Time-Series Event-Based Prediction: An Unsupervised Learning Framework Based on Genetic Programming

Ahmed Kattan

AI Real-world Application Lab, Um Al Qura University, Saudi Arabia

Shaheen Fatima

Department of Computer Science, Loughborough University, Loughborough, UK

Muhammad Arif

AI Real-world Application Lab, Um Al Qura University, Saudi Arabia

Abstract

In this paper, we propose an unsupervised learning framework based on Genetic Programming (GP) to predict the position of any particular target event (defined by the user) in a time-series. GP is used to automatically build a library of candidate temporal features. The proposed framework receives a training set $S = \{(\mathbf{V_a})|\mathbf{a} = \mathbf{0}...\mathbf{n}\}$, where each V_a is a time-series vector such that $\forall V_a \in S$, $V_a = \{(\mathbf{x_t})|\mathbf{t} = \mathbf{0}...\mathbf{t_{max}}\}$ where t_{max} is the size of the time-series. All $V_a \in S$ are assumed to be generated from the same environment. The proposed framework uses a divide-and-conquer strategy for the training phase. The training process of the proposed framework works as follow. The user specifies the target event that needs to be predicted (e.g., *Highest value, Second Highest value,* ..., etc.). Then, the framework classifies the training samples into different $Bins$, where $Bins = \{(\mathbf{b_i})|\mathbf{i} = \mathbf{0}...\mathbf{t_{max}}\}$, based on the time-slot t of the target event in each V_a training sample. Each $b_i \in Bins$ will contain a subset of S. For each b_i, the proposed framework further classifies its samples into statistically independent clusters. To achieve this, each b_i is treated as an independent problem where GP is used to evolve programs to extract statistical features from each b_i's members and classify them into different clusters using the *K-Means* algorithm. At the end of the training process, GP is used to build an 'event detector' that receives an unseen time-series and predicts the time-slot where the target event is expected to occur. Empirical evidence on artificially generated data and real-world data shows that the proposed framework significantly outperforms standard Radial Basis Function Networks, standard GP system, Gaussian Process regression, Linear regression, and Polynomial Regression.

Keywords: Unsupervised Learning, Genetic Programming, Time-series, K-means,

Email addresses: Ajkattan@uqu.edu.sa (Ahmed Kattan), S.S.Fatima@lboro.ac.uk (Shaheen Fatima), Mahamid@uqu.edu.sa (Muhammad Arif)

Prediction, Event detection.

1. Introduction

A time-series is a sequence of data points, measured typically at successive time instants spaced at equidistant time intervals. Usually, time-series data have a natural temporal ordering, which makes time-series analysis distinct from other common data analysis problems, in which there is no natural ordering of the observations. In many real-world applications, a vector V of observations $\{x_0, x_1, ..., x_{t_{max}}\}$ collected from equidistant time periods maintains some form of salient characteristics that can be exploited to predict the near future. Although time-series analysis algorithms may use solid mathematical formulas or complex statistical models, their predictions are entirely limited to the available historical data. Thus, no matter how accurate these algorithms tend to be on training data, they cannot guarantee a 100% correct prediction of the future. For this reason, time-series prediction can be seen as conditional statements of the form that "if such-and-such behaviour continues in the future, then so and so may happen..." [7].

Generally, time-series analysis is divided into two categories; A) Forecasting algorithms in which the aim is to predict the value x at time $t + 1$ given that sufficient historical data points are available, and B) Discovering events in a time-series. Discovering an event means to detect unusual variations in the time-series pattern and label them as rare events. An event in a time-series is defined as *"the occurrence of a variation in values over a time span that is of particular interest to a user"* [37]. The focus of this paper is on time-series events detection. Generally, work on time-series event-based detection is divided into two main categories. The first category is based on extract rule sets from the time-series and correlate them with particular events using machine learning algorithms (e.g., see [36]). The disadvantage of these techniques is that they are suitable only when rules for determining the occurrence of an event are clear and well understood. The second category is to detect changes in the flow of the time-series values and label these changes as events (e.g., see [37]). The underlying assumption of these models is that it is possible to mathematically model a time-series to detect unusual variations. The advantage of this approach is that it requires no previous knowledge of the problem domain. However, its main disadvantage is that it looks at the time-series from only one dimension, assuming events are correlated by the past behaviour in the time-series itself and ignoring the fact that other variables may cause an event. Another disadvantage is that it defines events based on time-series variations and prevents the user from defining a particular event of interest.

For the purpose of this work, we consider an event to be the occurrence of an occasion defined by the user. For example, given a time-series vector $V = \{(\mathbf{x_t})|\mathbf{t} = \mathbf{0}...\mathbf{t_{max}}\}$, a user may sometimes be interested in knowing when the highest point (i.e., max x_j for $0 \leq j \leq t_{max}$) is likely to occur (t_{max} is the size of the time-series). In general, the user may be interested in knowing when the n^{th} point will occur (e.g., highest point, second highest, or the lowest point in an unseen time-series), depending on the problem domain.

The contributions of this paper are twofold:

1. We propose an unsupervised learning framework based on GP to predict the position of any particular target event (defined by the user) in an unseen time-series.
2. Unlike other time-series-event based detectors, the proposed framework learns the behaviour of the environment that generates the time-series itself and uses this knowledge to predict when a target event is likely to occur in an unseen time-series.

The proposed framework receives training examples of historical time-series vectors generated from the same environment and uses GP to automatically build a library of candidate temporal features. In this paper we will use the term "*behaviour*" to refer to statistical features. Thus, for example, as illustrated in Figure 1, two time-series V_1 and V_2 generated from the same environment may not be identical but have similar behaviour in their trends of going up and down. In real-world applications, the environment can be anything including, but not limited to, stock markets, buyer-seller negotiations, or prices of oil, gas, or electricity in international markets.

The proposed framework works as follows. The examples of the training set are first put in different bins based on the exact time (in $[0, t_{max}]$) at which the event of interest happens. All bins are considered independent learning problems, and the next goal is to partition each bin into clusters of time-series of similar statistical features using GP and the K-Means algorithm. As new unseen time-series comes in (sequentially, one point at a time), the system measures the similarity between the new unseen time-series and clusters that have formed in the training phase. The closest cluster among all clusters of all bins is computed, and the algorithm returns the time of occurrence of the event as its prediction (more on this in Section 3). One advantage of the proposed framework is that it allows the user to define a particular target event of interest. Another advantage of this framework is that it requires no previous knowledge of the problem domain in order to predict events. As will be shown in the experiments section, this approach is experimented with, first on artificial data for the sake of understanding the behaviour of the method, then on real-world data from Google Trends service, reporting frequencies of use of keywords in Google searches. The results of the proposed method are better than those of standard Radial Basis Function Networks, standard GP system, Gaussian Process regression, Linear regression, and Polynomial Regression.

The proposed framework has many potential applications. For example, in economics, a monopsony is a market form in which only one buyer faces many sellers (e.g., for military equipment, contracts are limited to governments) [20]. Each seller makes different offers to the buyer in different time-slots based on their true valuation of the deal. The buyer needs to know the best time to accept an offer before it increases and the buyer loses the opportunity to maximise his/her savings. The buyer cannot recall offers from the past because the seller's interests and true valuation change over time. Another example is multi-round online auctions with limited time steps [34]. Here, if the buyer stores historical data about the sellers' behaviour, he/she can predicts whether the seller's bids will win on item, the seller will re-list the same item in the next round with a lower price, or he/she should bid in the current round. Also, as we will see in the experiments section, the proposed framework can be used to anal-

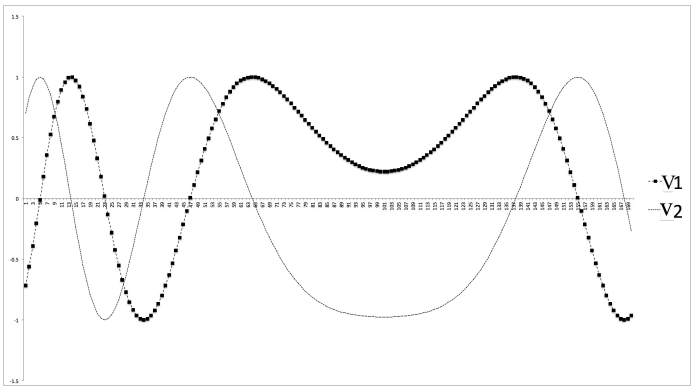

Figure 1: V_1 and V_2 not identical but have similar behaviour.

yse time-series data regarding keyword searches on the Internet and predict the next peak to assist marketing managers in deciding the best time to release their digital marketing campaigns. In addition to the examples mentioned above, the framework can assist market makers in understanding the behaviour of the player so as to design better market rules. Hence, understanding the buyer-seller relation is very useful for situations when governments decide to intervene in the stock markets (or any other market) during a crisis. In order to minimise the impact of a crisis, governments sometimes opt for injecting cash through buying shares from different stocks. However, such acts may not induce the right reaction when the correct rules of engagement are not properly set up and followed. By carefully studying the players' behaviour gain in such a scenario, governmental efforts towards salvaging markets during panic periods may be more effective.

The remaining of this paper is organised as follows. In the next section, we discuss related work and highlight the difference between our framework and other existing algorithms. In Section 3, we provide a comprehensive description of the framework. This is followed by experimental results and analysis in Sections 4 and 5, respectively. Finally, conclusions and possible future work is provided in Section 6.

2. Related Work

2.1. GP for Time-series Forecasting

Time-series forecasting algorithms can be divided into two main categories, *A)* algorithms to forecast univariate time-series, and *B)* algorithms to forecast multivariate time-series [35]. The difference is that the former assumes that the future behaviour of the time-series is affected by its past, for example, say, sales are affected by sales levels in previous periods. However, the latter assumes that other variables impact the time-series behaviour, for example, sales are affected by marketing budget. It is possible that a forecasting method could combine more than one of the above approaches, as, when an algorithm hybridises univariate with multivariate forecasts to adapt its performance

based on the patterns [7]. Researchers are trying to develop accurate forecasting models. Typically, standard forecasting algorithms are based on statistical models [5]. Most statistical forecasting methods assume that the time-series can be rendered as approximately stationary through the use of mathematical transformations [1]. For example, Nogales and Conejo in [23] proposed a transfer function model to predict electricity prices based on both past electricity prices and demands.

Although statistical models have dominated time-series forecasting research for a long time, researchers recently have realised the benefits of evolutionary computation and, in particular, GP in solving this problem. GP has been used for time-series prediction in several applications. The application of the GP in the time-series forecasting domain has been mainly used to induce a prediction model consisting of the best possible approximation of the stochastic process that could have generated by an observed time-series. Thus, the aim is to induce a model f that maps a vector V to the value x_{t+1}, where V is a vector of previously collected observations. These models are known as single-step predictors [1], and used to predict one step ahead of the time-series given that V contain sufficient historical information. GP has also been used for long-term forecasts [1], where iterated single-step prediction models are employed to forecast further steps beyond x_{i+1}. The idea is that the system uses its output predictions as inputs with the original time-series, assuming that the prediction is accurate enough to iteratively build-up the model . The predicted output is fed back as input for the next prediction while all other inputs are shifted back one place. The weakness of this mechanism that it is sensitive to its initial output values, as the prediction errors in the initial predictions may drift the model's predictions in subsequent iterations.

A recursive time-series prediction based on GP was proposed in [25]. In this work, the author used GP to evolve an approximation model that represents a simple prediction process. Then a second model is evolved through with a new training data, which are the prediction errors of the previous model. This process repeats for a fixed number of iterations. The final model is the weighted summation of all evolved models. The values of the weights are optimised using a GA engine.

Geum in [19] proposed a Genetic Recursive Regression (GRR). The idea was initiated from the weakness of the standard GP in forecasting stochastic time-series. Thus, the author applied the standard GP (forecasting the time-series as a standard symbolic regression) recursively to solve different parts of the time-series. To this end, an assumption was made that the dynamics of a time-series comprise a deterministic and a stochastic part. Using a 'zoom-in' metaphor the recursion process starts by zooming in on the difference of the residual time-series. By subtracting the model built by standard GP for the deterministic part from the original time-series, the stochastic part would be obtained as a residual time series. At the end of this recursion process, GRR ends with several evolved expressions. The algorithm compiles all expressions in a regression model as $a_0 + a_1 \cdot exp_1 + ... + a_n \cdot exp_2$. The numerical coefficients a_i are obtained by the least square method with regression model.

Using the same idea but with different implementation, Chen *et al.* in [8] proposed a hybrid evolutionary method to identify a system of ordinary differential equations (ODEs) and a particle swarm optimization (PSO) algorithm to fine tune the parameters of the additive tree models for the system of ordinary differential equations. In this work, the authors used tree-like representation where the root node is fixed at the

'+' operator and N number of branches each represents an equation. A population of additive trees is generated and standard search operators are applied to guide the evolutionary process. At some interval of generations, the best tree is selected to be further optimised using PSO. The PSO optimises the set of weights so the root node represents the weighted summation of all of its branches. The proposed algorithm was validated on a network traffic dataset and compared against the traffic prediction of gene expression programming (GEP) and GP system.

Tsang *et al.* in [33] proposed a decision support tool for financial forecasting based on a GP called Evolutionary Dynamic Data Investment Evaluator (EDDIE). In this system, GP is used to evolve decision trees. The evolved decision trees receive indicators that are collected from the finance literature. The contribution of EDDIE is effectively searching for combinations (interactions) of financial indicators. According to the authors, EDDIE can test users' hypotheses, for example, *"will any of these shares rise by r% within the next n days?"* so as to make buy/sell recommendations.

In [1], Agapitos *et. al.* used GP to predict winning bidding prices for customer offers in the Trading Agent Competition - Supply Chain Management (TAC SCM) game [1] were the problem of price prediction was presented as a time-series forecasting problem. The SCM environment was treated as a black-box system that generates time-series in terms of winning bids that reflect the variable macro-economic factors influencing the market at a particular point in time. GP has been supplied with a language to allow extraction of statistical features from the given time-series. Each function in the primitive set receives three inputs: *A)* the vector of time-series, *B)* the start location where the operator will be applied, and *C)* the end location where the operator will end. Thus, GP extracts different features from different intervals on the given time-series. The same author, in [2], proposed a seasonal forecasting temperature model by means of GP. GP was applied to learn an accurate, localised, long-term forecast of a temperature profile as part of the broader process of determining the appropriate pricing model for weather derivatives. The author explored two families of program representations for time-series modelling. The first is the standard GP technique,in which forecasting is solved as a standard symbolic regression problem. The second representation is based on long-term forecast (iterated one-step prediction) to resemble autoregressive (GP-AR) and autoregressive moving average (GP-ARMA) time-series models. The three models were used in a bagging framework together to build one generalised prediction model. The results show that ensemble learning of multi-model predictors enhanced their generalisation ability, unlike the use of single-model predictions. Standard GP (solving the problem as a symbolic regression) was unstable, producing some very poor-generalising models in some cases, while the performance of (GP-AR) and (GP-ARMA) models showed higher stability.

In [6], Cao *et al.* presented an approach to the evolutionary modelling problem of ordinary differential equations based on embedding a genetic algorithm (GA) into a GP system; the latter was employed to discover and optimise the structure of a model, while the former was employed to optimise its parameters.

[1]This game was introduced by Carnegie Mellon University and the Swedish Institute of Computer Science in 2003 [24].

In [10], a new multi-level GP approach is introduced for forecasting transport energy demand. In [38], the authors propose a forecast method which is a GP framework based on least square method.

2.2. Event-Based Detection

Time-series analysis models have been used to forecast the values of the future observations or to discover non-linear relationships among time-series variables to detect an event of interest. In this section, we will focus the literature review on time-series event-based prediction models, since they are more relevant to the work reported in this paper.

Time-series event detection is referred to as "change-point detection" in the statistics literature [12]. Guralnik and Srivastava [12] developed an approach for event detection in time-series data. The approach detects events by detecting the change in the model that describes the underlying data. The authors proposed two versions of their model: A) a batch version in which the model receives all time-series data before processing, and B) an incremental version in which the model processes new data-points one at a time.

Povinelli *et al.* [28] proposed a method for analysing time-series data that was inspired by data mining to predict future events, The proposed method employs time-delayed embedding and identifies temporal patterns in the resulting phase spaces. Hence, the method assumes that there transition pattern occurs just before the target event. Once this transition pattern is detected, it is possible to predict the occurrence of an event. Using a training set, a temporal pattern cluster is defined as a neighbourhood of the target event consisting of all points within a certain distance from the target event. The system constructs a heterogeneous collection of temporal pattern clusters.

In [36], Weiss and Hirsh proposed a system to predict events in non-numerical time-series (or categorical time-series). The idea is based on two steps: A) identifying prediction patterns, in which a GA is used to evolve a grammar that defining conditions that are correlated with a particular event, so that the user can predict the occurrence of the event once these conditions happen, and B) generating production rules, in which a greedy approach is used to create an ordered list of prediction patterns from the set of candidate patterns from step A. Although the idea of event-based prediction was tackled in this work, the proposed algorithm was designed for a very particular case and cannot be easily generalised to standard numerical time-series as most real-world applications require.

Time-series classification can be seen as pattern detection, for which, the aim is to distinguish different time-series vectors and map them into different classes. In [9], the authors proposed a new distance measure for time-series classification. The the new distance measure is simply the weighted summation of eight standard time-series distance measures. These are: Euclidean distance, Manhattan distance, Maximum Norm, Mean dissimilarity, Root mean square dissimilarity, Peak dissimilarity, Cosine distance, and Dynamic time-warping distance. Genetic Algorithm (GA) engine is used to optimise the weights. The new distance was tested with 1NN classification and has been compared against standard distance measures. Results indicate that using a combination of weighted distance measures leads 1NN to produce slightly better classification accuracy.

Mendivil [21] *et al.* proposed the use of Master-Salve parallel GA to optimise the architecture and weights of Neural Network to model chaotic time-series. The proposed model showed that the optimised Neural Network with a good accuracy. In [29], Pulido *et al.* proposed a hybrid approach for time-series prediction by using an ensemble neural network with fuzzy integration of responses and its optimization with genetic algorithms.

Relatively little work has been done on event-based time-series prediction using GP. Kattan *et al.* [16] proposed a learning framework based on GP to detect the occurrence of fatigue in EMG signals. The framework uses a sliding window technique to extract statistical features (evolved by GP) from a given training set of EMG time-series and divide the signals into blocks. GP is encouraged to divide the training EMG blocks in a certain way. Later, these blocks are organised into clusters based on their status (i.e., none-fatigue, transition-to-fatigue and fatigue). Unseen EMG signals are divided by an evolved tree into blocks and matched with the pre-defined clusters (which were formed during the training phase). The disadvantage of the proposed method is that it requires an expert to manually label EMG signal blocks for the training phase.

In [13], the authors proposed an evolutionary approach based on GP to evolve mining rules from time-series. The proposed method is based on discretizing the time-series using sliding window techniques to extract features to be divided into equal-size intervalsm and mapped into integer values to be classified into groups. The proposed method used specialized pattern matching hardware (capable of processing 100MB/ second) to locate rule occurrences for the purpose of calculating rule fitness. The authors claimed that their proposed method can be used in two versions. The first version of the method can be used to find rules to predict specific events in a time-series. In the second version, the user has no previous knowledge about the data and simply wants to extract useful information (rules) from the time-series. The authors admit that their method is not better at time-series prediction or classification than existing methods. Instead, their main contribution lies in the flexibility of the new method and the freedom it affords the data miner, in that he or she can specify a rule format and quality function suited for the application at hand.

Recently, in [37], Xie, *et al.* used GP to detect temporal and spatial relationships in a time-series to detect events. The proposed algorithm used a sliding window mechanism that scans the given time-series sequentially. GP evolves a tree which evaluates the data under the sliding window. If the tree's output from two consecutive windows is greater than some defined threshold, then an event is detected.

As can be seen, most time-series event-based detectors analyse the time-series itself and detect different patterns in order to label different events. Unlike other works, in this paper, we try to understand the environment that generates time-series (rather than the usual way of analysing patterns in a single time-series). Thus, the proposed framework learns the patterns that indicate the occurrence of the target event in unseen data. To this end, GP has been used to automatically build a library of candidate temporal features from historical time-series vectors generated from the same environment. Moreover, the proposed framework allows the user to define the target event of interest and customise the training phase accordingly. An advantage of the proposed framework is that it requires no previous knowledge in the problem domain and builds the library of features automatically.

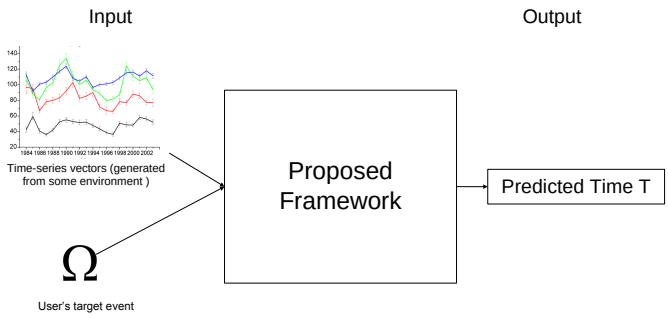

Figure 2: Overview of the framework inputs/output.

3. The Proposed Learning Framework

The proposed framework process is divided into two main phases: *A)* Training phase, in which the framework extracts statistical features from training time-series vectors (generated from the same environment) and matches them with the target event defined by the user (the training phase can be seen as an attempt to understand the distinct set of possible behaviours that an environment may generate in order to predict target events in unseen time-series vectors), and *B)* Testing phase, in which the framework receives an unseen time-series and matches it with learnt patterns in order to predict the time-slot where the unseen time-series is expected to show the target event.

Before the training phase starts, the user needs to set the event that he/she wants to predict. We will refer to this event as Ω in order to represent the target prediction. For example, if the user is interested in predicting the position (i.e., t_i the time of occurrence) of the highest point in a time-series, then, $\Omega = Highest$, if the user is interested in knowing the position of the second-highest point, then, $\Omega = 2^{nd}Highest$, and so on. Thus, Ω allows the user to set the target event in order to train the framework to learn statistical features from historical time-series vectors and correlate them with this target event. Note that Ω represents the order of the point within the entire time-series, not its value. Figure 2 elucidates an abstracted representation of the framework.

3.1. Training Phase

The training process is broadly illustrated in Figure 3. In the training phase, the framework first receives a collection of vectors $S = \{(\mathbf{V_a})|\mathbf{a} = \mathbf{0}...\mathbf{n}\}$. Each $V_a \in S$ is a time-series of t_{max} observations (collected from equidistant time-slots). Thus, $\forall V_a \in S, V_a = \{(\mathbf{x_t})|\mathbf{t} = \mathbf{0}...\mathbf{t_{max}}\}$. Here, we assume that S contains a set of time-series vectors that have been generated from the same environment. The data used to build

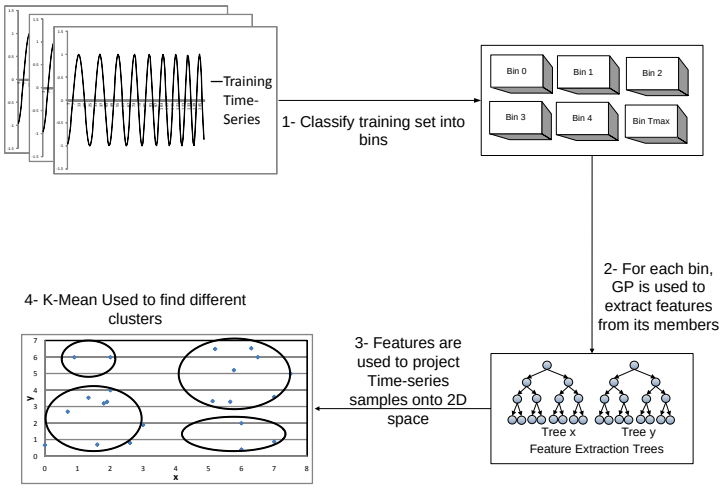

Figure 3: Training process overview.

up the framework in the training phase is referred to as *training set*. At the beginning of the training process, the framework preforms an initial screening for the training set and classifies the vectors into t_{max} $Bins$ so that $Bins = \{(\mathbf{b_i})|\mathbf{i} = \mathbf{0}...\mathbf{t_{max}}\}$. This classification is based on the Ω which is defined by the user, so that each b_i contains the time-series in which the Ω condition occurred at time t. For example, say, $\Omega = Highest$, then, b_0 will contain time-series vectors that have the highest point in time-slot t_0 and b_1 will contain time-series vectors which have the highest point in time-slot t_1, and so on. Figure 4 shows a graphic illustration of this process. Remember that all time-series vectors in the training set have the same length. The reason that we put the training set into different bins in this manner is to identify smaller subsets of the training set so as to simplify the learning process and assist the learner to generalise its knowledge. Thus, instead of letting the learner learn all of the patterns in the training samples at once, we divide the problem into smaller sub-problems that are easier to learn since the position of the target event in all time-series samples is aligned within each bin.

The aim is to learn the patterns in each $b_i \in Bins$ and then match these patterns with an unseen time-series in order to predict the position of the target event. The framework further classifies the training samples in each bin into statistically independent clusters in order to simplify the learning problem (i.e., divide and conquer strategy). One may argue that the user can simply apply a single learner to learn the patterns of time-series vectors in each bin and then pass unseen time-series to all learners using a voting scheme to produce a prediction of the target event's time-slot. Although, this seems like a logical solution, however, as demonstrated in preliminary experiments, this argument is flawed for the following reason. While the framework, after the initial screening process, organises the training set into bins and the members of each $b_i \in Bins$ share a common factor which is that the Ω target event (specified by the

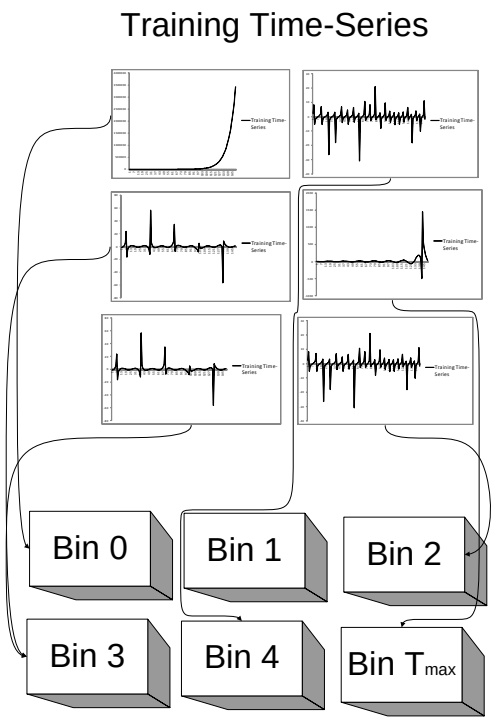

Figure 4: Training Time-series samples are divided into bins based on the location of the Ω event.

user) occurs exactly at time t_i, they may have completely different behaviours (i.e., statistical characteristics) in their trends of going up and down. For example, say, two time-series V_0 and V_1 belong to b_5 where the $\Omega = Highest$. Both V_0 and V_1 have the highest point occurring exactly at time-slot t_5, however, as illustrated in Figure 5, the data-points of V_0 may be a plateau with a single peak at t_5, while V_1 may follow an increasing trend until time t_5 and then decrease until t_{max}. These possible variations in behaviour of different time-series in the same bin may increase the complexity of a single learner to learn their patterns and generalise its model. Therefore, we further classify time-series vectors in each bin into different statistically independent clusters. The problem, here, is that the number of these different behaviours (i.e., their distinct statistical characteristics) is unknown. Moreover, the definition of what is considered to be statistically similar or dissimilar is not clear. In the next sub-section, we will explain how we solved this problem and allow the framework to automatically classify time-series vectors into different statistically independent clusters.

3.1.1. Detect Different Behaviours

The idea of dividing the training set into smaller subsets and learning each subset independently is not new [14, 30, 15]. Here, we used the same concept of divide-and-

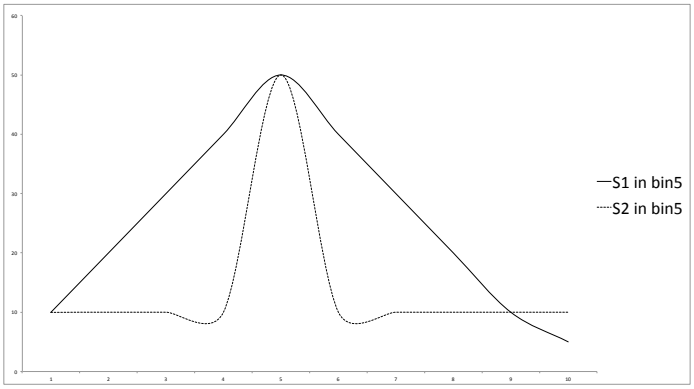

Figure 5: Example of two time-series V_0 and V_1 share the same target event ($\Omega = Highest$) at t_5 but have different behaviour.

conquer using GP to divide the training set automatically.

We used GP to find the number of different behaviours among the time-series vectors in each $b_i \in Bins$. [2] Remember that we use the word 'behaviours' to refer to distinguishable statistical characteristics. Each bin is treated as an independent problem. To this end, a full GP run is executed in each bin where it will start at an initial, randomly generated, population using the ramped half-and-half method (e.g., see [27] and [17]). Each individual in our GP representation is composed of two trees: *Feature* X and *Feature* Y. Each tree receives a time-series vector as its input and returns a single value as the output. For this task, GP has been supplied with a language that allows the discovery of different patterns in the training set. Table 1 reports the primitive set of the GP system. The two outputs together are treated as coordinates for a time-series in a 2-D plane. The process of mapping time-series in the training set to 2-D points is repeated for all of the time-series vectors within each bin. Note that we take all available history into consideration by extracting features from the full time-series, rather than a fixed-width sliding window.

Note that each tree represents a highly composite feature, i.e., a mixture of statistical features combined with constants and arithmetic operators. Once the training time-series are projected via the two components (*Feature* X and *Feature* Y), K-Means clustering [18] is applied in order to group similar instances into different clusters. This process is further illustrated in Figure 6.

During the evolutionary process, the GP projects the training time-series vectors so as to minimise the distance between training samples that have similar statistical features and to maximise the distance between training samples that have dissimilar features. To this end, GP's individuals will be used to generate projections of time-series vectors onto 2-D space. Each individual will be ranked based on its performance, in terms of generating separated clusters, and grouped around their centeriods. However,

[2] Each bin is treated as an independent problem for which our GP system will try to detect different behaviours among its members.

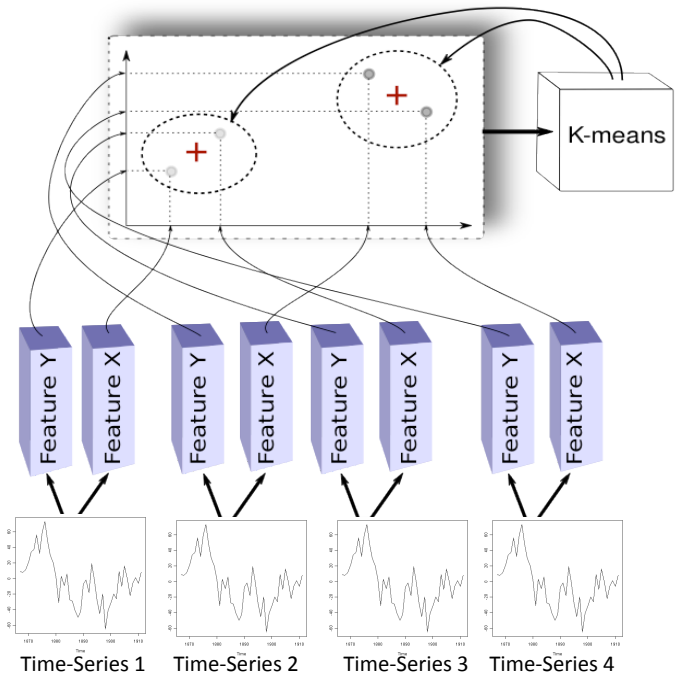

Figure 6: GP evolves two trees; Feature X and Feature Y. Each tree receives time-series vector as input and returns a single value as the output. The two outputs together are treated as coordinates for a time-series in a 2-D plane.

two key problems need to be solved to achieve this aim; *A)* the number of groups (or clusters) to be formed by the K-Mean is unknown, so we need to identify the number of clusters, and *B)* there is no clear definition of what is considered similar or dissimilar, so we need to set a similarity measure. For example, an individual in the GP population (composed of two trees according to our representation) may projects training samples closely on the 2-D space based on their mean values alone. Should considered to be similar?

The next subsection will explain the solutions implemented to overcome these two problems.

3.1.2. Identify the Number of Clusters

We used a standard pattern classification approach on the outputs produced by the two projection trees to discover regularities in the training data from each bin. In principle, any classification method can be used with our approach. Here, we use K-Means clustering (e.g., see [18]) to organise the training data (as re-represented by their two projection trees) into groups. K-Mean was selected for its simplicity of implementation and its execution speed, but other techniques might work equally well. In future work we will explore this particular aspect in the framework.

The K-Means algorithm is a partitioning technique that normally requires the user

Table 1: Primitives set

Function	Arity	Input	Output
+/, -, /, *, pow	2	Real Number	Real Number
Mean, Std, Skewness, Kurtosis, Variance, Average deviation, Entropy	1	Vector of Real Numbers	Real Number
Constants 1-6	0	N/A	Real Number
Training Time-series	0	Vector of real Numbers	N/A

to fix the number of clusters to be formed. However, in our case, the optimal number of subdivisions of a problem into sub-problems is unknown. In our previous work [15], we designed a method that set the optimal number of clusters (or more precisely, near optimal). The idea simply relies on the fact that K-Means is a very fast algorithm. Thus, the framework repeatedly instructs K-Means to divide the data set into K clusters, where $K = 2, 3, ...K_{max}$ ($K_{max} = 20$, in our implementation). At each K-Mean call, the framework computes the clusters' quality. The value of K that provided the best quality clusters is then used to split the training set as the K value. The quality of the clusters is calculated by measuring cluster *separation* and *representativeness*. Ideal clusters are those that are separated from each other and densely grouped near their centroids.

To measure the first quality (i.e., separation), we used a modified version of the Davies Bouldin Index (DBI) [32]. DBI validates how well the clustering has been done using quantities and features inherent to the dataset. DBI measures the nearness of the clusters' members to their centroids, divided by the distance between clusters' centroids. Thus, a small DBI index indicates well separated and closely grouped clusters. Therefore, we favour clusters with a low DBI value.

DBI can be expressed as follows. Let C_i be the centroid of the i^{th} cluster and d_i^n be the n^{th} data member of the i^{th} cluster. In addition, let the Euclidean distance between d_i^n and C_i be expressed by the function dis(d_i^n , C_i). Furthermore, let K be the total number of clusters. Finally, let the standard deviation be denoted as *std()*. Then,

$$DBI = \frac{\sum_{i=0}^{k} std[dis(d_i^0, C_i), ..., dis(d_i^n, C_i)]}{\sum_{i=0}^{k} \sum_{j=i}^{k} ||C_i - C_j||_2}$$

The second quality measure in our implementation is the representativeness of clusters. This is simply evaluated by verifying whether the clusters contain enough members to represent certain behaviour to match its regularities with unseen data. In certain conditions, the projection trees may project the data in such a way that it is unlikely to be suitable for classifying unseen data. For example, clusters that have few members are unlikely to be representative of unseen data. To avoid pathologies of this kind, the framework verifies that the formed clusters have a sufficiently large number of members. In particular, it penalises the values of K that lead K-mean to form clusters in

Algorithm 1: Finding the optimal number of clusters in the projected space.

```
Project(n, treeX, treeY);
List Qk;
for int k=2; k ≤ K_max; k++ do
    //call the K-means algorithm
    K-means(k, n);
    int separation = calculate_DBI();
    if check_clusters_representativeness() == true then
        theta = 0
    else
        theta = 1000
    end
    Qk.append(separation + theta, k)
end
//find the best number of clusters
int number_of_clusters = Qk.get_min_k();
```

which fewer than a minimum number of members are present. In this work, the minimum allowed number of members for each cluster was set to 5 samples. The selection of this number was based on trial and error during preliminary experiments. Note that the minimum number of members in each cluster should be based on the total number of projected data-points. In future research, we will consider to set this variable as a ratio of the total projected points.

More formally, the quality, Q_k, of the clusters obtained when K-Means is required to produce K clusters can be expressed as follows. Let θ_k be the penalty value applied to the quality if K-Mean forms one or more clusters with fewer members than the minimum allowed members. If any particular cluster has fewer than a minimum number of members we set $\theta_k = 1000$, while $\theta_k = 0$ if no problem is found. Furthermore, let DBI_k represent the corresponding cluster separation. Then,

$$Q_k = DBI_k + \theta_k \tag{1}$$

This θ_k value will decide whether the evolutionary process should discriminate against inferior individuals that led to poor projections. Algorithm 1 illustrates the process of identifying the optimal number of clusters in details.

After running K-Means for all values of K in the range from 2 to K_{max}, we choose the optimal K as follows:

$$K_{best} = \arg \min_{Q_2 < k < Q_{k_{max}}} K \tag{2}$$

The main factor that affects the optimal number of clusters is the density of the projected samples. The method described above effectively analyses the density of the data from this point of view. On one hand, the advantage of this approach is that it greatly simplifies the classification of time-series behaviours. This is because evolution

pushes projection trees to represent the data in such a way as to optimise the performance of the classification algorithm. Moreover, this approach does not require the experimenter to split the training set manually, which may be difficult to do and may require human expertise in the problem domain. Also, the approach does not impose any significant constraints on the shape of the clusters. On the other hand, a disadvantage of this approach is that the K-Means algorithm has to be executed several times per fitness evaluation, which slows down the evolution a little. However, this only needs to be done during evolution. Once the framework ends the training phase, it will be ready to predict the behaviour of unseen time-series based on the projected clusters.

Once GP projects the training samples of any particular bin into separated and representative clusters, as defined above, it is reasonable to assume that time-series members of each cluster shares similar statistical features.

3.1.3. Fitness Evaluation

We evaluate the GP's individuals (represented by two projection trees) by measuring how well they projected the training samples into high quality clusters as defined in Section 3.1.2. Thus, as defined in Equation 1, the fitness value of any individual is calculated as follows:

$$Fitness = \arg \min_{2<k<k_{max}} Q_k$$

For each individual we store the best K value. Thus, when the framework recalls the best-of-run individual it will simply re-project the data and run the K-Mean algorithms without the need to repeat the process of identifying the best K value again.

The clusters formed by K-Means represent subsets of training examples. Each cluster represents a particular behaviour in the bin that it belongs to.

3.1.4. Search Operators

We used tournament selection and the standard genetic operators: sub-tree crossover, sub-tree mutation, and reproduction. Naturally, the genetic operators have to take the multi-tree representation of individuals into account.

There are several options for applying genetic operators to a multi-tree representation: apply an operator to all trees within an individual, or use different operators for different trees. Also, there is the option of constraining crossover to only happen between trees at the same position in the parents or allow crossover between different trees within the representation, and so on.

Similar to our previous work in [15], we allow crossover to freely select feature-extraction trees. In other words, the *Feature X* tree of one parent can be crossed over with either the *Feature X* tree or *Feature Y* of the other parent and *vice versa*.

3.2. Testing Phase

In the training phase, the framework automatically builds a library of candidate temporal features. This is achieved by separating the training set into different bins based on the exact time t_i of occurrence of the Ω target event and then form statistically independent clusters in each bin. In the testing phase, the framework will receive an unseen time-series (point-by-point) in sequential order and match its patterns with all formed clusters. Once an accurate match is found (with an acceptable confidence

level) the framework can indicate that the unseen time-series share similar statistical characteristics with previously seen historical time-series vectors and it predicts that the target event will be likely to occur at the same time-slot as these historical time-series vectors. Remember that all time training examples within each bin $b_i \in Bins$ have the Ω target event occur exactly at time t_i and all training examples within the same cluster (in any bin) have similar statistical features.

We assume that, in a real-world application, unseen observations generated from an environment are not available beforehand. Therefore, the framework processes new observations sequentially (i.e., point-by-point as they become available) in real time. To this end, newly collected points, $\{\hat{x_0}, \hat{x_1}, ..., \hat{x_{t_{max}}}\}$ are stored in a vector referred to as $\hat{V}$. The dimensionality of this vector increases as new observations are collected from the environment, and thus, let $D(\hat{V})$ be the dimension of the $\hat{V}$ vector (i.e., the number of newly collected unseen observations). At each collected observation, the framework will calculate the average Euclidean distance between $\hat{V}$ and clusters' members (i.e., time-series vectors) for all clusters of all bins. Each bin will contain at least 2 clusters and at most K_{max} clusters. Note that the Euclidean distance is calculated according to the available dimensions. For example, if $D(\hat{V}) = 4$ and $t_{max} = 10$ (remember that t_{max} represents the length of training time-series vectors), the framework will only look at the first 4 dimensions in all training time-series in all clusters in all bins. This is normal because there is no straightforward way to calculate the distance between two vectors of different dimensionality. The average distance between the newly collected observations and each cluster in b_i is calculated as follows:

$$AverageDistance = \frac{1}{N_{b_{g(cluster_k)}}} \times \sum_{j=0}^{N_{b_{g(cluster_k)}}} \sqrt{\sum_{t=0}^{D(\hat{V})} (\hat{V}[\hat{x_t}] - V_j[x_t])^2} \quad (3)$$

Here, $N_{b_{g(cluster_k)}}$ is the total number of members in the k^{th} cluster of the g^{th} bin. The average distance is calculated for all clusters in each bin. The cluster (in all bins) that returns the minimum average distance (as defined in Equation 3 above) is considered to be the one that most likely shares similar statistical features with the newly collected observations. Thus, the framework predicts that the target event will occur at time t_i where i is determined based on the i^{th} bin in which the cluster that returned the lowest average distance falls in. We also calculate the confidence level of the prediction.

3.2.1. Confidence Level

The confidence level tells the framework how close the given prediction is to a previous historical case. This allows the framework, as we will see later, to decide whether it is confident enough to give the user a prediction, or whether it not confident in its prediction and still needs to gather more information about the unseen time-series.

The confidence level is simply the smallest Euclidean distance between the collected observations from the unseen time-series and a training sample from the cluster that produced the lowest average distance (as defined in Equation 3) divided by the dimensions of the $\hat{V}$. Thus, the average distance measures the similarity between the collected observations to a previously detected behaviour. The confidence level tells

us how similar the collected observations are to a particular historical case. Here, the more similar the collected observations are to a previous historical case, the more confident the framework will be about its prediction. The confidence level is calculated as follows:

$$confidence = \frac{1}{D(\hat{S})} \times \arg \min_{1<a<cluster_k} \sum_{q=0}^{|cluster_k|} \sqrt{\sum_{t=0}^{D(\hat{V})} (\hat{V}[\hat{x_t}] - V_{q_{(cluster_k)}}[x_t])^2} \tag{4}$$

Here, $\min(cluster_k)$ refers to the k^{th} cluster that returned the lowest average distance in all bins while $|cluster_k|$ is the total number of time-series vectors in the k^{th} cluster. Moreover, $V_{q_{(cluster_k)}}$ is the q^{th} time-series vector in this k^{th} cluster. Finally, we divided the confidence by the dimensions of the collected observation (i.e., number of collected observations). This is necessary to allow for a comparison of different confidence levels from different dimensions. For example, at time t_0 the framework will receive the first observation $\hat{V} = \{\hat{x_0}\}$ from the environment and calculates its average distance toward all clusters in all bins. The cluster that returns the lowest average distance most likely shares similar statistical features with the collected observation. Then, the confidence level will be calculated as denoted in Equation 4. Next, at time t_1 $\hat{V} = \{\hat{x_0}, \hat{x_1}\}$ where $\hat{V}$ will be two dimensions. Here, we divide the confidence level by dimensions of $\hat{V}$ in order to allow a comparison between the confidence levels from t_0 and t_1. The framework favours lower confidence levels as this means a smaller distance between collected observations and a previous historical case.

3.2.2. Prediction

For each collected $\hat{x_i}$ observation from the environment, where $i = \{0, 1, ..., t_{max}\}$, the framework calculates a prediction $\hat{t}$ of the position of the Ω target event and a confidence level. Hence, starting from $t = 0$ (first time-slot), where $\hat{V} = \{\hat{x_0}\}$, a prediction $P = \hat{t}$ and confidence C are calculated. If the predicted $\hat{t}$ is still in the future (i.e., $\hat{t} > t$, where t is the current time-slot) the framework decides to collect more observations until time $\hat{t}$ assuming that there is still room to increase the prediction accuracy by collecting more data. At each newly collected observation from the environment the framework will calculate a prediction and its confidence level (using Equations 3 and 4). Once the framework finds a better confidence level in subsequent time-slots it will update P and C accordingly. If the newly updated $P = \hat{t}$ is still in the future, then the framework will keep collecting new observations until time $\hat{t}$. Otherwise, if the newly updated $P = \hat{t}$ has already passed, then the framework stops collecting new observations and returns the current time-slot. If $P = \hat{t}$ meets time t (the current time-slot) then the frameworks declares that the target event has occurred.

This process is explained in detail in Algorithm 2. In lines **1-3**, the framework initiates a prediction variable, confidence variable, and unseen observations' vector. In lines **4-22**, the framework enters a loop until t_{max}. A new observation point $\hat{x_i}$ is collected from the environment and stored in vector $\hat{V}$ at line **6**. Then, the framework calculates a prediction $\hat{t}$ for the time-slot where the target event is likely to occur and a confidence level for this prediction in lines **7-8**, as explained previously. Lines **9-13**

***Algorithm 2**: Prediction of target event's position.*

```
1  var Final_confidence = MAX_INTEGER;
2  var Final_prediction = MAX_INTEGER;
3  var V̂ //Vector to store new unseen observations
4  repeat
5      //Collect new x̂_i observation from the environment and store it into V̂
6      Append_New_Observation(x̂_i, V̂);
7      t̂ = Predict (V̂); //Predict the position of the target event
8      C = Confidence(V̂); //Calculate the prediction's confidence ;
9      if C < Final_confidence then
10         //If the new confidence better than previous then update
11         Final_confidence = C;
12         Final_prediction = t̂;
13     end
14     if Final_prediction == t then
15         //The predicted time-slot meets the current time
16         //Break the loop and return Final_prediction;
17     end
18     if Final_prediction < t then
19         //The predicted time-slot less than the current time
20         //Break the loop and return Final_prediction;
21     end
22 until t != t_max;
```

update the initial variables with the best confidence level, so far. Thus, the initial variables will be updated once the framework finds better confidence level. Lines **14-17** check whether the predicted time-slot ($Final\_prediction$) meets the current position where the framework is paused. Remember that the framework collects unseen observations sequentially starting from the first observation $\hat{x_0}$ that occurred at time $t = 0$ until $x_{t\_\hat{max}}$. At each new collected observation the frameworks tries to predict the position of the target event. In lines **18-21** the framework will break the loop in case the predicted time-slot is less than the current time-slot and returns the $Final\_prediction$.

4. Experiments

4.1. Experiment Strategy

The key question that should be addressed in any experimental setup is: will the given algorithm solve the problem that it intends to solve or not ?

In an abstract sense, algorithms can be viewed as mathematical formulation of a particular problem and a set of computer instructions to solve this problem. Naturally, all algorithms are bounded to the *"No Free Lunch"* theory and it is not possible to

Table 2: Parameters setting for Standard GP

Parameter	Value
Generations	20
Population	100
Crossover	90%
Mutation	5%
Elitism	5%
Tournament size	2
Primitive set	+,-,*,/, constants (1-6)

design a single approach that solves all instances in a class of problems. Ideally, one would like to prove an algorithm's robustness and performance mathematically. However, due to the complexity of most real-world problems a mathematical proof may be beyond human capability. Another problem with the mathematical proof (or analytical proof) is that the implementation of the algorithm will differ depending on the programming language and the computer architecture used to run it. Thus, even if a perfect analytical proof of an algorithm exists it will not draw solid conclusions about its performance across different platforms. Perhaps in the near future researchers may invent heuristics techniques that evolve mathematical proof that take into consideration all factors that impact algorithms' performance and one would save time to empirically test an algorithm. However, for now, researchers tend to design empirical experiments that focus on a small (often very small) area in the problem space and draw some conclusions about their algorithms' strengths and weaknesses. In fact, most researchers in the area of evolutionary computation only test their algorithms empirically [31]. To this extent, one can design insightful experiments by testing the algorithm empirically without any particular theoretical assumption and let the evidence drive the conclusions [31].

Our experiments are designed to test the behaviour of the proposed framework under different circumstances. The experiments are divided into two parts. In the first part, experiments are carried out on artificially generated data where we can control the difficultly of the problem and simulate different levels of hardness. In the second part, we test the algorithm with a real-world problem. The reason for testing the framework on artificially generated data is to build an understanding of the framework's behaviour under a controlled environment where we can analyse the effect of each variable on the performance. Furthermore, the artificially generated data can represent (to some extent) a group of real-world applications. Nevertheless, testing the framework with real-world data will show its performance in situations where the data can be stochastic and controlled by a real environment, which is difficult to generate artificially. In addition, testing the framework with real-world data will demonstrate the potential of our framework.

4.2. Algorithms Implementation and Settings

We compared the proposed framework with a standard GP [27], standard Radial Basis Functions Networks (RBFN) [4], Gaussian Process (or sometimes called Krig-

Table 3: Parameters setting in the proposed framework

Parameter	Value
Generations	20
Population	100
Crossover	90%
Mutation	5%
Elitism	5%
Tournament size	2
Max Clusters	20
Clusters min allowed members	5

ing) [3], Linear Regression model (LR) [3], and Polynomial Regression model (PR) [3]. The reason we chose these models is that they are some of the most widely used algorithms for time-series applications. In order to ensure a fair comparison, before applying the GP, RBFN, Kriging, LR and PR the training data are classified into different bins (similar to our approach) and we let each algorithm learn the patterns in each bin separately.

All parameters are chosen experimentally in such a way as to improve the convergence rate of all algorithms in comparisons. Table 3 illustrates the settings used in our proposed framework.

For standard GP, the system solves the problem as a standard symbolic regression problem. The GP receives the time-series vectors in each bin (each bin is solved as an independent problem) and evolves a function $f(x)$ in such a way as to minimise the error between the evolved function's outputs and the time-series vectors within a bin. Once GP evolves a generalised prediction function for each bin, unseen observations are treated in same manner as in the proposed framework (see Section 3.2.2). Thus, each unseen observation $\hat{x_i}$ is passed to each evolved function in each bin. The function that produces the lowest prediction error returns a prediction of the position of the Ω target event based on samples in its associated bin, and the prediction error as confidence level. Here, lower prediction error indicates higher confidence level. The same procedure as described in Algorithm 2 follows. The settings used for standard GP are descried in Table 2.

Similarly, for the RBFN, Kriging, LR and PR a prediction model is trained using the time-series vectors in each bin so as to minimise the training error. Again, at each $\hat{x_i}$ unseen observations are passed to trained RBFN models in each bin. The model that produces the lowest prediction error returns a prediction of the position of the Ω target event as it is labelled on its associated bin, and the prediction error as confidence level. The same procedure described in Algorithm 2 is followed. For RBFN, we used the standard notation described in [22].

4.3. Artificially Generated Data

Experiments on artificially generated data were sampled from two different models as denoted below.

1. $S(x_i) = sin(r_i) \times c_i$.
2. $S(x_i) = tan(r_i) \times c_i$.

Here, $r_i \in R = \{r_0, r_1, ...r_{t_{max}}\}$, where R is an ordered list of random numbers uniformly generated from the interval $[0, 5]$ and $c_i \in C = \{c_0, c_1, ...c_{t_{max}}\}$ where C is an ordered list of constants starting from random constant from the interval $[1, 7]$ and decreases in steps of 0.01. Thus, changing the R and C lists will result in completely different time-series. We used each model to generate 2000 time-series vectors for the training set and another 100 time-series vectors for the testing set. Both training and testing sets contain completely different samples. We used variations of the Sin and Tan wave functions to test the framework ability to differentiate between similar periodic behaviours. Figures 7 and 8 illustrate some examples of time-series generated using the models above. For each model, from the two models listed above, we tested the algorithm in a full experimental set. For each experimental set, of each model, we tested the framework under different values of t_{max}. Namely, $t_{max} = \{5, 10, 15, 20, 25\}$, where for each value we tested the model through 20 independent runs. Thus, in total, for the experimental test on artificially generated data, we ran the framework $2 \times 5 \times 20$ different times (this is 2 models, $5 t_{max}$ values, and 20 independent runs for each model-t_{max} combination). Remember that t_{max} defines the length of the generated time-series. It is also the number of bins the framework will generate to distribute the training samples within according to the positions of their target events. Note that we focused the experiments on small values of t_{max}. This is to stress the framework ability to detect the Ω target event in limited time-slots.

For all experiments, we set the target event as $\Omega = Highest$. Thus, we train the framework to predict the position of the highest point in unseen time-series. This is not an easy task given the periodic nature of the generated time-series. The reason for using the Sin and Tan wave functions is that they have different forms of periodic behaviours. In the Sin function, peaks tend to be clear and smooth (as illustrated in Figure 7). The difficulty of this task is discovering the position of the highest peak among several regular peaks in the time-series. Whereas the Tan function generates rather small variations in the time-series with several sharp positive and negative peaks (see Figure 8). The difficulty of this task is detecting this sudden variation in the data without a clear warning before. Also, the framework needs to detect the highest peak among all these variations. Note that the framework has no knowledge about the function used to generate the data. However, it tries to learn the environment's possible behaviours and to match new observations with previously detected behaviours.

4.3.1. Results

In each run, we calculated the *Mean, Max, Standard Deviation*, and *Median* of *Hit_rate* and *Accuracy*. The *Hit_rate* shows the percentage of times where thaat each algorithm managed to correctly predict the position of the target event (highest peak in our case), while the *Accuracy* shows how far predictions are from the real time-slot of the target event. The accuracy is calculated as follows. Let *Closeness* be an equation to measure how close the given prediction is to the real position of the time event. Thus,

$$Closeness = 1 - \frac{prediction\_error}{max\_possible\_error}$$

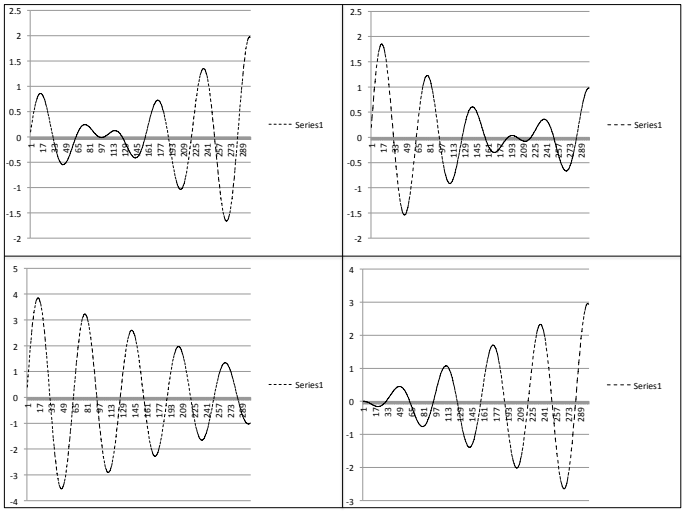

Figure 7: Example of time-series generated using variation of the Sin function.

where $prediction\_error$ is the summation of the absolute difference errors between the predictor's output and the real position of the target event on the testing set, while $max\_possible\_error$ is the worst possible prediction that may happen for each target event. This is calculated as follows:

$$\text{max\_possible\_error} = \begin{cases} \text{T(target\_event)}_i & \text{ifT(target\_event)}_i > t_{max} - T(target\_event)_i \\ \text{t}_{max} - T(target\_event)_i & \text{Otherwise} \end{cases}$$

$T(target\_event)_i$ is the position (i.e., time slot) where the target event has occurred in the unseen time-series.

Using the above equations the accuracy of predictions is calculated as follows:

$$Accuracy = \sqrt{Closeness \times Hit\_rate} \tag{5}$$

Table 4 summarises the results of 600 independent runs for the Sin function. This is 20 independent runs for each system under each t_{max} value. As can be seen in the Table, the proposed framework (referred to as *GP-Clusters* in the results Tables) has achieved higher numbers in most experimental cases in terms of mean, max and median.

The only cases where the proposed framework has failed to achieve the highest median is in $t_{max} = 20$ and 25 against RBFN and LR. Also, it failed to achieve the highest max in $t_{max} = 15$.

Now, if we focus our attention on the *Hit_rate* we will find that the proposed framework achieved a better mean than all competitors with margins ranging from 16% to 48% on average. In addition, it achieved better max *Hit_rate* (i.e., maximum *Hit_rate* across the whole 20 runs) with margins ranging from -5% to 44%. It is notable that the highest *Hit_rate* is 55% of the testing cases. This is because the problem of predicting the exact position of a target event is too difficult. Therefore, we do not discard non-hit predictions and consider them completely wrong because they still can give an indication to the user. The prediction accuracy, as explained above, shows how far non-hit

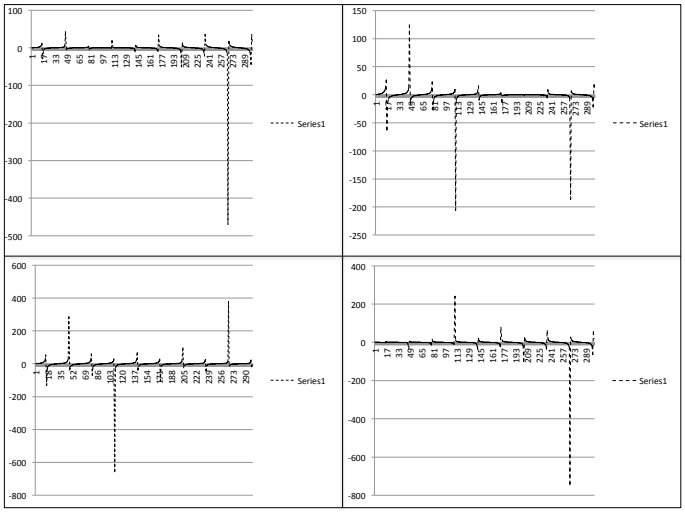

Figure 8: Example of time-series generated using variation of the Tan function.

predictions are far from the real time of the target event. If we now turn our attention to accuracy results in Table 4 it is clear that our proposed framework has higher mean, max, and median accuracy than those of the competitors in all test cases. To further verify the significance of our results in Table 4 a Kolmogorov-Smirnov two-sample test [26] has been performed on the results for all pairs of systems under test and for all five test t_{max} values. Table 5 reports the *P-value* for the tests. As one can see, in 16 out of 25 cases our framework is statistically significantly superior to its competitors at the standard 5% significance level.

From the results in Table 4, we can see that the proposed framework comes in first place in most cases, then standard GP in second place, and finally RBFN in third place.

Table 6 summarises the results of 600 independent runs for the Tan function (i.e., the second model). This is 20 independent runs for each system under each t_{max} value. It is clear from the table that our proposed algorithm is better than its competitors in most cases in both *Hit_rate* and prediction's accuracies. When $t_{max} = 15$ the framework achieved a similar median as standard GP, Kirging, and LR. When $t_{max} = 25$ the framework was outperformed by the RBFN in terms of max accuracy and achieved similar Hit_rate as the Kriging. Note that, unlike with the Sin function, the Tan generates time-series with sharp and irregular peaks; therefore, it is difficult to detect the target event efficiently. However, the proposed framework achieves higher mean *Hit_rate* on most test cases with margins ranging from 11% to 43%. It also achieves a higher max *Hit_rate* in all test cases with margins ranging from -0.02% to 19%.

Again, we verified the significance of our results in Table 6 using a Kolmogorov-Smirnov two-sample test [26] on the results for all pairs of systems under test and for all five test t_{max} values. Table 7 reports the *P-value* for the tests. In 12 cases our system is statistically significantly superior to its competitors.

In this subsection, we illustrated the framework performance in detecting a pre-

Table 4: Summary of 300 runs with the Sin function (20 runs for each system under each T_{max} value.)

	GP-Clusters		SGP		RBFN		Kriging		LR		PR	
	Hit	Accuracy	Hit	Accuracy	Hit	Accuracy	Hit	Accuracy	Hit	Accuracy	Hit	Accuracy
$t_{max} = 5$												
Mean	**0.33**	**0.46**	0.23	0.37	0.19	0.31	0.19	0.30	0.21	0.33	0.22	0.33
Max	**0.55**	**0.67**	0.45	0.59	0.45	0.54	0.45	0.57	0.55	0.62	0.45	0.53
StD	0.13	0.12	0.10	0.11	0.12	0.15	0.13	0.16	0.11	0.12	0.12	0.15
Median	**0.32**	**0.48**	0.18	0.34	0.18	0.33	0.18	0.32	0.18	0.35	0.27	0.37
$t_{max} = 10$												
Mean	**0.18**	**0.30**	0.10	0.19	0.09	0.17	0.15	0.25	0.09	0.16	0.12	0.22
Max	**0.45**	**0.56**	**0.45**	0.52	0.36	0.44	0.36	0.49	**0.45**	0.53	0.36	0.49
StD	0.13	0.16	0.12	0.16	0.10	0.15	0.12	0.15	0.11	0.16	0.09	0.14
Median	**0.18**	**0.32**	0.09	0.22	0.09	0.20	0.14	0.27	0.09	0.20	0.09	0.24
$t_{max} = 15$												
Mean	**0.15**	**0.24**	0.05	0.11	0.06	0.12	0.06	0.13	0.05	0.12	0.10	0.17
Max	0.45	**0.60**	0.18	0.33	0.27	0.41	0.27	0.42	0.27	0.43	**0.55**	0.63
StD	0.13	0.18	0.06	0.13	0.08	0.15	0.08	0.15	0.07	0.14	0.13	0.17
Median	**0.18**	**0.31**	0.00	0.00	0.00	0.00	0.05	0.10	0.00	0.00	0.09	0.21
$t_{max} = 20$												
Mean	**0.15**	**0.26**	0.04	0.08	0.05	0.14	0.06	0.13	0.07	0.15	0.03	0.07
Max	**0.55**	**0.65**	0.18	0.32	0.18	0.35	0.18	0.35	0.18	0.36	0.18	0.34
StD	0.16	0.20	0.06	0.12	0.05	0.13	0.07	0.13	0.07	0.15	0.05	0.11
Median	**0.09**	**0.25**	0.00	0.00	**0.09**	0.21	0.05	0.10	**0.09**	0.23	0.00	0.00
$t_{max} = 25$												
Mean	**0.15**	**0.26**	0.04	0.08	0.05	0.14	0.03	0.06	0.02	0.06	0.06	0.13
Max	**0.55**	**0.65**	0.18	0.32	0.18	0.35	0.18	0.34	0.09	0.26	0.36	0.43
StD	0.16	0.20	0.06	0.12	0.05	0.13	0.05	0.12	0.04	0.11	0.09	0.14
Median	**0.09**	**0.25**	0.00	0.00	**0.09**	0.21	0.00	0.00	0.00	0.00	0.05	0.10

*__Bold__ numbers are the highest

Table 5: Kolmogorov-Smirnov two samples statistical test for Sin function's results

	GP-Clusters Vs SGP	GP-Clusters Vs RBFN	GP-Clusters Vs Kriging	GP-Clusters Vs LR	GP-Clusters Vs PR
$t_{max} = 5$	0.1349	**0.0232**	**0.0082**	**0.0026**	**0.0232**
$t_{max} = 10$	**1.83E-04**	**7.25E-04**	**1.53E-06**	**7.25E-04**	**1.53E-06**
$t_{max} = 15$	**2.32E-02**	0.0591	0.0591	**0.0232**	0.771
$t_{max} = 20$	**0.0026**	0.1349	**0.0082**	0.4973	**0.0026**
$t_{max} = 25$	0.0591	**0.0232**	0.771	**0.0232**	0.1349

*__Bold__ numbers are lower than 5% statistically significant

defined target event in a periodic unseen time-series. The results are encouraging in the sense that the framework can actually predict the position of the target event and in those cases where it fails to give an exact prediction it is not too far from the real position. In the next subsection we will present results of experiments conducted on real-world data.

4.4. Real-World Data

For the real-world problem, we used data from *Google Trends* service [11]. Google Trends is a free service provided by Google offering data about the search terms that people entered into Google search engine. The service provides free downloadable historical time-series data about any keyword. It, also, offers the flexibility to restrict the search by country.

One of the uses of Google Trends is for E-Marketing managers to monitor the internet to see how often people type certain keywords related to their products at different times over the year. Using this information, E-Marketing managers can decide the best time to release their marketing campaigns so their advertisements coincide with people's searchers and eventually achieve higher hit rates.

For the purpose of our experiments, we imported time-series data about people's searches for the keywords; *Mobile Phone, Holidays*, and *Cinema* and we restricted the search to get data from *USA, USA* and *UK*, respectively. All imported data from Google Trends represent the weekly frequencies of searches of the keywords mentioned above in Google since January 2004 until May 2012. There are 439 data points. Figures 9, 10, and 11 visually illustrate the data used in the experiments. Data were divided into chunks of 5 data points, ending with 87 different chunks (or different time-series vectors). To this end, we used the first 67 time-series vectors to train the framework (and the other competitors) and the last 20 as the testing set. Here, the aim is to predict the position of the highest peak in the future (treating the testing set as unseen weekly observations in the future) so as to help mobile companies to select the best time to advertise their new offers, travel agencies to predict the best time of releasing holidays packages, or to help cinema companies to preview new shows at the most appropriate time.

Table 6: Summary of 600 runs with the Tan function (20 runs for each system under each t_{max} value.)

	GP-Clusters		SGP		RBFN		Kriging		LR		PR	
	Hit	Accurcy	Hit	Accurcy	Hit	Accurcy	Hit	Accurcy	Hit	Accurcy	Hit	Accurcy
$t_{max} = 5$												
Mean	**0.27**	**0.40**	0.21	0.34	0.18	0.30	0.22	0.36	0.15	0.26	0.19	0.31
Max	**0.55**	**0.63**	0.45	0.58	0.45	0.57	0.36	0.51	0.27	0.43	0.45	0.58
StD	0.13	0.15	0.10	0.12	0.12	0.16	0.10	0.13	0.10	0.15	0.12	0.16
Median	**0.27**	**0.44**	0.18	0.34	0.18	0.31	0.23	0.38	0.18	0.30	0.18	0.32
$t_{max} = 10$												
Mean	**0.23**	**0.36**	0.10	0.19	0.10	0.19	0.09	0.17	0.10	0.19	0.10	0.20
Max	**0.45**	**0.58**	0.36	0.47	0.27	0.45	0.27	0.46	0.27	0.41	0.27	0.45
StD	0.12	0.15	0.10	0.16	0.09	0.15	0.11	0.17	0.08	0.14	0.08	0.13
Median	**0.23**	**0.37**	0.09	0.21	0.09	0.22	0.09	0.21	0.09	0.23	0.09	0.25
$t_{max} = 15$												
Mean	**0.11**	**0.20**	0.06	0.14	0.08	0.16	0.08	0.18	0.06	0.14	0.04	0.10
Max	**0.36**	**0.53**	0.18	0.35	0.18	0.35	0.27	0.46	0.18	0.36	0.18	0.33
StD	0.12	0.19	0.07	0.14	0.07	0.14	0.07	0.14	0.07	0.14	0.05	0.13
Median	**0.09**	0.22	**0.09**	0.21	**0.09**	**0.23**	**0.09**	0.21	**0.09**	0.18	0.00	0.00
$t_{max} = 20$												
Mean	**0.10**	**0.20**	0.06	0.13	0.04	0.09	0.05	0.13	0.07	0.16	0.05	0.11
Max	**0.27**	**0.44**	**0.27**	0.43	0.18	0.34	0.18	0.36	0.18	0.38	0.18	0.34
StD	0.09	0.15	0.08	0.16	0.05	0.13	0.05	0.13	0.06	0.13	0.06	0.13
Median	**0.09**	**0.22**	0.00	0.00	0.00	0.00	**0.09**	0.20	0.09	0.20	0.00	0.00
$t_{max} = 25$												
Mean	**0.07**	**0.16**	0.03	0.07	0.04	0.08	0.05	0.12	0.04	0.09	0.01	0.03
Max	**0.18**	0.37	0.09	0.26	0.27	**0.38**	**0.18**	0.33	0.09	0.26	0.09	0.25
StD	0.06	0.13	0.04	0.11	0.07	0.12	0.06	0.13	0.05	0.11	0.03	0.08
Median	**0.09**	**0.21**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

*__Bold__ numbers are the highest

Table 7: Kolmogorov-Smirnov two samples statistical test for Tan function's results

	GP-Clusters Vs SGP	GP-Clusters Vs RBFN	GP-Clusters Vs Kriging	GP-Clusters Vs LR	GP-Clusters Vs PR
$t_{max} = 5$	**0.0082**	**0.0232**	0.1349	**0.0026**	**0.0232**
$t_{max} = 10$	**7.25E-04**	**2.60E-03**	**7.25E-04**	**7.25E-04**	**4.15E-05**
$t_{max} = 15$	0.4973	0.4973	0.4973	0.4973	0.2753
$t_{max} = 20$	0.4973	0.1349	0.771	0.4973	0.2753
$t_{max} = 25$	0.1349	**0.0591**	0.771	**0.1349**	**0.0026**

*__Bold__ numbers are lower than 5% statistically significant

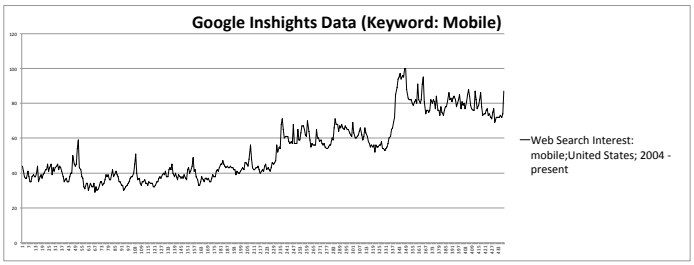

Figure 9: Google Trends Data used in the experiments.
Keyword: Mobile Phone.
Restirction: United State.
Dates: Since January 2004 - May 2012.

4.4.1. Results

Table 8 illustrates the results of the runs. In total, we ran the three systems 300 times. This is 20 times for each system under each keyword. We report the same measures as in the previous set of experiments. The prediction accuracy is calculated in Equation 5. As can be seen in the table, it is amazing that our proposed framework outperformed all competitors in 2 out of 3 test cases significantly. In terms of *Hit_rates*, our framework achieved the highest mean in all 2 keywords (namely, *Holidays* and *Cinema*) with margins ranging from -6% to 25% and it achieved highest max *Hit_rates* with margins ranging from -5% to 27%. It is also interesting to see that the framework precisely predicted the target event in 56% of the testing cases under *Holidays* keyword. As in the previous experimental set, results were verified using Kolmogorov-Smirnov two samples in Table 9. Our results are statistically superior in 10 out of 15 cases. It is interesting to see the framework doing well in real-world situation. These results are encouraging in the sense that good predictions have been achieved and further improvement is still possible.

Interestingly, when the framework obtained results near to standard GP, RBFN and Kriging in the *Mobile* testing case the corresponding *P-Value* shows $P > 5\%$. One might expect that performing more runs would eventually statistically confirm the su-

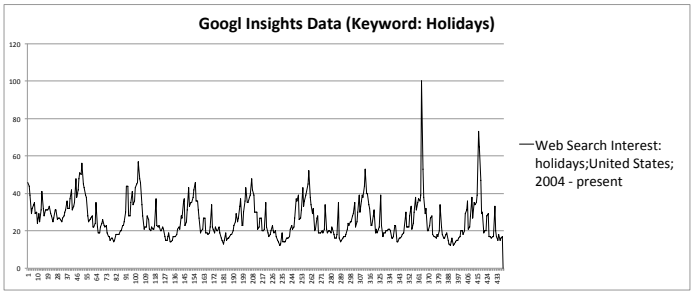

Figure 10: Google Trends Data used in the experiments.
Keyword: Holidays.
Restirction: United State.
Dates: Since January 2004 - May 2012.

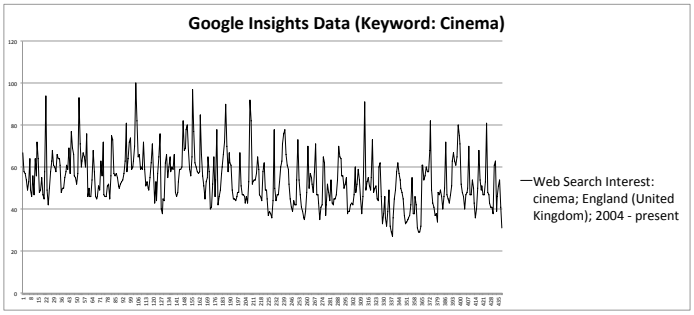

Figure 11: Google Trends Data used in the experiments.
Keyword: Cinema.
Restirction: United Kingdom.
Dates: Since January 2004 - May 2012.

periority of our framework in more cases.

In the next section, we will analyse different aspects of the framework and look closely at the predictions shedding light on the dynamics of the framework's procedures.

5. Analysis

Due to the dynamic nature of evolutionary algorithms, experiments render distributions not numbers [31]. It is no longer sufficient to report the mean of best-of-run values over a finite number of runs and to perform an off-the-shelf and statistical test to conclude that the presented work is robust. It is important to build an understanding of why the algorithm performs well on the testing cases and poor in other cases (assuming the experimental design was broad enough) in order to gain more knowledge about the problem it is meant to solve and about the behaviour of the algorithm itself. This will direct future works and help readers to start from where others ended.

Table 8: Summary of 300 runs with the Google Trends data (20 runs for each system under each keyword.)

	GP-Clusters		SGP		RBFN		Kriging		LR		PR	
	Hit	Accuracy	Hit	Accuracy	Hit	Accuracy	Hit	Accuracy	Hit	Accuracy	Hit	Accuracy
Mobile - US												
Mean	0.22	**0.37**	0.23	**0.37**	**0.24**	0.36	0.19	0.33	0.12	0.22	0.19	0.32
Max	**0.43**	0.55	0.38	0.49	0.38	0.53	0.33	0.45	0.19	0.35	0.38	0.49
StD	0.10	0.09	0.09	0.07	0.09	0.09	0.08	0.08	0.06	0.09	0.07	0.07
Median	0.19	0.34	0.19	**0.36**	**0.24**	0.35	0.19	0.34	0.12	0.22	0.19	0.28
Holidays - US												
Mean	**0.32**	**0.42**	0.19	0.31	0.17	0.30	0.18	0.29	0.20	0.33	0.24	0.35
Max	**0.56**	**0.59**	0.38	0.43	0.44	0.53	0.38	0.51	0.44	0.55	0.38	0.45
StD	0.14	0.12	0.10	0.08	0.09	0.11	0.11	0.13	0.08	0.07	0.12	0.08
Median	**0.34**	**0.44**	0.13	0.27	0.16	0.29	0.13	0.27	0.19	0.32	0.31	0.40
Cinema - UK												
Mean	**0.24**	**0.40**	0.18	0.33	0.21	0.35	0.19	0.33	0.15	0.28	0.17	0.30
Max	0.31	**0.45**	**0.32**	0.43	0.29	0.42	0.27	0.41	0.23	0.38	0.19	0.34
StD	0.04	0.04	0.05	0.04	0.05	0.05	0.05	0.05	0.05	0.07	0.02	0.02
Median	**0.23**	**0.39**	0.18	0.34	0.20	0.36	0.19	0.33	0.15	0.27	0.16	0.29

*__Bold__ numbers are the highest

Table 9: Kolmogorov-Smirnov two samples statistical test for Google Insight results.

	GP-Clusters Vs SGP	GP-Clusters Vs RBFN	GP-Clusters Vs Krig-ing	GP-Clusters Vs LR	GP-Clusters Vs PR
Moblie - US	**0.0026**	0.4973	0.2753	**1.53E-06**	**0.0026**
Holidays - US	0.771	**2.60E-03**	0.4973	0.0591	**0.0082**
Cenima - UK	**2.49E-07**	**0.0232**	**1.83E-04**	**1.53E-06**	**4.74E-09**

*__Bold__ numbers are lower than 5% statistically significant

Figure 12: Average number of clusters within all bins.

5.1. Clusters

Figure 12 illustrates the average number of clusters in all bins for the 20 runs under each t_{max} value for the Tan function. This sheds light on the number of distinct behaviours that the framework managed to detect. Interestingly, the number of clusters decreases as the t_{max} value increases. One would expect that when the length of time-series increases the number of possible behaviours would increase as well. However, it is the opposite case as illustrated in Figure 12. When we looked closer at the formed clusters in order to understand this phenomenon (e.g., see Figure 13), we found that the number of projected points in each bin decreases as the value of t_{max} increases (remember that GP collapses each training time-series sample into a 2D data-point, see Section 3.1). This is because we fixed the number of training samples in all experiments to 2000 samples. The framework distributes the training samples on all bins according to their target event positions (see Section 3). Hence, when t_{max} increases the 2000 training samples will be distributed on more bins. This observation prompts an interesting question regarding the ideal training technique that we should use for the framework. Obviously, when t_{max} values increases, one needs more training samples to learn as much as possible about the environment. However, in many real-world applications, historical data may not be easily available. In future work, we will consider the use of sampling and compression techniques to balance the training set to be appropriate for the t_{max} value.

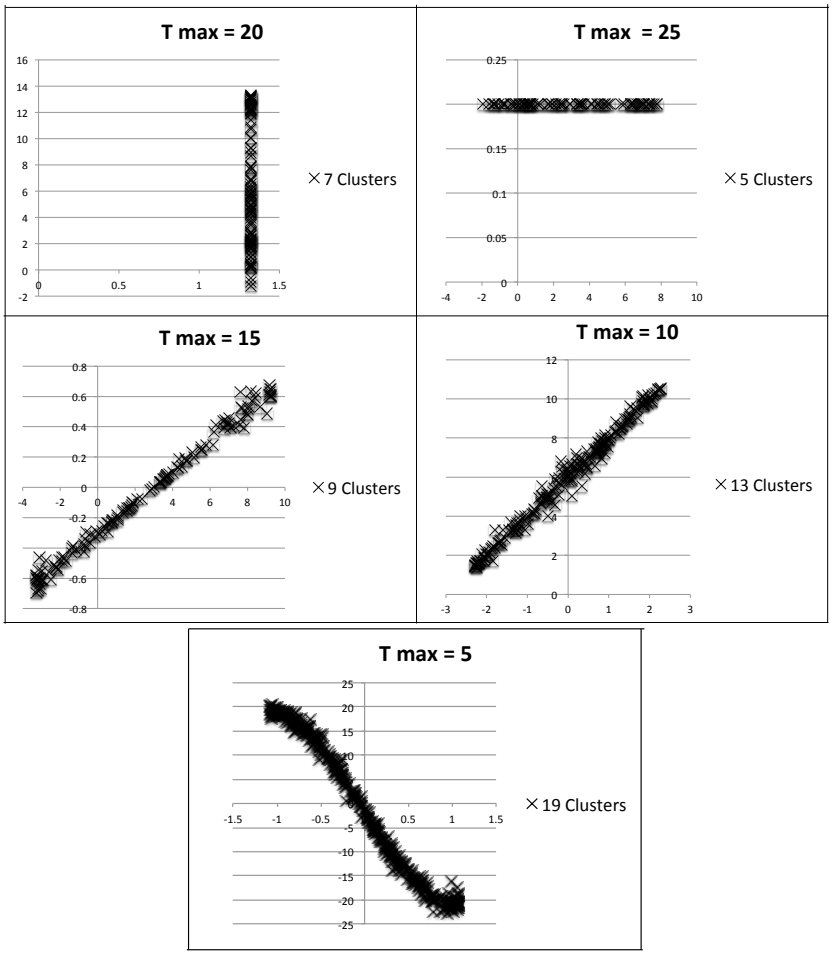

Figure 13: Examples of projected training time-series by GP on 2D space.

5.2. Learn from Evolution

In Figure 14, we show the average proportions of GP functions used in each generation in one of the experimental sets across 20 runs ($t_{max} = 5$ under the Tan function). While GP is known to be sensitive to its inputs (meaning that one node can change a tree's output significantly) it is hard to indicate which functions are most important. However, the figure gives an indication of the evolution's preferences in terms of functions (or sub tree) favoured by the selection process. This can shed some light on the least favoured functions and help to exclude them from the GP primitive set in future experiments, so as to decrease the search space without affecting the solutions' quality.

It is clear from Figure 14 that the four arithmetic operators receive the highest proportions. This is natural because they are located on top of evolved trees where statistical functions are treated as inputs for the arithmetic operators. We noticed that

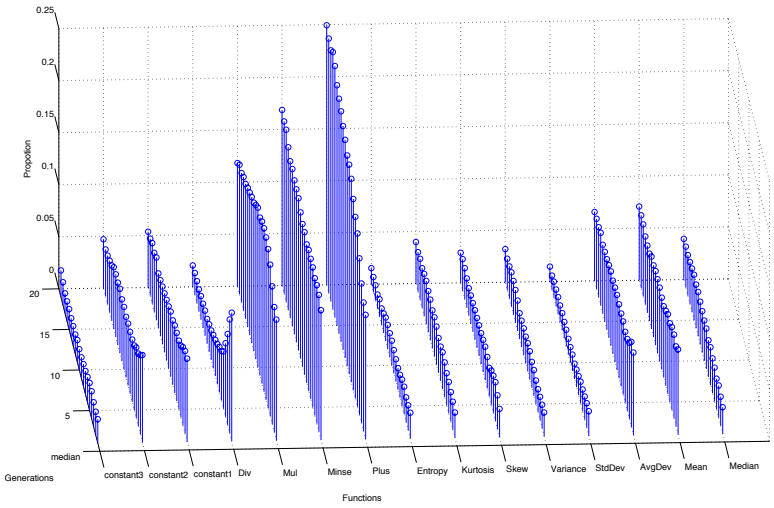

Figure 14: Average proportions of each function used in each generation.
Results collected from 20 runs using $t_{max} = 5$ under the Tan function.

the selection of the *Div* operator decreased gradually as the evolution progresses. Now, looking at the proportions of statistical functions we can see that *Mean* and *AvgDev* (or average deviation) dominate most trees and their proportions increase slowly as the evolution progress. Then, *Median* and *Kurtosis* receiving a considerable attention by evolution. The least favoured, as indicated by Figure 14, is the *StdDev* (or standard deviation).

Because the performance of the evolved programs are good, it would be interesting to know what it actually does and learn from the evolutionary process. Fortunately, most evolved programs are small enough to be understandable as illustrated in Figure 15. For example, we can learn that something as simple as $f(x) = \frac{Entropy(V)}{Average\_Devition(V)} + Skew(V)$ and $f(y) = Median(V)$ can extract statistical features from the set of time-series vectors in such a way that distinguishes their behaviours (i.e., their trends of going up and down).

6. Conclusion and Future Work

In our opinion, the goodness of an algorithm should not be measured only by its results or how far it is from other state-of-the-art techniques; rather, it should also be evaluated by its novelty and how far it will allow other researchers to build on it in order to achieve a truly intelligent system. If an ideal time-series event detector system were to exist, it would return a full prediction of what the environment will generate before it actually generates anything. While this ideal system does not yet exist, in this work, we present a framework which we feel can be one step closer to this system. The framework uses GP to predict the position of target events (defined by the user)

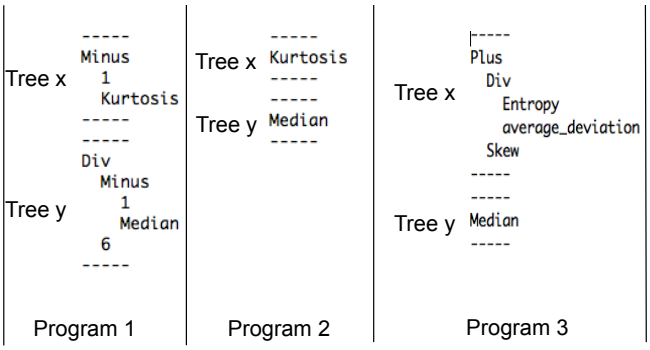

Figure 15: Examples of evolved programs to project time-series into 2D space.

in a time-series. The proposed framework learns the behaviour of the environment (that generates the time-series data) by analysing historical time-series vectors. GP is used to evolve programs that distinguish different behaviours in the training time-series vectors and learns their patterns. Based on the learnt knowledge, it compares the unseen observation-points (coming from the environment) with previously learnt patterns in order to predict the time when the unseen time-series is expected to show the target event.

This framework can be seen as a trial to analyse time-series events from the generating environment's perspective rather than analysing a single time-series. In real-world applications, the environment can be anything, including but not limited to stock markets, buyer-seller negotiations or prices of oiland gas or electricity in international markets. The framework automatically builds a library of of candidate temporal features, using divide and conquer strategy, by separating the training set into different bins based on the exact time t_i of occurrence of the target event (which is defined by the user) and then further classifies each bin's members into statistically independent clusters.

The proposed framework has many potential applications. For example, as shown in the experiments section (see Section 4.4), the proposed framework can be used to analyse time-series data (from Google Trends) of keyword searches on the Internet and predict the next peak so as to assist marketing managers in deciding the best time to release their digital marketing campaigns.

The advantage of the proposed framework is that it divides the training samples into subsets based on their distinguished behaviours automatically without previous knowledge of the problem domain. Moreover, the proposed framework allows the user to define a particular target event of interest.

Although, experimental results on artificially generated data and real-world problem demonstrated the superiority of the framework over a standard GP and RBFN, it is fair to report that the proposed framework suffers from several disadvantages that we would like to address in future research. Firstly, the framework requires large training samples in order to divide them into bins based on the position of the target event (see Section 3.1 for details). Secondly, the framework, in its current realisation, predicts

the target event upon its occurrence. It can not give an early warning prediction of the target event. The framework does not guarantee 100% correctness in its predictions. Instead, it aims to improve the odds in its user's favour. The framework can significantly enhance the user's productivity in finding patterns and monitoring the environment.

The following interesting questions arose from our analysis of the experimental results. We would like to address these issues in future research.

1. We would like to explore the idea of artificially generating training examples (i.e., time-series vectors in our case) using sampling techniques in order to overcome the limitation of requiring a large number of training examples.
2. In this work, we assume a static environment that generates a finite set of behaviours. In the future, we would like to test the system with dynamic environments.

References

[1] A. Agapitos, M. Dyson, J. Kovalchuk, and S. M. Lucas. On the genetic programming of time-series predictors for supply chain management. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, 2008.

[2] A. Agapitos, M. O'Neill, and A. Brabazon. Evolving seasonal forecasting models with genetic programming for pricing weather-derivatives. In *Applications of Evolutionary Computing, EvoApplications 2012: EvoCOMNET, EvoCOMPLEX, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoNUM, EvoPAR, EvoRISK, EvoSTIM, EvoSTOC*, 11-13 Apr. 2011.

[3] J. R. Anderson, R. S. Michalski, R. S. Michalski, T. M. Mitchell, et al. *Machine learning: An artificial intelligence approach*, volume 2. Morgan Kaufmann, 1986.

[4] A. G. Bors. Introduction of the radial basis function (rbf) networks. Technical report, Department of Computer Science, University of York, UK, 2001.

[5] P. Brockwell and R. Davis. *Time series: theory and methods*. springer Verlag, 2009.

[6] H. Cao, L. Kang, Y. Chen, and J. Yu. Evolutionary modeling of systems of ordinary differential equations with genetic programming. *Genetic Programming and Evolvable Machines*, 1(4):309–337, Oct. 2000.

[7] C. Chatfield. *The analysis of time series*. Texts in statistical science. Chapman Hall, London [u.a.], 5. ed edition, 1996.

[8] Y. Chen, B. Yang, Q. Meng, Y. Zhao, and A. Abraham. Time-series forecasting using a system of ordinary differential equations. *Information Sciences*, 181(1):106–114, 2011.

[9] D. Dohare and V. S. Devi. Combination of similarity measures for time series classification using genetic algorithms. In *IEEE Congress on Evolutionary Computation*, pages 401–408. IEEE, 2011.

[10] M. Forouzanfar, A. Doustmohammadi, S. Hasanzadeh, and H. Shakouri G. Transport energy demand forecast using multi-level genetic programming. *Applied Energy*, 91(1):496–503, 2012.

[11] Google. Google insights, June 2012. `http://www.google.com/insights/`.

[12] V. Guralnik and J. Srivastava. Event detection from time series data. In *KDD*, pages 33–42, 1999.

[13] M. Hetland and P. Strom. Evolutionary rule mining in time series databases. *Machine Learning*, 58:107–125, 2005.

[14] D. Jackson. The performance of a selection architecture for genetic programming. In *Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*, volume 4971 of *LNCS*, pages 170–181, Naples, 26-28 Mar. 2008. Springer.

[15] A. Kattan, A. Agapitos, and R. Poli. Unsupervised problem decomposition using genetic programming. In A. I. Esparcia-Alczar, A. Ekrt, S. Silva, S. Dignum, and A. S. Etaner-Uyar, editors, *EuroGP*, volume 6021 of *Lecture Notes in Computer Science*, pages 122–133. Springer, 2010.

[16] A. Kattan, M. Al-Mulla, F. Sepulveda, and R. Poli. Detecting localised muscle fatigue during isometric contraction using genetic programming. In A. D. Correia, A. C. Rosa, and K. Madani, editors, *IJCCI*, pages 292–297. INSTICC Press, 2009.

[17] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, May 1994.

[18] C. Langin. Introduction to data mining. *Scalable Computing: Practice and Experience*, 9(4), 2008.

[19] G. Y. Lee. Genetic recursive regression for modeling and forecasting real-world chaotic time series. In L. Spector, W. B. Langdon, U.-M. O'Reilly, and P. J. Angeline, editors, *Advances in Genetic Programming 3*, chapter 17, pages 401–423. MIT Press, Cambridge, MA, USA, June 1999.

[20] A. Manning. *Monopsony in motion: Imperfect competition in labor markets*. Princeton Univ Pr, 2003.

[21] S. G. Mendivil, O. Castillo, and P. Melin. Optimization of artificial neural network architectures for time series prediction using parallel genetic algorithms. In O. Castillo, P. Melin, J. Kacprzyk, and W. Pedrycz, editors, *Soft Computing for Hybrid Intelligent Systems*, volume 154 of *Studies in Computational Intelligence*, pages 387–399. Springer Berlin Heidelberg, 2008.

[22] A. Moraglio and A. Kattan. Geometric generalisation of surrogate model based optimisation to combinatorial spaces. In *EvoCop*, Lecture Notes in Computer Science. Springer, 2011.

[23] F. Nogales and A. Conejo. Electricity price forecasting through transfer function models. *Journal of the Operational Research Society*, 57(4):350–356, 2005.

[24] S. S. I. of Computer Science. The trading agent competition, June 2012. `http://www.sics.se/tac`.

[25] W. Panyaworayan and G. Wuetschner. Time series prediction using a recursive algorithm of a combination of genetic programming and constant optimization. In A. M. Barry, editor, *GECCO 2002: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference*, pages 101–107, New York, 8 July 2002. AAAI.

[26] J. A. Peacock. Two-dimensional goodness-of-fit testing in astronomy. *Royal Astronomical Society, Monthly Notices*, 202:615–627, 1983.

[27] R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming*. Published via `http://lulu.com`, 2008. (With contributions by J. R. Koza).

[28] R. J. Povinelli and X. Feng. A new temporal pattern identification method for characterization and prediction of complex time series events. *IEEE Trans. Knowl. Data Eng.*, 15(2):339–352, 2003.

[29] M. Pulido, P. Melin, and O. Castillo. Genetic optimization of ensemble neural networks for complex time series prediction. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 202 –206, 31 2011-aug. 5 2011.

[30] J. Rosca, M. P. Johnson, and P. Maes. Evolutionary Speciation for Problem Decomposition, 1996. Available via Citeseer.

[31] R. Ruiz. Review of "experimental methods for the analysis of optimization algorithms", thomas bartz-beielstein, marco chiarandini, lu's paquete, mike preuss. springer, 2010. *European Journal of Operational Research*, 214(2):453–456, 2011.

[32] F. Sepulveda, M. Meckes, and B. Conway. Cluster separation index suggests usefulness of non-motor EEG channels in detecting wrist movement direction intention. In *Proceedings of the 2004 IEEE Conference on Cybernetics and Intelligent Systems*, pages 943–947, Singapore. IEEE Press.

[33] E. Tsang, P. Yung, and J. Li. EDDIE-automation, a decision support tool for financial forecasting. *Decision Support Systems*, 37(4):559–565, 2004.

[34] W. Wang, Z. Hidvégi, and A. Whinston. Shill bidding in multi-round online auctions. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 7–pp. IEEE, 2002.

[35] W. W. S. Wei. *Time series analysis - univariate and multivariate methods.* Addison-Wesley, 1989.

[36] G. M. Weiss and H. Hirsh. Learning to predict rare events in categorical time-series data. Technical report, In Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining, AAAI Press, Menlo Park, CA, 1998.

[37] F. Xie, A. Song, and V. Ciesielski. Event detection in time series by genetic programming. In X. Li, editor, *Proceedings of the 2012 IEEE Congress on Evolutionary Computation*, pages 2507–2514, Brisbane, Australia, 10-15 June 2012.

[38] F. Yang, M. Li, A. Huang, and J. Li. Forecasting time series with genetic programming based on least square method. *Journal of Systems Science and Complexity*, 27(1):117–129, 2014.