *Point Archive (PA)* represents the points (exemplars) that distinguish between different GP in a Pareto sense [11]. Pruning in *Point Archive* is done in order to keep it under specific memory boundary limits. Based on [29] each of the in-class or out-class occupies only 50 percent of the predefined *Point Archive (PA)* size which is set as the maximum. After reaching the maximum, the Euclidean distance of each point's feature space is calculated and the new points, replace the nearest old ones. Thus, from a streaming application perspective we assume that the newly replaced points make more useful distinctions than points encountered earlier. This *Point Archive (PA)* size also determines the window

Table 3.1: CMGP Archive and Population size parameter per class

Point Archive	Point Population	Learner Archive	Learner Population
50	50	50	50

size used in our entropy calculations which could be variable for each run but definitely will not go over pre-defined maximum *point Archive* size. Table 3.1 shows initial values for some of the parameters used in CMGP. Similar to *Point Archive* and *point population*, there is also a (GP) *learner population* and *learner archives*. The *learner archives* are GP individuals that are non-dominated relative to the content of the *point population* from a Pareto perspective. At any generation, the content of the *learner archives* therefore represents the current solution to the classification task.

II **Gaussian membership operator**: GP individuals utilize a *Gaussian membership operator* as opposed to the generally assumed sigmoid style operator. As shown in Figure 3.3, CMGP evolves the properties of the Gaussian, thus it is not necessary for GP individuals to represent an entire class. Instead multiple individuals are potentially evolved to represent different subsets of exemplars associated with the same class. Indeed, evolving solutions to multi-class problems from the same population is straightforward [30, 31]. The parameterization of the Gaussian is also evolved. Thus, following a forward pass through *Training Set (TS)*, each GP individual describes a distribution of points on *gpOut* – the number line representing the output from a GP individual. Such a distribution is potentially unique to each GP individual. Applying a clustering algorithm to the distribution identifies the single most significant cluster. It is with respect to this cluster that the Gaussian is parameterized and task decomposition occurs as a natural artifact of the evolutionary cycle. Moreover, from a streaming application perspective, decomposing the task potentially provides the basis for incremental replacement of different individuals as the content of the stream changes. Conversely if a monolithic / single champion GP individual had to represent the solution, any change in the underlying process generating the stream would likely require an entirely new GP individual to be evolved from scratch.

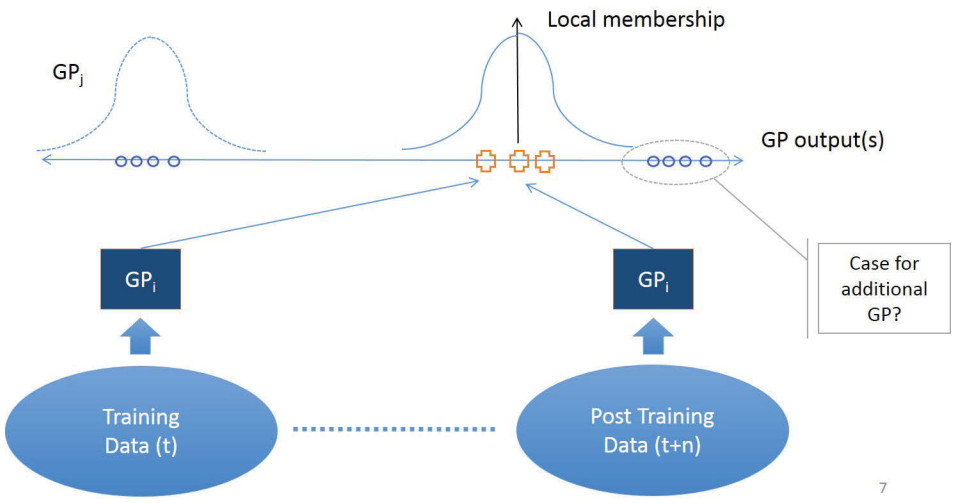


Figure 3.3: GP classifier with Gaussian operator

III **Evolutionary Multi-objective Optimization** (EMO): provides the basis for evaluating fitness and directing the application of variation operators. Assuming EMO during fitness evaluation means that properties characterizing the Gaussian membership function can be more clearly expressed. Thus, relative to the subset of points mapped to the Gaussian (of each GP individual) the following objective criteria are used to express fitness: sum square error; capacity for correctly labelling instances beyond the current capability of the *learner archives (LA)*; instruction count.

The sequential application of each of these components defines the overall architecture of CMGP. Figure 3.4 illustrates how the EMO and Pareto archiving properties relate to each other using the basic data structures assumed by CMGP. As a necessary precondition for EMO, the Gaussian parameterization has already taken place. Thus, at each generation the operation takes the following basic form.

1. Initialize the content of *point population* relative to *training set*.

2. Establish the output of each GP individual from *learner population* and *learner archive* using the exemplars defined by *point population* and *Point Archive*.

3. Evaluate fitness of the union of GP individuals from *learner population* and *learner archive* in an EMO setting. Any elitist EMO is potentially applicable

to this process, however PCGA [27] was specifically employed in this work.

4. Apply selection and variation operators to replace a fixed number of individuals from *learner population*.

5. The *Point Archive* and corresponding *learner archive* are now updated under a suitable Pareto archiving framework, in this case IPCA [11, 12]. This potentially results in new content for the *learner archive* and *Point Archive*. Specifically, the *learner archive (LA)* represent points (exemplars) that distinguish between the performance of GP individuals. Thus, loss of such a point would likely result in the loss in a learner (GP individual) that is non-dominated in the Pareto sense. This completes a generation of the CMGP algorithm – please see [30, 31] for further details.

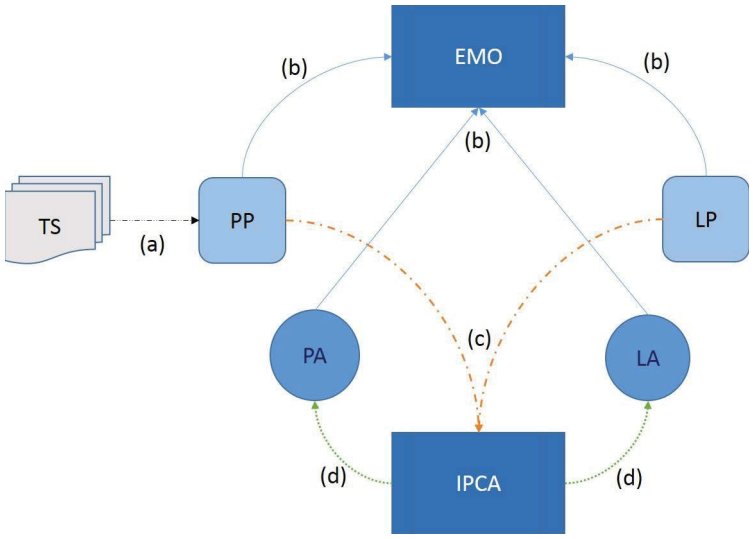


Figure 3.4: Overview of relation between EMO and Pareto archiving (IPCA) in original CMGP framework. Given a static *training set (TS)* a sampling is made to construct the *point population (PP)*. EMO is performed against the union of GP individuals sourced from *learner population (LP)* and *learner archive (LA)* with respect to the subset of training exemplars defined by the union of *point population (PP)* and *Point Archive(PA)*. This updates the content of *learner population (LP)*. A Pareto archiving step (IPCA) is then performed to update *Point Archive(PA)* and *learner archive (LA)*. This completes a single CMGP generation.

From a streaming data perspective, we are interested in two independent issues: 1) Does the content of the *Point Archive(PA)* provide a better basis for detecting

changes in the stream under the guise of an entropy 'filter' style approach? and; 2) Can changes in data from the stream be characterized without use of labels by measuring the degree of variation relative to *Learner Archive(LA)* content? Chapter 4 addresses each case by defining how the stream interfaces to the point population and then introduces two corresponding CMGP variants for satisfying each scenario.

Chapter 4

Change Detection in Streams

As introduced in Chapter 2, change detection can potentially be performed directly on the stream itself (or a form of filter) or with respect to changes detected in the behavior of the classifiers once they are initialized. In all cases, we assume that some initial labeled sample of data is available from the stream and used to seed the *Training Set* (Figure 3.4). A fixed number of generations is performed using the regular CMGP algorithm from Chapter 3. Three variations will then be considered:

1. **Baseline:** No further adaptation or change detection. Thus, the initial labeled sample defining the *training set* is used to construct the champion team of CMGP solutions after which there is no further modification. The stream content represents the test set as per other methods but with no adaption based on change.

2. **Entropy filter:** Change detection is performed independently from CMGP using the classical entropy based filter as applied to a pair of sliding windows on the stream. Such a process requires the stream to be labelled. If a change is detected, then the point population content can be updated, and CMGP will be retrained for a fixed number of cycles. Thus, CMGP labels the stream using the current *Learner Archive* content, with updates to *Learner Archive* being instigated by the entropy filter (Section 4.2).

3. **Entropy-PA filter :** The *Point Archive* data structure provides an alternative source for reference data used during the entropy sliding window. In all other respects the operation remains the same as the regular entropy filter. This modification is also described in Section 4.3.

4. **Behavioural CMGP :** The GP champions defined by the *Learner Archive* classify exemplars as they are encountered in the stream. We can therefore

characterize the difference between the 'confidence' of *Learner Archive* champions that are actually supplying the labels in a pair of sliding windows. Such a process is entirely independent of label information. Only when a significant difference appears in the confidence between pairs of windows are labels requested and then only for the window triggering the event. This process is described in Section 4.4.

4.1 Baseline

In order to have baseline results for comparison purposes, the CMGP framework was changed to accept stream data. This means that the framework now has been changed from an offline (batch mode) to an online algorithm, similar to [22, 38]. The training is done only once on the training set with no further learning cycles, because it is not sensitive to changes. This means that the content of the Learner archive at the end of training is thereafter used to classify the test partition of the stream. Thus, if the stream is non-stationary, the classification performance of the GP solution represents an "empirical lower bound" on the performance we could expect if training were permitted.

4.2 Entropy based change detection

A widely cited generic scheme for change detection on streaming data is based on the pairwise estimation of entropy between two sliding windows [37]. Data-sets are usually collected through monitoring data for a long period of time (*data stream*), which increases the possibility of having a variety of changes in the structure of data or even the introduction of new data over time. These changes might happen periodically or non-periodically. As described in Chapter 2, some of these changes are radical and happen in a switching way rather than gradual, and can be categorized as *concept shift change*. It is important to identify such changes so as to be able to trigger re-training cycles efficiently as needed through new incoming on-line data. Hence, with these changes, the performance and detection rate of the algorithm will not decrease. At the same time, as long as data is consistent and similar, there is no need to go through training cycles -thus eliminating what is usually a time-

and resource-consuming process for most classification algorithms. In addition, the more training performed, the greater the labelling requirement. Thus, reducing time and increasing performance are the primary reasons for using change detection mechanisms. Together, these features result in an adaptable, robust algorithm that is flexible (given the condition that the change detection calculation is efficient, rapid and dependable.)

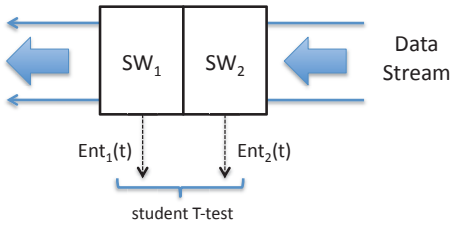


Figure 4.1: Pairwise sliding window (SW) configuration for entropy based change detection. $Ent_x(t)$ is the entropy estimated against the content of SW x at SW location t. Note in Equ. (4.2) $i \equiv t$

.

4.2.1 Entropy Filter

As a change detection method, Vorburger[37] introduces a new entropy measure (Equation (4.1))based on Shannon's entropy [35], which he declares as "reliable, noise-resistant, fast, and computationally efficient" [37].

$$H_i = \frac{1}{S}\sum_{s=1}^{S}\sum_{c=1}^{C}\sum_{b=1}^{B} -\omega_{iscb}(P_{old}log_2(P_{old}) + P_{new}log_2(P_{new})) \tag{4.1}$$

where, Hi represents entropy at time i. All data streams are composed of sequential flows, with each flow including S feature sets. There are a total number of C classes in the flows, and B is used as a discrete aggregation variable for the values of each feature stream (s). Although $B > 1$ allows us to identify changes for uniform distributions among each s feature set of data stream, we have assumed $B{=}2$ in order to simplify calculations (similar to Vorburger's assumption [37]). At time i, P is the probability of occurrence of an instance belonging to class c, with feature domain s in bin b.

$$P = \frac{\nu_{iscb}}{\lambda_i}$$

The weighting factor (ω) should be 1 in order to simplify calculations (see Vorburger [37]). In order to have final results in the range of [0 1], the entropy should be normalized by using distribution of classes across all data regardless of their features or bins as follows:

$$H_{i_{norm}} = \sum_{c=1}^{C} -\omega_{ic}(\frac{\nu_{ic}}{\lambda_i} log_2(\frac{\nu_{ic}}{\lambda_i})) \tag{4.2}$$

In this entropy calculation, at each time *i*, sliding window technique has been used; this technique involves having two equally sized and consecutive windows called $Ent_1(t)$ and $Ent_2(t)$ pointing to the most recent data (Figure 4.1). At each time of *i*, these two windows slide sequentially over the flows in a data stream. Absolute similarity in their content, returns an entropy result of 1; whereas no similarity returns a value of 0.

4.2.2 Automatic threshold setting

A tolerance threshold now needs to be introduced for defining how close the entropy needs to be before the content between pairs of sliding windows is declared as having changed. For automating this process, a statistical hypothesis test called *student's T-Test* was used:

$$t = \frac{\widehat{x_s} - \widehat{y_s}}{\sqrt{\frac{\sum(x_i - \widehat{x_s})^2 + \sum(y_i - \widehat{y_s})^2}{N_x + N_y - 2} \left(\frac{1}{N_x} + \frac{1}{N_y}\right)}} \tag{4.3}$$

This test is called to determine the statistical difference between the mean values of two arrays, x and y respectively. $\widehat{y_s}$ and $\widehat{x_s}$ are the corresponding mean values, and N_x and N_y are the cardinalities of the arrays. These arrays were each composed of entropy results from the data stream input. The *reference array* was collected after the last training cycle whereas the second array is constantly updated by the new incoming results. In other words, the reference array is only updated after identifying changes that triggers CMGP re-training and then stays the same until the next change is detected. Meanwhile, the content of the second array reflects updated content from the stream. According to entropy filter measures, 1 represents similarity and 0 absolute difference. Hence, if the *p-value* of T-test was less than ε, a re-training cycle would be initiated based on the determination that a change has happened in

the data stream. The initial value of ε is set as "0.9999" by default, but it could be changed depending on the dataset and the results driven by the baseline method (described in Section 4.1).

4.3 Entropy based change detection using PA

Although the sliding-window technique has been used extensively [23, 25, 39], there is a neglected factor in which data that was previously encountered and trained on, potentially results in a retriggering event. Hence, a false alarm of change detection is triggered. One solution to this problem could be keeping a history of learned shift changes and related classifiers [18]. However, it seems that we can make use of the CMGP Point Archive as the reference window. In the following we will therefore consider two cases, one in which the original pairwise sliding window approach is used to initiate retraining and one in which the Point Archive represents the 'reference' sliding window.

As mentioned previously, student *T-test* is employed to test for a significant change in entropy. When a *p-value* less than ε is returned the content of the second sliding window (SW_2) is merged into the *Training Set* of CMGP. Figure 4.1 summarizes the architecture. As per the original entropy filter, class labels are still necessary for the stream data. The process used for the merge assumes an exemplar age heuristic. Exemplars residing within the *Training Set* for longer are more likely to be replaced by an exemplar from SW_2. The process is summarized by Algorithm 1. In summary, the current *Training Set* is mapped to a roulette wheel with area inversely proportional to exemplar age (Step 3). Up to $|SW_2|$ individuals are stochastically selected (Step 4) and replaced by SW_2 content (Step 5). A fixed number of training epochs are then performed to update the Point and Learner archives ($T \div 4$) as per the original CMGP algorithm. In the following, we denote such a scheme as **Ent**.

One additional implication of assuming change detection based on Entropy estimated from consecutive pairs of sliding windows is that content of the reference window (SW_1, Figure 4.1) is always varying. Conversely, the original CMGP algorithm of Section 3 provides a definition for identifying a minimal subset of (previously encountered) exemplars that Pareto archiving has retained as supporting the identification of the champion team of classifiers (or distinctions). In the following, we

Algorithm 1 Merging content of sliding window with that of the Training Set (TS). X.age denotes the age count of exemplars from 'X'; SW_2 is the current sliding window content; X.$range$ is the age proportional distribution of individuals from 'X'; ts is a vector of the individuals from TS that will be replaced.

1. $SW_2.age = 1$;
2. $TS.age{+}{+}$;
3. $TS.range \leftarrow \text{roulette}(TS.age)$;
4. $\forall i \in SW_2 : ts.(i) = \text{select}(rnd(\cdot), TS.range)$;
5. $TS \leftarrow \text{replace}(ts,\ SW_2)$;

will denote **Ent-PA** as entropy based change detection performed with respect to the *Point Archive* content with SW_2 taking its content from the stream as before, Figure 4.2.

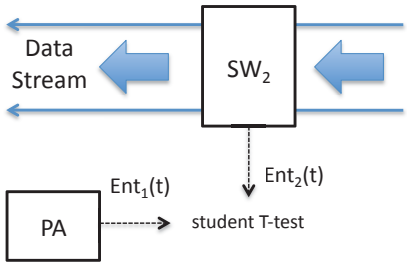


Figure 4.2: Entropy based change detection with *Point Archive(PA)* as the reference window and sliding window (SW_2)as the source. $Ent_1(t)$ denotes entropy with respect to the current content of the PA. $Ent_2(t)$ denotes entropy with respect to the current location of the SW in the stream.

Basically, the *Point Archive* of Figure 3.4 is a sample of exemplars that make distinctions in the classifiers retained by the corresponding Learner Archive. Hence, the Point Archive potentially reflects the reference set of most important exemplars that characterize the conditions under which the Learner Archive might change. CMGP's advantage over alternative memory like methods (e.g [18, 33, 39]) is the strength of the relationship between (GP) Learner archive content and Point Archive content instead of using weighting or classification error measures.

4.4 CMGP based change detection

Figure 4.3 summarizes the architecture assumed for performing change detection *without* prior labelling of the stream. Following the original CMGP framework of Section 3, the initial data content of TS is used to construct the initial team of GP champions or the LA content (as in the case of the filter methods of Ent and Ent-PA). As per the Ent-PA architecture, the team of GP champions (identified by the LA) provide labels for each exemplar under the current sliding window location SW_2 (Figure 4.3). However, CMGP does not create a single label from the team of GP champions. Instead, a winner-takes-all approach is assumed. Thus, the result of executing each GP champion for each exemplar is a degree of membership of the Gaussian associated with its mapping onto *gpOut*. Only the LA champion with maximum membership wins the right to suggest its label.

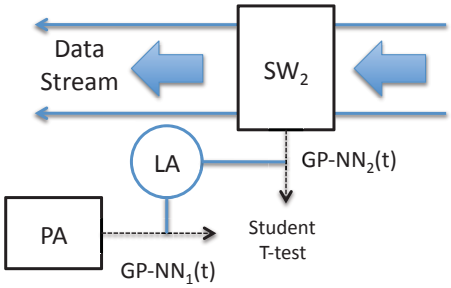


Figure 4.3: GP based behavioural characterization of a change without labels. $GP\text{-}NN_i$ [2]is the record of winning membership values over the content of PA ($i = 1$) or sliding window ($i = 2$). The LA defines a common set of GP champions.

Naturally, any change in the ratio of class labels between Point Archive(PA) and SW_2 is not necessarily indicative of a change in the underlying process describing the stream data. However, if a shift takes place in the process describing the data, we might expect this to be reflected in the confidence in the Gaussian membership associated with the winning LA champion. Thus, the Gaussian membership for PA is recorded and compared to that of SW_2. A student T-test is again applied to each and only when a significant difference is registered do we update the content of the point population (PP) using Algorithm 1 and perform a fixed number of training epochs ($T \div 4$) to update LA and PA relative to the new TA content. Hereafter this algorithm is denoted behavioural CMGP or **Bv-GP**.

Chapter 5

Experiments & Results

5.1 Datasets

For benchmarking purposes, three sets of datasets where used in this research: UCIS and NIMS from Arndt [3], as well as KDD'99 [15]. It is imposrtant to note that in all cases, separate datasets for initial training were used.

1. UCIS/NIMS

   Several researches [2, 3, 8, 16] have attempted to analyze traffic using binary identifications instead of payload, port or IP address. The motivation behind these attempts is, in part, forensic analysis where hackers can easily manipulate or conceal IP address and port numbers. In order to be able to segregate network traffic without using main characteristics, a concept called "flow" has been used. According to Arndt [3], "A flow is identified by a 5-tuple of the form (Source IP, Source Port, Destination IP, Destination Port, Transport Protocol) and consists of all the packets matching this 5-tuple during a given time frame". Flow attributes are then derived that describe basic statistical properties of packets associated with the same flow. In the case of this work up to 40 flow attributes are available care of the open source tool "NETMATE".[1]

   The training dataset used in this research is sampled from Dalhousie network traffic traces (UCIS). A random sample of SSL and non-SSL flows were used. The non-SSL flows consist of SKYPE, P2P, SSH and a variety of other protocols from port 80 (i.e HTTP) called OTHERS. The 40 features that are represented in these data-sets could explain the significant increase in the cost of computation of flows, since each of them needs extra data extraction (I/O overhead), memory and CPU cycles. Table 1 includes names of the example features used in this dataset.

[1] http://dan.arndt.ca/nims/calculating-flow-statistics-using-netmate/

Data based features	Time based features
total packets	minimum inter-arrival time
total bytes	mean inter-arrival time
minimum packet length	maximum inter-arrival time
mean packet length	standard deviation of inter-arrival times
maximum packet length	minimum active time
standard deviation of packet lengths	mean active time
average sub-flow packets	maximum active time
average sub-flow bytes	standard deviation of active times
push flag count	minimum idle time
urg flag count	maximum idle time
header length	mean idle times

Table 5.1: NIMS dataset features examples

NIMS is an artificially produced dataset that is processed exactly like UCIS. Although it contains similar network traffic data, underlying concept for producing this dataset was a virtual lab designed by Arndt [3] for collecting web-browsing data. Same statistical feature extraction through Netmate was also applied on network traces. For further information on both UCIS and NIMS dataset, refer to [3]. In this research, we have introduced and used NIMS data in combination with UCIS in order to test the robustness and effectiveness of the change detection method.

2. KDD

KDD CUP 99 (or KDD'99) is a dataset used for the Third International Knowledge Discovery and Data Mining Tools Competition [15].[2] It was supposed to be used for building an intrusion detector for computer networks that should have had the capability of identifying "normal" traffic traces from "ab-normal" or "attacks". Although, it contains 4 different types of network attacks (DoS, R2L, U2R, and Probing), only DoS and Probe attacks shown in Table 2 were used to create a flow as an input for this study. DoS itself is the result of three different processes: Neptune, Back, Smurf; whereas Probe can be the result of one of four different processes: Nmap, IPsweep, Portsweep or Satan.

[2]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

DOS	PROBE
Neptune	Nmap
Back	IPsweep
Smurf	Portsweep
	Satan

Table 5.2: KDD attack types

5.2 Empirical Evaluation

As explained in Section 5.1, we benchmark the baseline and three approaches of Section 4 for classification on streaming data (Ent, Ent-PA and Bv-GP) under data sets constructed from network flow data. The first dataset is UCIS where the basic goal of the classification task will be to label flows as either SSL or non-SSL, where SSL is an example of an encrypted protocol, in this case secure HTTP. What makes the task of particular interest from the perspective of change detection in streams is that SSL flows might consist of different services – such as online banking, Skype or p2p – and such services can appear as SSL or non-SSL. Previous research has demonstrated that machine learning trained under offline (batch) models of classification require training sets with tens of thousands of exemplars to provide acceptable classification performance [2]. In this work, the initial TS content is limited to 1,000 flows with 10,000 flows appearing in the stream. Moreover, the stream data appears in blocks of 2,000 flows. As shown in Figure 5.1, the underlying process describing the post training stream consists of blocks with non-SSL content sequenced as follows: skype-1, p2p-1, skype-2, skype-3, p2p-2; whereas SSL content is sampled uniformly throughout. The initial 1,000 flows used to establish our initial classifiers include SSL (class 1) and non-SSL content (skype and p2p). Hereafter this data set is denoted **SPS**.

Similar to the first dataset, the interest of the second data set is again on identifying SSL from non-SSL traffic. However, there are 3 classes in this dataset: SSL (class 0), HTTP (class 1) and Skype-P2P (class 2). The initial training set (TS) is also in the size of 1000 but it is composed solely from UCIS. In the test set, new data from NIMS is introduced to see whether the change detector method will treat them as previously seen data or as a new change. There are five blocks with the size of 3,000 flows appearing in the stream where new data from NIMS is introduced in block-2(HTTP) and block-3(SSL). The term *SPS-NIMS* will be used from now on

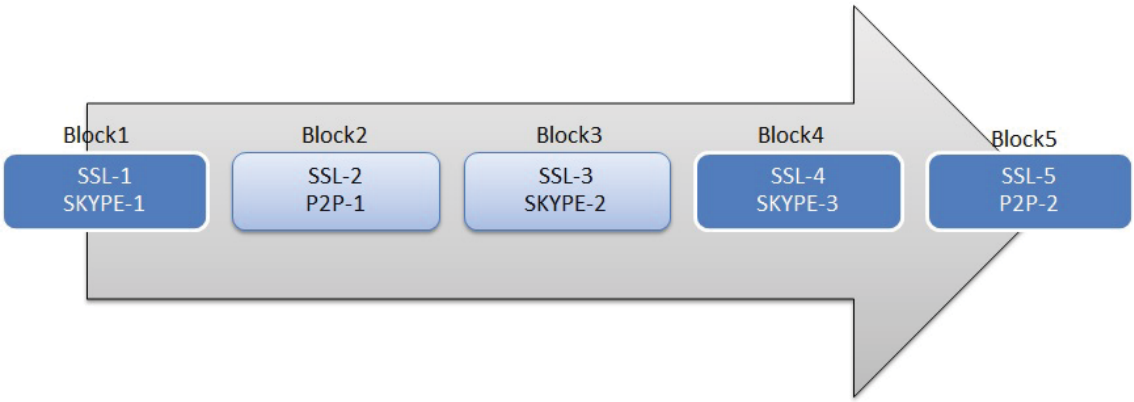


Figure 5.1: SPS data stream

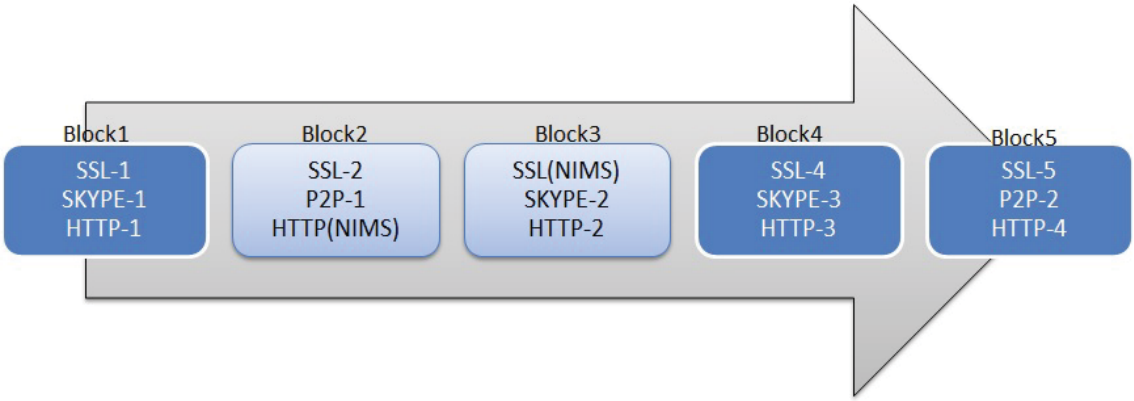


Figure 5.2: SPS-NIMS data stream

to refer to this dataset. Figure 5.2 shows the SPS-NIMS dataset as a block-based stream.

A third flow dataset is constructed from the KDD'99 contest data set,[3] again describing network traffic as flows. This represents a task in which normal behaviour needs to be distinguished from different forms of malicious behaviour. As mentioned in Section 5.1, we are specifically interested in distinguishing Denial of Service (DoS) and Probing behaviours from 'normal' behaviour. This again implies that the same label (abnormal as opposed to normal) is associated with multiple processes, and we are likely not to see all attack types during training. Moreover, normal behaviour itself is very difficult to characterize on account of the cyclic nature of computer network

[3]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

utilization. The (post training) stream itself is again composed from five distinct blocks and follows a sequence in which blocks representing the abnormal class label appear in the following order: DoS-1, Probe-1, DoS-2, DoS-3, Probe-2. All DoS and Probe blocks consist of 3,000 and 4,000 flows, respectively. Figure 5.3 shows the KDD dataset as a block-based stream.

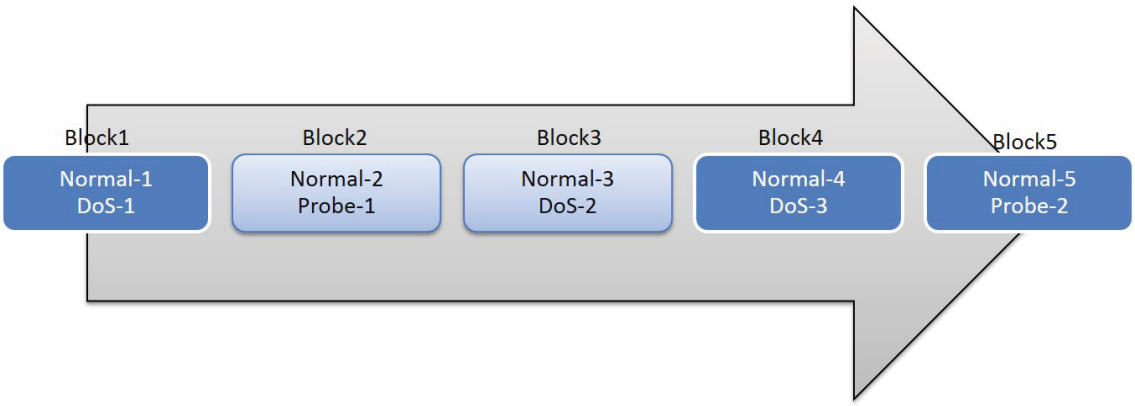


Figure 5.3: KDD data stream

Evaluation will take the form of detection rate as measured on each block of the stream. Thus, for the baseline algorithm, CMGP is trained on the initial TS content in batch mode and then the champion GP individuals from the LA (Figure 3.4) label the stream content. Evaluation of the remaining 'streaming enabled' algorithms implies that whenever a significant change is detected[4] then the point population is updated according to Algorithm 1 and a common number of generations are performed in order to update the learner and point archives (LA and PA). In all cases, classification performance is measured per block in terms of the average detection rate, or $DR = \frac{1}{|C|}\sum_{i \in C} DR(i)$ and $DR(i) = \frac{tp}{tp+fn}$ where tp and fn are the true positive and false negative counts as measured relative to the same block of the stream. Moreover, we are also interested in quantifying how much retraining is performed during the stream, hence we also report the number of re-triggering events. All versions share a common parameterization, as summarized by Table 5.3. The window step size (Δt) defines how much movement appears between consecutive shifts of the sliding windows SW_i. Given that there are two classes in each data set, then the overall PP and PA size

[4]All three streaming algorithms use a student T-test.

Table 5.3: Principle CMGP parameters

Max. PP size	50 per class
EPOCHS (T)	100
PA size	50 per class
Sliding window size (SW_i)	as PP size
Sliding window step size (Δt)	1
Change detection p-value	0.9999

matches that of the sliding sliding window.

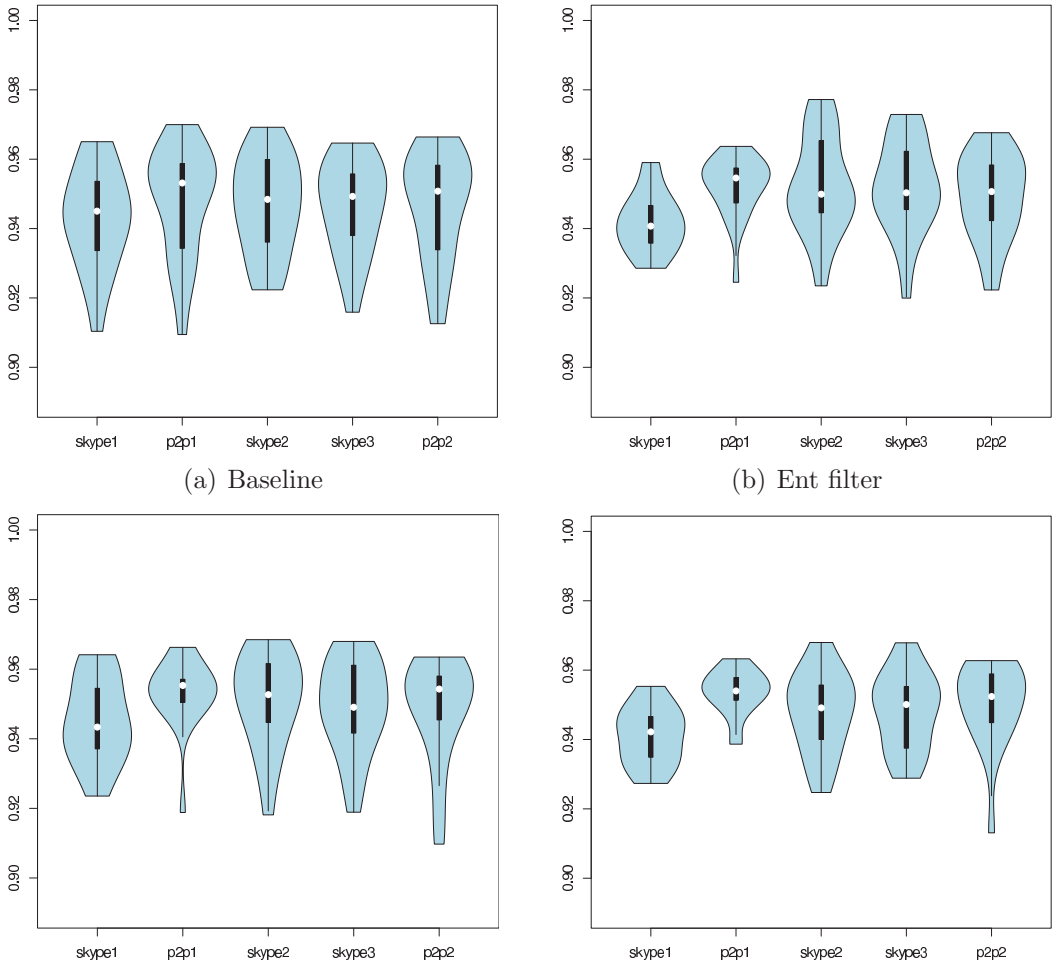
5.3 Results

Results will be summarized in terms of a dual violin–box plot for the 20 runs performed per algorithm. Box plots define 1st, 2nd (median) and 3rd quartile statistics (bold line) as well as limits of distribution within 1.5 times the 1st (3rd) quartile respectively (line line). The distribution of the violin provides a visual account of how normal or multimodal the results are. Claims regarding result significance will be expressed in terms of student T-test.

Figures 5.4 and 5.5 summarize the average detection rate and the number of re-trigger events for each block of the SPS stream (plus overall distribution in the case of the re-trigger metric).[5] From the perspective of detection rate, all variants provide similar performance; thus, there is a strong overlap in quartiles. However, it does appear that all three algorithms that incorporate online learning (Ent, Ent-PA and Bv-GP) provide tighter distributions than the baseline offline case (although there is no statistically significant difference between block-wise detection rates (Table 5.4)). Hence, Bv-GP – the label free case – is able to match the detection rate of the labelled scenarios.

In the case of the number of re-triggering events, there is significant variation in the overall rates of re-triggering between Bv-GP and the two filter methods (Figure 5.5(a)). Moreover, from the perspective of the block specific re-triggering rates (remainder of Figure 5.5) it is also apparent that the two filter methods emphasize

[5]Lack of variation in re-trigger counts results in the absence of a violin for that particular column in violin plots and cannot necessarily be concluded as an indicatore for the value of zero.



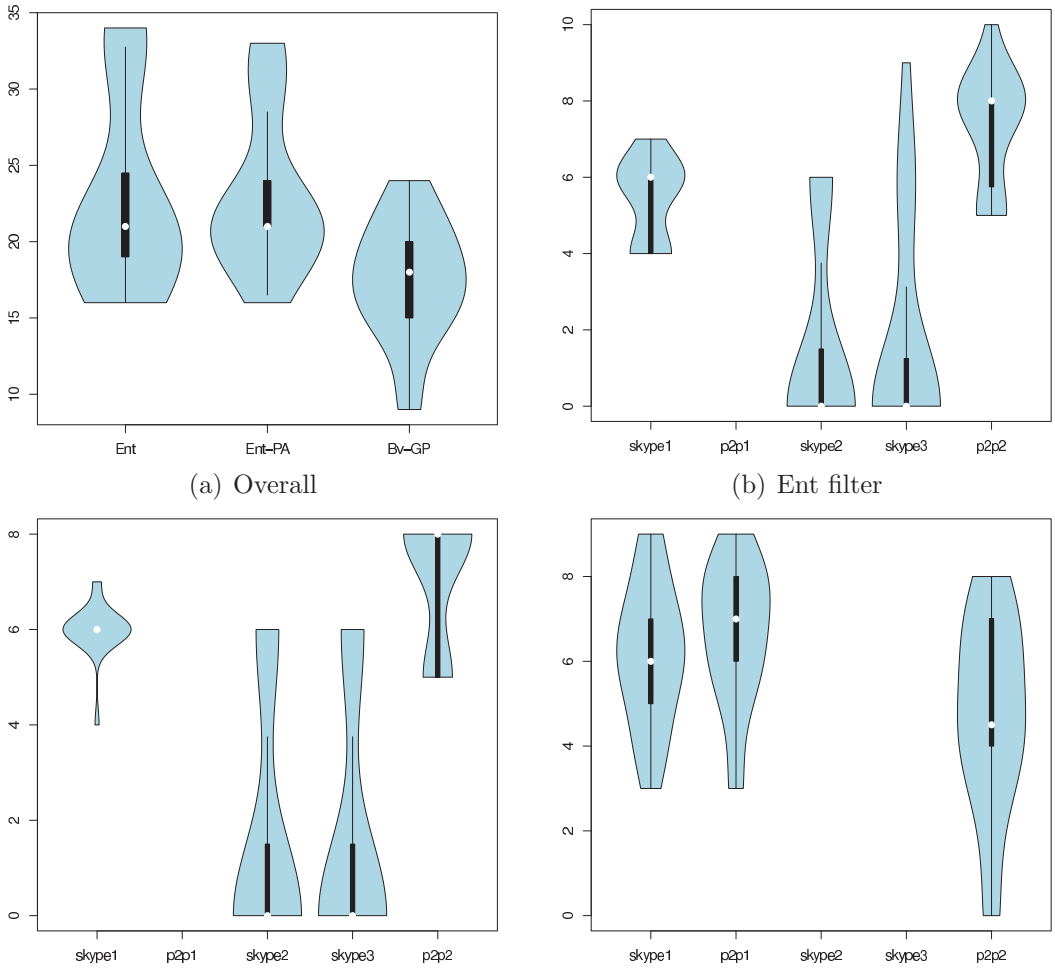
(a) Baseline

(b) Ent filter

(c) Ent-PA filter

(d) Bv-GP

Figure 5.4: Average detection rate over SPS stream following pre-training.



(a) Overall

(b) Ent filter

(c) Ent-PA filter

(d) Bv-GP

Figure 5.5: Re-trigger rates over SPS stream following pre-training. Subplot (a) denotes the overall combined number of re-trigger events across the length of the stream. The remaining subplots separate the re-trigger count across the different partitions of the stream.

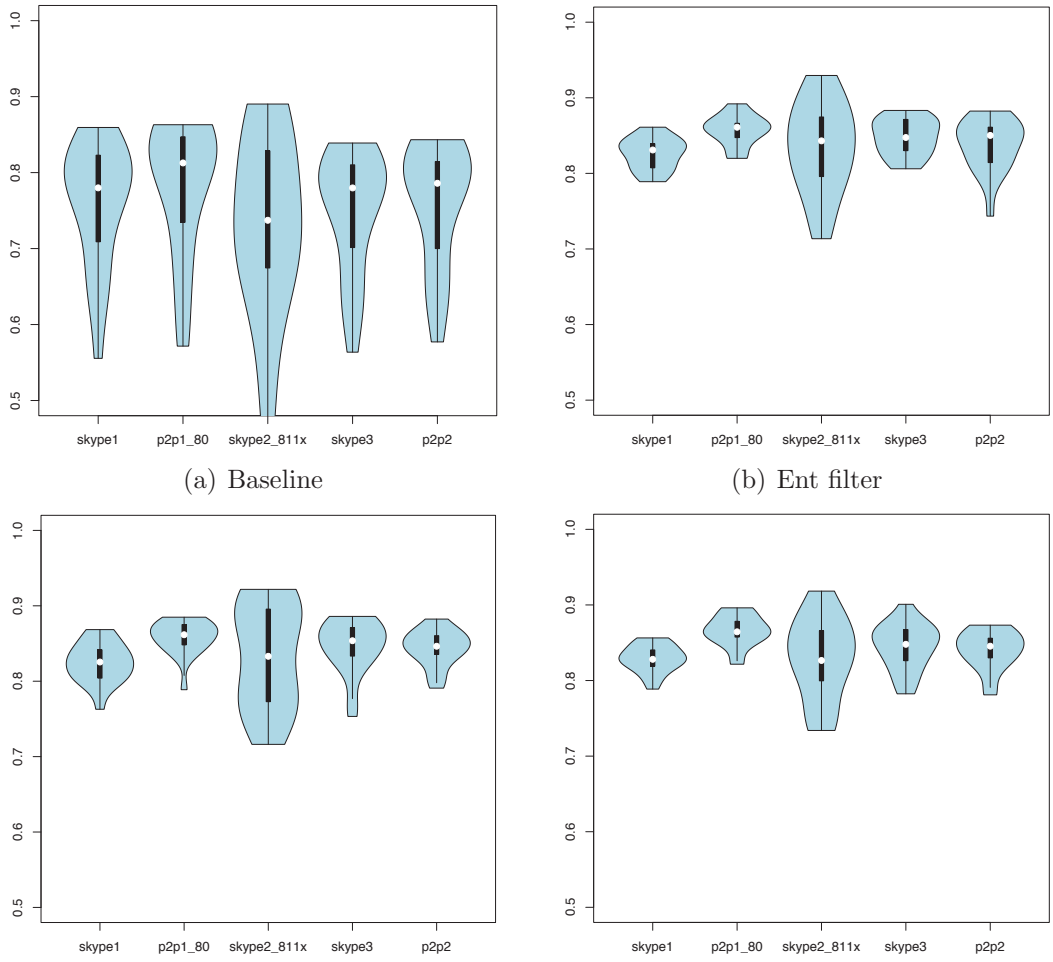
Table 5.4: Per block student T-test p-values for Bv-GP versus each alternate algorithm.

SPS data set.			
	Ent	Ent-PA	Baseline-GP
skype-1	0.75062	0.24764	0.6557
p2p-1	0.57394	0.93874	0.13478
skype-2	0.24422	0.37513	0.72949
skype-3	0.3237	0.79357	0.63402
p2p-2	0.72898	0.62541	0.37671
SPS-NIMS data set.			
	Ent	Ent-PA	Baseline-GP
block-1	0.68293	0.5775	0.001214
block-2	0.23624	0.34969	0.13478
block-3	0.80152	0.84195	0.003289
block-4	0.69664	0.81106	0.00005543
block-5	0.74945	0.65205	0.000149
KDD data set.			
	Ent	Ent-PA	Baseline-GP
DoS-1	0.45384	0.24802	0.011799
Probe-1	0.836	0.71556	0.00063401
DoS-2	0.403	0.29116	0.079582
DoS-3	0.67294	0.84723	0.0024116
Probe-2	0.21864	0.48974	7.5623e-06

retraining on skype-1 and p2p-2 while frequently ignoring the transition into p2p-1 and skype-2. Conversely, Bv-GP re-triggers on both initial instances of skype and p2p, then ignores the second and third instances of skype, and finally re-triggering on the last block of p2p.

In the SPS-NIMS dataset (Figures 5.6 and 5.7), the distribution of detection rate using the Bv-GP change detection method matches that of the filter methods and returns a statistically significant improvement over the baseline (Table 5.4). As previously known, SSL represents a separate class from other protocols and in block-3 (skype2_811x), new SSL data is introduced in the stream. However, Bv-GP retains a tight detection rate variance even though it does not make use of label information. In terms of trigger counts (Figure 5.7), Ent seems to be ignoring the introduction of the new data in blocks 2 and 3 , whereas Ent-PA appears to be ignore the new data in block 3. In comparison Bv-GP returns most of its re-triggering during the first three blocks and has a wide variance on the last block. As a result, Bv-GP continues to match the performance of the labelled streaming variants (Ent filter and Ent-PA filter) and provides significant improvements over the Baseline for 4 of the 5 blocks (Table 5.4).

As per the SPS-NIMS data set, the detection rates for the three online algorithms under the KDD data set (Figure 5.8) are now statistically independent from the (offline) baseline method (Table 5.4). The overall and block-wise distribution of re-trigger events however, are again rather different (Figure 5.9). Both of the filter methods (Ent and Ent-PA) follow a similar pattern for re-triggering training. Conversely, Bv-GP did not trigger retraining on the 'Probe-1' block in 50% of the runs. Moreover, there appears to be an incremental decrease in the instance of retraining as the number of blocks increases. This appears to indicate that the content of 'Probe-2' is significantly different from 'Probe-1'. Most agreement in re-trigger activity across all algorithms appears with respect to Probe-1 i.e., a low amount of retriggering.



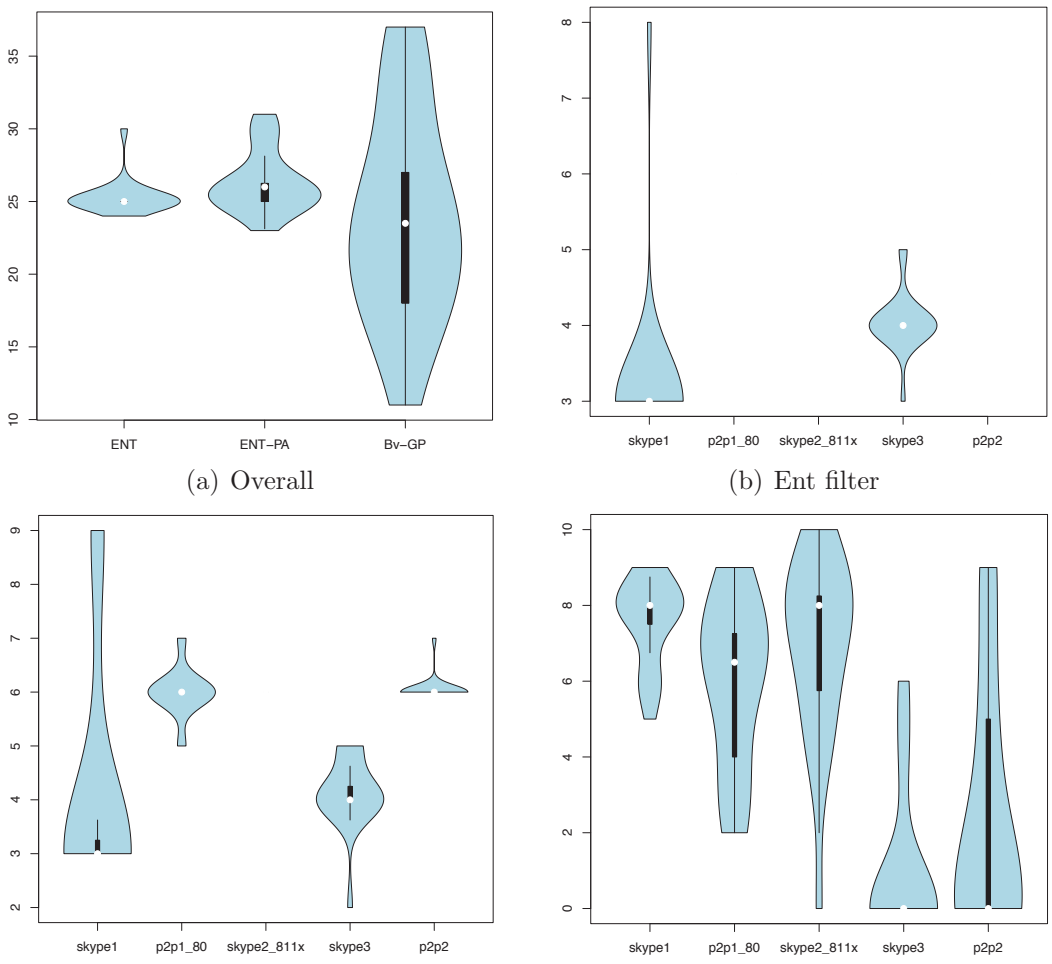
(a) Baseline

(b) Ent filter

(c) Ent-PA filter

(d) Bv-GP

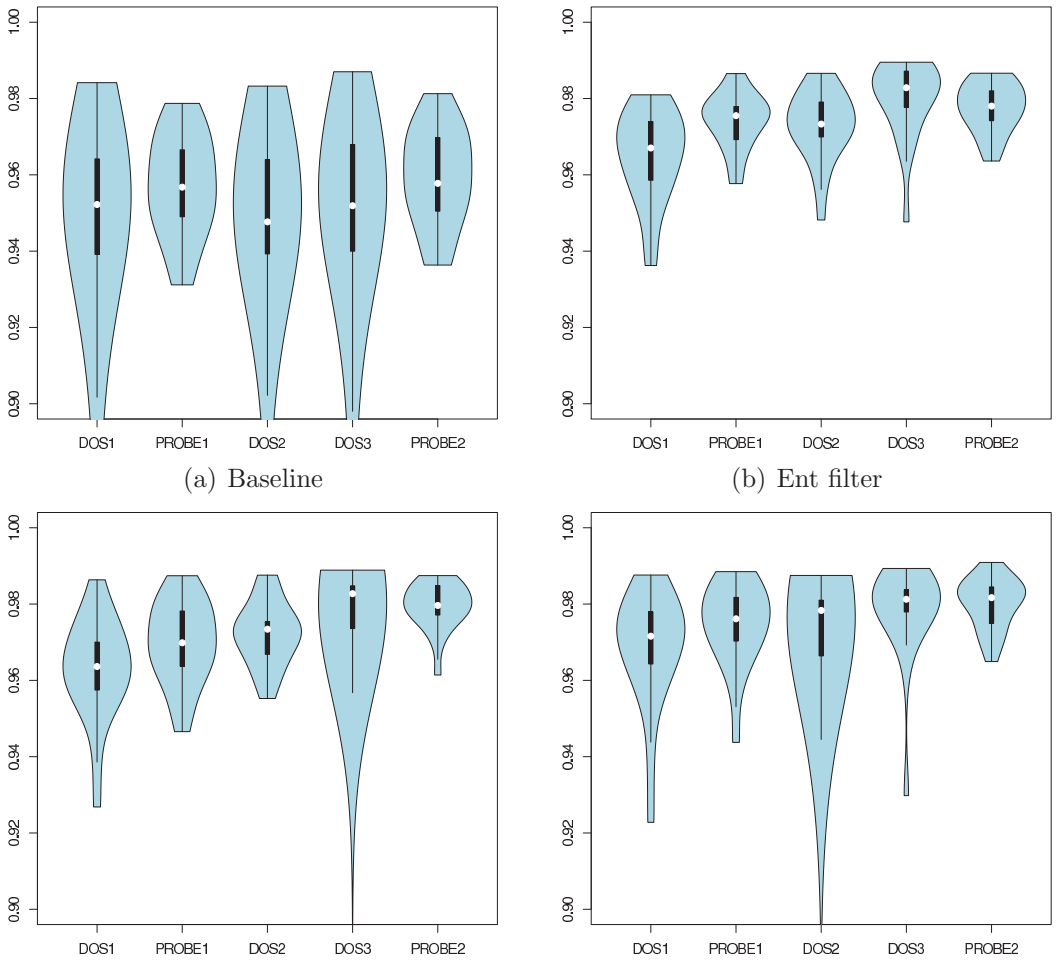
Figure 5.6: Average detection rate over SPS-NIMS stream following pre-training.



(a) Overall (b) Ent filter

(c) Ent-PA filter (d) Bv-GP

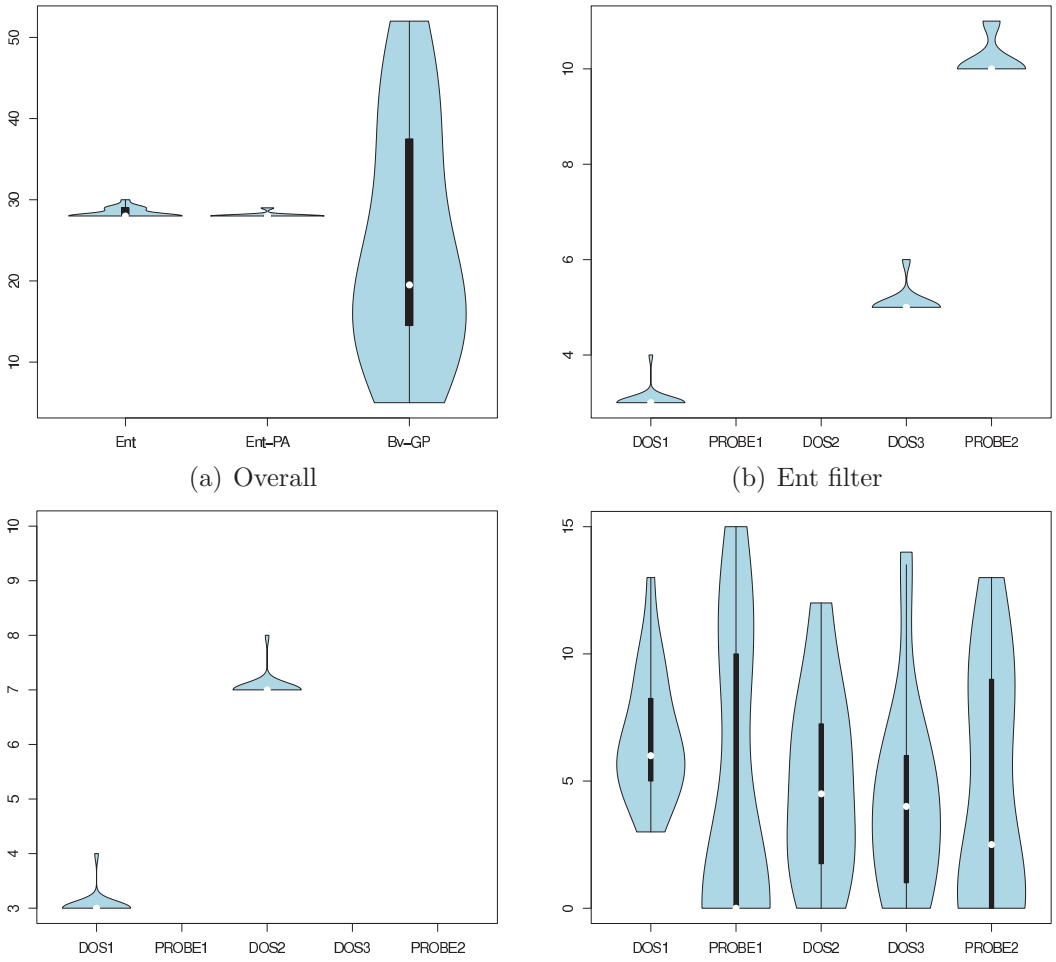
Figure 5.7: Re-trigger rates over SPS-NIMS stream following pre-training. Subplot (a) denotes the overall combined number of re-trigger events across the length of the stream. The remaining subplots separate the re-trigger count across the different partitions of the stream.



(a) Baseline (b) Ent filter

(c) Ent-PA filter (d) Bv-GP

Figure 5.8: Average detection rate over KDD stream following pre-training.



(a) Overall (b) Ent filter

(c) Ent-PA filter (d) Bv-GP

Figure 5.9: Re-trigger rates over SPS stream following pre-training. Subplot (a) denotes the overall combined number of re-trigger events across the length of the stream. The remaining subplots separate the re-trigger count across the different partitions of the stream.

Chapter 6

Conclusion & Future Work

In this thesis, the task of change detection has been considered from the perspective of an online streaming context. The classical approach to this task assumes an entropy pairwise sliding window approach in which class-wise entropy is used to trigger re-training events (Ent and Ent-PA). The principle drawback of such an approach is the need for label information that is the stream has to be pre-labelled. From the perspective of streaming applications, such as network traffic analysis, this assumption would be prohibitively expensive. We demonstrate that by instead concentrating on the *behaviour* of the team of pre-trained champion GP individuals responsible for classifying the stream data, change detection can be performed without reference to any label information (bv-GP). Thus, only when changes are detected is label information requested. We demonstrate that bv-GP matches the stream detection rates of the classical approach without requiring the prior labelling of the stream itself. Moreover, the number of re-trigger events is frequently lower when using bv-GP, suggesting that the process for detecting changes is also more efficient / accurate.

Considering detecting changes without requiring labels, it should be noted that there is no ground truth. Previously seen attribute combination can be labeled later as different sets. Thus, it could potentialy result in thrashing of the archive content, learners and point archives(PA and LA), as described in [6].

It is envisaged that future research will extend the algorithm to the case of gradual as opposed to step changes in stream content. Currently changes in the datasets used were imposed manually for ease of measurement and comparison purposes. Moreover, the features used in the datasets should be investigated further to identify the most suitable composition in the case of UCIS (real world dataset). In addition, this research was limited to the datasets from network traffic analysis and could be expanded to other data streams in the field of, say financial markets or signal processing.

The idea of change detection without requiring labels was specifically implemented

on the CMGP framework. More percised investigation should be done on initial parameter settings of the CMGP framework. Also, important attributes and features that were used most frequently by CMGP should be studied thoroughly. Moreover, future research might consider similar algorithms in which *Pareto archiving* is employed but extend it to non Gaussian wrapper operators. Likewise, benchmarking against other non-evolutionary streaming algorithms for classification would be appropriate e.g., MOA [10]. One step further would be applying this technique on other machine learning algorithms such as unsupervised learning models.

In conclusion, applying a change detection method without requiring labels is a relatively new method for reducing human interaction with learning algorithms. Human interaction is needed in order to provide proper labels and this technique attempts to decrease the labelling requirement without impacting on the detection rate.

Bibliography

[1] H. Abdulsalam, D. B. Skillicorn, and P. Martin. *Knowledge and Data Engineering, IEEE Transactions on Classification Using Streaming Random Forests*, 23(1):22 –36, jan. 2011.

[2] R. Alshammari and A. N. Zincir-Heywood. Machine learning based encrypted traffic classification: identifying ssh and skype. In *Proceedings of the Second IEEE international conference on Computational intelligence for security and defense applications*, CISDA'09, pages 289–296, Piscataway, NJ, USA, 2009. IEEE Press.

[3] D. J. Arndt and A. N. Zincir-Heywood. A comparison of three machine learning techniques for encrypted network traffic analysis. In *Computational Intelligence for Security and Defense Applications (CISDA), 2011 IEEE Symposium on*, pages 107 –114, april 2011.

[4] D.J. Arndt. An Investigation of Using Machine Learning with Distribution Based Flow Features for Classifying SSL Encrypted Network Traffic. Master's thesis, Faculty of Computer Science, Dalhousie University, NS, Canada, 2012.

[5] A. Atwater and M. I. Heywood. Benchmarking Pareto archiving heuristics in the presence of concept drift: diversity versus age. In *ACM Genetic and Evolutionary Computation Conference*, pages 885–892, 2013.

[6] A. Atwater, M. I. Heywood, and A. N. Zincir-Heywood. GP under streaming data constraints: A case for Pareto archiving? In *ACM Genetic and Evolutionary Computation Conference*, pages 703–710, 2012.

[7] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '02, pages 633–634, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[8] C. Bacquet, A. N. Zincir-Heywood, and M. I. Heywood. An investigation of multi-objective genetic algorithms for encrypted traffic identification. In *CISIS'09*, pages 93–100, 2009.

[9] W. Banzhaf, F. D. Francone, R. E. Keller, and P. Nordin. *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[10] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. Moa: Massive online analysis. *J. Mach. Learn. Res.*, 11:1601–1604, August 2010.

[11] E. D. de Jong. A monotonic archive for pareto-coevolution. *Evolutionary Computation*, 15(1):61–94, 2007.

[12] E. D. de Jong and J. B. Pollack. The incremental Pareto-coevolution archive. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 3102 of *LNCS*, pages 525–536, 2004.

[13] K. Deb. *Multi-objective Optimization using Evolutionary Algorithms*. John Wiley and Sons, 2001.

[14] I. Dempsey, M. O'Neill, and A. Brabazon. *Foundations in Grammatical Evolution for Dynamic Environments*, volume 194 of *Studies in Computational Intelligence*. Springer, 2009.

[15] C. Elkan. Results of the kdd'99 classifier learning contest. *SIGKDD Explorations*, 1(2):63–64, 2000.

[16] J. Erman, A. Mahanti, M. F. Arlit, I. Cohen, and C. L. Williamson. Offline/realtime traffic classification using semi-supervised learning. *Perform. Eval.*, 64(9-12):1194–1213, 2007.

[17] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. In *In SBIA Brazilian Symposium on Artificial Intelligence*, pages 286–295. Springer Verlag, 2004.

[18] J. B. Gomes, E. Menasalvas, and P. A. C. Sousa. Learning recurring concepts from data streams with a context-aware ensemble. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pages 994–999, New York, NY, USA, 2011. ACM.

[19] M. B. Harries, C. Sammut, and K. Horn. Extracting hidden context. *Mach. Learn.*, 32(2):101–126, August 1998.

[20] D. P. Helmbold, P. M. Long, M. Li, and L. Valiant. Tracking drifting concepts by minimizing disagreements. volume 14, pages 27–45, Hingham, MA, USA, January 1994. Kluwer Academic Publishers.

[21] A. Holst, M. Bohlin, J. Ekman, O. Sellin, B. Lindstrm, and S. Larsen. Statistical anomaly detection for train fleets. *AI Magazine*, 34(1):33–42, 2013.

[22] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '01, pages 97–106, New York, NY, USA, 2001. ACM.

[23] D. Kifer, S. Ben-David, and J. Gehrke. Detecting change in data streams. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 180–191. VLDB Endowment, 2004.

[24] R. Klinkenberg. Concept drift and the importance of examples. In *Text Mining: Theoretical Aspects and Applications*, pages 55–77. Physica-Verlag, 2003.

[25] R. Klinkenberg. Learning drifting concepts: Example selection vs. example weighting. *Intell. Data Anal.*, 8(3):281–300, August 2004.

[26] J. Z. Kolter and M. A. Maloof. Dynamic weighted majority: A new ensemble method for tracking concept drift. pages 123–130, 2003.

[27] R. Kumar and P. Rockett. Improved sampling of the pareto front in multi-objective optimization by steady-state evolution. *Evolutionary Computation*, 10(3):283–314, 2002.

[28] L. I. Kuncheva. Classifier ensembles for changing environments. In *In Multiple Classifier Systems*, pages 1–15. Springer, 2004.

[29] M. Lemczyk and M. I. Heywood. Training binary gp classifiers efficiently: A pareto-coevolutionary approach. In *In European Conference on Genetic Programming, volume 4445 of LNCS*, pages 229–240, 2007.

[30] A. R. McIntyre. *Novelty detection + coevolution = automatic problem decomposition*. PhD thesis, Faculty of Computer Science, Dalhousie Univeristy, NS, Canada, 2007. http://web.cs.dal.ca/˜mheywood/Thesis/PhD.html.

[31] A. R. McIntyre and M. I. Heywood. Classification as clustering: A Pareto cooperative-competitive GP approach. *Evolutionary Computation*, 19(1):137–166, March 2011.

[32] R. W. Morrison. *Designing evolutionary algorithms for dynamic environments*. Natural Computing. Springer, 2004.

[33] M. Salganicoff. Tolerating concept and sampling shift in lazy learning usingprediction error context switching. *Artif. Intell. Rev.*, 11(1-5):133–155, February 1997.

[34] J. C. Schlimmer and Jr. R. H. Granger. Incremental learning from noisy data. *Mach. Learn.*, 1(3):317–354, March 1986.

[35] C. E. Shannon. A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1):3–55, January 2001.

[36] A. Tsymbal. The problem of concept drift: Definitions and related work. Technical report, Technical Report TCD-CS-2004-15. Trinity College Dublin, 2004.

[37] P. Vorburger and A. Bernstein. Entropy-based concept shift detection. In *Proceedings of the Sixth International Conference on Data Mining*, ICDM '06, pages 1113–1118, Washington, DC, USA, 2006. IEEE Computer Society.

[38] H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 226–235, New York, NY, USA, 2003. ACM.

[39] G. Widmer and M. Kubat. Effective learning in dynamic environments by explicit context tracking. In *European Conference on Machine Learning*, pages 227–243. Springer-Verlag, 1993.

[40] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. In *Machine Learning*, pages 69–101, 1996.

[41] Y. Yang, X. Wu, and X. Zhu. Mining in anticipation for concept change: Proactive-reactive prediction in data streams. *Data Min. Knowl. Discov.*, 13(3):261–289, November 2006.

[42] X. Zhu, P. Zhang, X. Lin, and Y. Shi. Active learning from stream data using optimal weight classifier ensemble. *IEEE Transactions on Systems, Man, and Cybernetics–Part B*, 40(6):1607–1621, 2010.