Proceedings of the IASTED International Conference
Artificial Intelligence and Applications (AIA 2009)
February 16 - 18, 2009 Innsbruck, Austria

APPLYING MACHINE LEARNING METHODS FOR TIME SERIES FORECASTING

Ben Choi & Raj Chukkapalli
Computer Science, Louisiana Tech University, USA
pro@BenChoi.org

ABSTRACT
This paper describes a strategy on learning from time series data and on using learned model for forecasting. Time series forecasting, which analyzes and predicts a variable changing over time, has received much attention due to its use for forecasting stock prices, but it can also be used for pattern recognition and data mining. Our method for learning from time series data consists of detecting patterns within the data, describing the detected patterns, clustering the patterns, and creating a model to describe the data. It uses a change-point detection method to partition a time series into segments, each of the segments is then described by an autoregressive model. Then, it partitions all the segments into clusters, each of the clusters is considered as a state for a Markov model. It then creates the transitions between states in the Markov model based on the transitions between segments as the time series progressing. Our method for using the learned model for forecasting consists of indentifying current state, forecasting trends, and adapting to changes. It uses a moving window to monitor real-time data and creates an autoregressive model for the recently observed data, which is then matched to a state of the learned Markov model. Following the transitions of the model, it forecasts future trends. It also continues to monitor real-time data and makes corrections if necessary for adapting to changes. We implemented and successfully tested the methods for an application of load balancing on a parallel computing system.

KEY WORDS
Machine Learning, Pattern Recognition, Inductive Inference, Time Series, Markov Model, Artificial Intelligence

1. Introduction

The goal of learning is to record and analyze past observable data in order to build models that can be used to predict future events. In this paper, we describe a strategy on analyzing time series data in order to build a Markov model that can then be used for forecasting. A time series records a variable changing over time. The variable can represent stock price, temperature, or brain wave depending on the applications, such as stock market [5], pattern recognition [24], knowledge discovery [16], and speech recognition [15]. Although time series analysis has been well studied, in this paper, we combine varies methods to form a strategy that can be used for diversified applications.

As an example of application, we implemented and successfully tested our strategy for load balancing on a parallel computing system. Our system learned from past load data and created Markov models that are then used to predict future system loads. With this ability of forecasting, our load balancing method performed 35% better than random method and 34% better than round robin method.

Our strategy is a complete process for learning and forecasting. It consists of methods for learning from past data and methods for using the learned model for forecasting. The ability of forecasting requires that the time series data is not totally random. The assumption, which is not often made explicit, is that the time series data must have some regular or repeatable patterns. To find patterns within the data, we use a change-point detection method to partition a time series into segments. Then, we group similar segments together by clustering. Each group of similar segments is considered as a state for a Markov model. By tracing the time series progressing from one segment to another, we build the transition function for the Markov model.

The learned Markov model is then used for forecasting real-time events. To do that, we first must indentify the current state, which tells us where we are now. Following the learned Markov model, we find a path from current state to the most likely next state, which tells us where we should go next. Since the transition from one state to the next is based on probability, it is possible that we will make the wrong choice even though we always choose the highest probable next state. To address this problem, we continue to monitor real-time data to see whether we made the correct forecast. If we found that we have made a wrong move, we will indentify the correct current state and continue from there.

The remaining of this paper is organized as follows. Section 2 provides the background and outlines the related research. Section 3 describes our methods for learning from past data. Section 4 describes our methods for using learned model for forecasting. Section 5 provides an application example and shows the test results. And, Section 6 provides the conclusion and outlines the future research.

2. Related Research

Time series forecasting analyzes and predicts a variable changing over time [2]. In general, a variable changing over time is denoted as *y(t)*, where the value of *y* is a function of time *t*. In this case, time is continuous. The function *y(t)* can be plotted in a graph using the x-axis for time *t*. To facilitate the analysis by digital computers, time is often converted to discrete by taking a regular sampling. For the discrete case, a time series is represented as a sequence, $Y = (y_0, y_1, y_2, \ldots)$. In this case, time is represented as an integer for the indexing and the sampling rate is not showed. This is one of the simplest cases of time series, where the change of y is only a function of time. There might have other factors causing y to change but those factors are not explicitly taking into account.

We can extend the one variable case of time series into a two variable case where one additional factor is taking into account. This extended time series is represented as a sequence of ordered pairs, $W = ((x_0, y_0), (x_1, y_1), (x_2, y_2), \ldots)$, where *x* represents an input factor, *y* represents the changing variable. In this case, *y* changes depending on the factor *x* and a discrete time that is represented here as the index. This type of time series have an advantage over the simple one variable case in that it can account for one additional factor and such is able to make better predictions. Learning from this extended time series is of course more difficult. The first author had created a learning system for the extended time series [4][3].

Learning from time series data [6] requires techniques for detecting patterns within the data. There have been many techniques for pattern recognition [19][23], all of these techniques have prior knowledge of the patterns that are to be recognized. Discovering patterns is entirely different problem where we have no prior knowledge of patterns that will occur in time series. A very complicated approach [21] was proposed for building hidden Markov model from time series data, where each state of the model represents one pattern in time series. The method defines a time series that is formed over a finite alphabet, was only tested on binary sequence. It, in its present form, is more relevant for finding patterns in Genome sequences and may not be applicable to other time series.

Self-organizing maps [11] were also used to discover patterns, a sliding window is used to segment the time series data and all the similar segments are clustered using the algorithm. As the time complexity of this technique is exponential, they proposed a method that reduces the dimensionality of the data using perceptually important points [11], which reduces dimension by compressing data and ignoring some of the changes in the time series.

Patterns can also be recognized using neural networks [10] working together with hidden Markov model [17], each state of the model represents a part of a pattern in the form of neural network. The method finds the best segmentation of time series by using expectation maximization algorithm together with Viterbi algorithm and the steps involved in finding the patterns using the technique makes it more complicated than what we are about to propose in this paper. The method uses complicated steps in finding the patterns and it tries to find the model that best fits the observed time series.

In this paper, we use our learning and forecasting strategy for load balancing by finding patterns in the past load data. Prediction of load using time series models of past load has not been extensively explored. Success has been reported by [1] and [12] to predict load, while [9] reported good prediction results from autoregressive models which were generated from past load data.

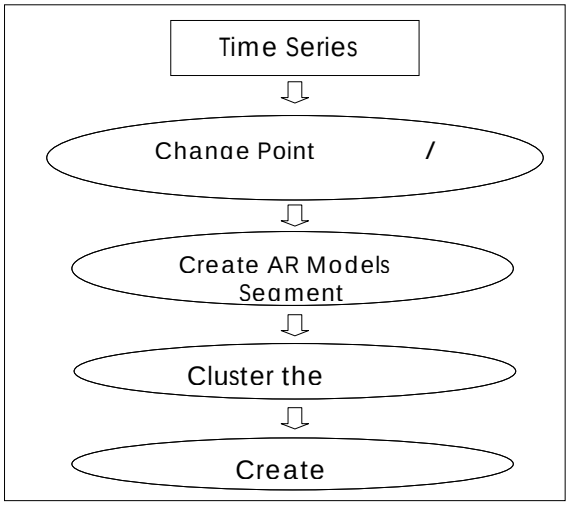

Figure 1. Creating Markov Model from Time Series

3. Learning from Past Data

In this section, we describe our method for learning from past observable time series data. The method consists of four major steps as outlined in Figure 1. Taking past data, the first step uses a change-point detection method for detecting patterns within the data. Based on the detected change points, it partitions the time series into segments. The second step uses autoregressive model to describe each segment. The third step groups similar segments together by clustering method. Considering each of the group as one state, the fourth step creates a Markov model. It builds the transition between states by tracing the transitions from one segment to another as the time series progressing. These four steps are described in details in the following four subsections.

3.1 Detecting Patterns within Data

We developed an algorithm for segmenting a time series, which is an extension of Taylor [22] that is generally used in statistical analysis of financial time charts. The algorithm uses recursive CUSUM statistic [14] to find the change points. CUSUM is cumulative sum of deviation of the current sub group from the initial group and is used to detect shift in the mean value of a measured quantity from the target value.

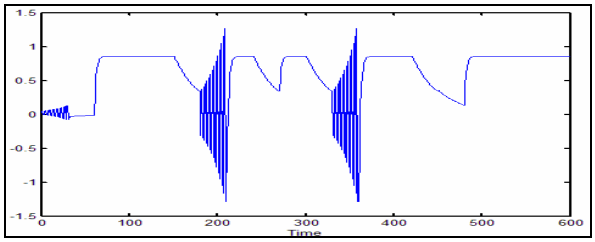

Figure 2. An unusual time Series *T* used as an example

We introduce additional factors that allow our algorithm to be adjustable to different types of time series applications. They also allow us to control the resolution with which the change points are detected. The adaptability of our generalized pattern discovery methodology relies on three parameters: size limit factor, proximity factor, and proximity size factor, as defined below.

Size Limit Factor: *Given a time series* $T = \{t_0, t_1, t_2 \ldots . t_n\}$, *if there is a sub series of T, called* T^1 *and* $Length(T^1) * M = Length(T)$, *then M should not be greater than Sizelimitfactor in order to analyze sub series* T^1 *for any change points.*

Proximity Factor: *If a change point was detected in a time series T and let L and R be the distance of this change point from left and right extremities of the series T respectively, let* $L_R = L / R$ *and* $R_L = R / L$, *and let N be maximum of* L_R *and* R_L, *then N should not be greater than Proximityfactor for that change point to not to be ignored.*

Proximity Size Factor: *Given a time series* $T = \{t_0, t_1, t_2 \ldots . t_n\}$, *if there is a sub series of T, called* T^1 *and* $Length(T^1) * O = Length(T)$, *then a change point is tested for Proximityfactor only if O is greater than Proximitysizefactor.*

The values of these parameters need to be adjusted by the domain expert of the data. If the variance of the time series is high and if the final Markov model generated need to represent even the smaller patterns then *Sizelimitfactor*, *Proximityfactor* and *Proximitysizefactor* need to be increased accordingly. However this adjustment of parameters representing the data domain knowledge, needs to be done only once for that data domain. If *Sizelimitfactor* is increased then the number of change points detected will also increase. *Sizelimitfactor* controls how minute the detected patterns can be. If patterns short in length are considered important for the data being analyzed then *Sizelimitfactor* needs to be increased. *Proximityfactor* controls if small changes at the extremities of a pattern are to be considered separate patterns by themselves. *Proximitysizefactor* was introduced to control *Proximityfactor* so that small changes being detected are small relative to the whole data size.

Our procedure for change point detection is outlined as follows:

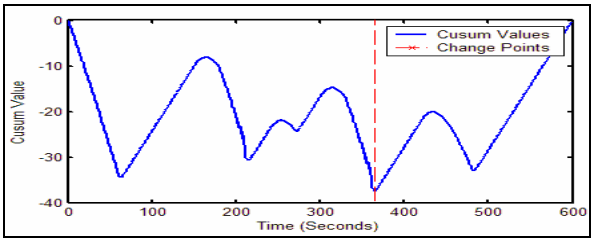

Figure 3. CUSUM chart for first level recursion of time series *T*. Vertical line indicates first change point detected

(1) Let the cumulative sums be denoted by $C = \{c_0, c_1, c_2 \ldots . c_n\}$. Calculate average of series *T*, represented by T_{avg}. Start by setting $c_0 = 0$.
(2) Calculate other cumulative sums by $c_i = c_{i-1} + (t_i - T_{avg})$.
(3) Calculate the difference between maximum and minimum of *C*. Represented by $C_{org\text{-}diff}$.
(4) Perform bootstraps. Calculate of CUSUM of N different shuffles of the series *T*.
(5) Let, *X* = Number of shuffles where C_{diff} is smaller than $C_{org\text{-}diff}$.
(6) After determining that there is a change point in series *T*, we detect the location of that change point in the series. This point will be the one, which is farthest to zero in *C*. In order to ignore small changes at the end of segments, we can ignore a change point if it is very close to one end of the segment. *Proximityfactor* denotes this threshold.
(7) The series is divided in to two parts at the change point and the above procedure is repeated until confidence level drops below 95% (adjustable). If the size of the parts is less than *Sizelimitfactor*, then the part is ignored.

Every change point has a confidence level associated with it. Confidence level of a change point is given as (*X/N*)100%. This level should meet minimum confidence requirement, which is usually set between 90 and 100%. This should be adjusted based on application.

Since not all the data points can be change points, recursion will have to stop at some depth k, thus the complexity of the change point algorithm for average case and worst case is $\Theta(n)$.

As an example to illustrate the above procedure, we show a time series *T* (Figure 2) that is produced from a process switching between three different linear functions. The process switches between three functions, g, *y* and *u*.

```
g(t) = 0.72g(t-1) + 0.24g(t-2)
y(t) = 0.05 - 1.05y(t-1)
u(t) = 0.343 + 0.6u(t-1)
```

Time series *T* (Figure 2) is generated by using the following sequence of functions: "yguuugyugugyuugguuuu". Figure 3 illustrates the Cusum chart for the first level iteration of the above procedure on time series *T*. The red line indicates the change point in the first iteration of the algorithm. Figure 4 shows the detected change points in the time series *T* (Figure 2), after the procedure finished.

3.2 Describing Detected Patterns

After the change points are detected in time series *T*, segments are identified as the time series data between two successive change points. Next, we need to select a model that can best describe the segments. All the segments should be modeled using the same method. In order to select a model, we need the domain knowledge of the time series data. We evaluated various models, such as autoregressive (AR), autoregressive integrated moving average (ARIMA), and neural network. For our example application on load balancing, we choose to use autoregressive model.

Based on the tool provided in Matlab package [20], we generate an autoregressive model to describe each segment. When a segment is modeled using autoregressive model, we use the data in-between change points and exclude the change points themselves. This decision to exclude the change points was taken after testing our algorithm with different data sets and finding that the exclusion of the change points produced better autoregressive models. To select the order of autoregressive model, we used the method suggested in [18] and [20]. The specific measure used to select the order is Schwarz's Bayesian Criterion, which has shown to give good results in our testing. The autoregressive model based [20] is of the form:

$$v(t) = w + a_1 * v(t-1) + \ldots + a_p * v(t-p) + noise(C)$$

where, a_1 to a_p represent the autoregressive coefficients, w represents the mean of segment and p is the order of the autoregressive model. In this modeling, the duration or the length of the segment is not represented in the autoregressive model, but is captured in later step.

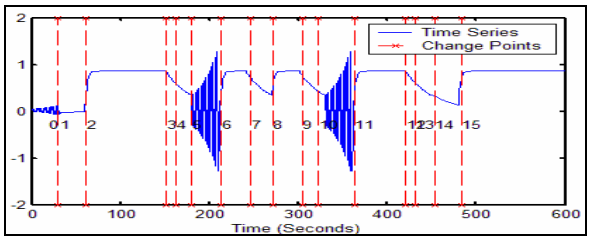

Figure 4. Change points detected for time series *T*, indicated by vertical lines

3.3 Clustering Patterns

After describing all the segments using autoregressive models, we partition the segments into groups by clustering. We implemented a clustering method that measures the similarity between segments by using weighted Euclidean distance [13]. The Euclidean distance, d between two autoregressive models m_1 and m_2 is measured as follows:

$$d = \sum_{i=1}^{p} w_i \left(\alpha_i - \alpha_i' \right)^2$$

where, α_i and α_i' are the i^{th} autoregressive coefficients of models m_1 and m_2; p is the order of the autoregressive model; and w_i is the weight of i^{th} autoregressive coefficient.

$$w_i = \frac{c^i}{K}$$

$$K = \sum_{i=1}^{p} c^i \text{ , where } 0 < c < 1$$

We set the threshold for the similarity between two segments as certain percentage of the median of the distances calculated in-between all the segments. Setting of threshold value for similarity will depend on the domain knowledge. For our application example, we used 20%. Based on the similarity measure and the threshold, we cluster the segments into groups. Each group is represented by a primary autoregressive model (S_{AR}).

3.4 Creating Model to Describe Data

After clustering segments into groups, each group is considered a state for a Markov model. Since each group is represented by a primary autoregressive model (S_{AR}), now each state is associated with the corresponding autoregressive model. For each state, we define the duration of a state (S_D) as the average duration of all the segments in the state. The duration of a state is useful for forecasting described in later section.

Next, we create the transitions between states in the Markov model. We trace the time series progressing in time moving from segment into the next. As the results, it moves from one state to the next. We define $\#(s_m \rightarrow s_n)$ as the number of transitions from state s_m to state s_n in the given time series data, and $\#s_m^t$ represents total number of transition out of state S_m. The transition probability $a_{m,n}$ from state s_m to state s_n is defined as:

$$a_{m,n} = \frac{\#(s_m \rightarrow s_n)}{\#s_m^t}$$

As an example, Figure 5 shows the Markov model generated from time series T (Figure 2).

4. Using Learned Model for Forecasting

In this section, we describe our method for using learned model for forecasting. The method consists of three major steps. The first step monitors real-time data in order to indentify the current state. The second step uses the current state as the starting state and follows the transition of the learned Markov model to forecast future trends. And, the third step continues to monitor real-time data and make correction necessary for adapting to changes. These three steps are described in details in the following three subsections.

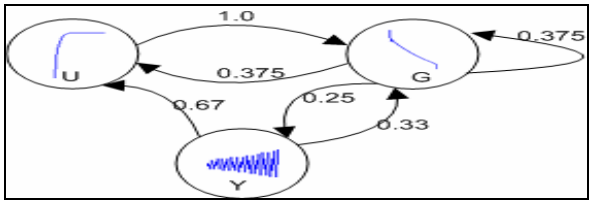

Figure 5. Learned Markov model for the time series T

4.1 Identifying Current State

In order to use the learned Markov model for forecasting, we initially need to identify the current state. This is done by monitoring the real-time data using a moving window. Let W be the duration of the window. We define $W = X * S_w$ where, X is a percentage value that depends on data domain, and S_w is the shortest window size. As the result of our testing, we choose X to be 250% and S_w to be the duration of the shortest segment of the training data. After collecting real-time data for the required duration, we create autoregressive model for the observed data. Then, we find a state closest to this autoregressive model in the learned Markov model. The resulting state is considered as the current state.

4.2 Forecasting Trends

After the current state is found, we use the learned Markov model for forecasting trends. To make the forecast, we start at the current state. We use the autoregressive model (S_{AR}) associated with the current state to generate the forecasted values. We continue to use this autoregressive model for the duration (S_D) of the state. Then, form the current state, we select the highest probability path ($a_{m,n}$) and move to the next state. We use the autoregressive model (S_{AR}) associated with this state to generate the forecasted values for the duration of this state. Then, we move to another next state and so on.

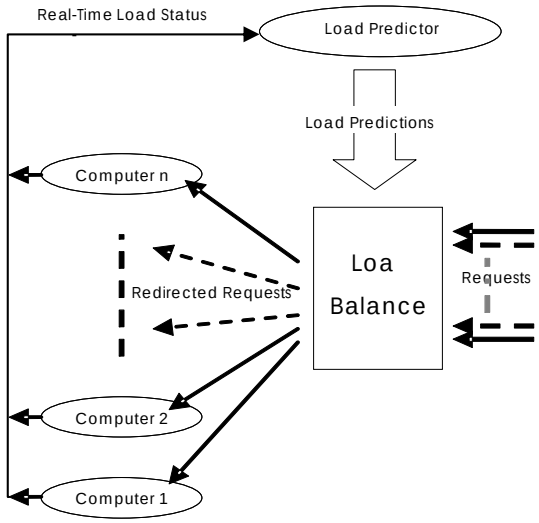

Figure 6. Generalized Load Balancer

4.3 Adapting to Changes

Since the transition from one state to the next is based on probability, it is possible that we will make a wrong move even though we always choose the highest probable next state. To address this problem, we continue to monitor real-time data to see whether we made the correct forecast. We use the moving window (as described in Section 4.1) to capture the real-time data and generate an autoregressive model, which is then matched to the autoregressive model associated with the current state. If there is a match, we had made the right move. Otherwise, we re-indentify the current state and continue from there.

5. Application Example and Test Results

In this section, we provide an application example of using our strategy for learning and forecasting. We also show some test results. We use our strategy to learn form past system load data and then used the learned model to forecast future loads.

Figure 6 shows the general configuration of a load balancer, which directing a request made by an application or user to one of the systems in a homogeneous cluster of systems. A proper load balancing system can significantly increase the performance. If we can predict the load on each of the systems in the cluster, then we can use that information to redirect a request to the system that will have least amount of load in future. This redirection ensures the request is served by a system that will have least amount of load in the future and consequently resulting in faster response to the request. Discovering patterns in load can help us in predicting the load. A periodically monitored load trace of a system can be considered as a time series.

We used load trace data from [7] for testing our load balancer. These trace files contain Unix load taken at 1Hz resolution. We selected two traces taken from two different computers namely Themis and Sahara. These two systems were then considered to be a in a cluster which has to be load balanced. Our segmentation algorithm found 22 statistically significant change points in Themis load trace, which were then grouped into seven clusters. In sahara's load trace, our segmentation algorithm found 28 statistically significant change points, which were then group into nine clusters.

We compared our load balancer with other methods. The results of comparing with round robin and random methods are shown in Table 1. We also compared our prediction accuracy with related work in [8]. As [8] was only used for prediction and not for load balancing, we were only able to compare the RMS errors. The median error in our prediction was 30% better than the median error in [8].

6. Conclusion and Future Research

In this paper, we described a complete process for learning and forecasting. For learning from past data, we extended a change-point detection method by introducing three additional factors, which allow the new method to be tuned for used in varies applications. We used autoregressive model to describe the detected segments and created a clustering method for grouping the segments. We defined each of the groups as a state in a Markov model and defined the duration of a state to be the average duration of all the segments in the state. We then created the transition function for the Markov model by tracing the progressing of the given time series. For using the learned model for forecasting, we used a moving window to monitor real-time data and indentified the current state. We then used the learned Markov model to make forecast. We also outlined a method for adapting to changes and for correcting possible error in the forecast. Then, we implemented and successfully tested our process for an application for load balancing on a parallel computing system.

Future research includes applying the described process for more applications. Other challenging research would be to extend the described process for multi-dimensional time series data, such as two variable case, W = $((x_0, y_0), (x_1, y_1), (x_2, y_2), \ldots)$ [4][3].

Table 1
Results of comparisons

Method	Our Method	Random	Round Robin
% Correct Prediction	84%	49%	50%

References

[1] Basu, S., Mukherjee, A., & Klivansky, S. "Time series models for internet traffic," *Tech. Rep. GIT-CC-95-27*, College of Computing, Georgia Institute of Technology, February 1995.

[2] Brockwell, P.J. & Davis, R.A., *Introduction To Time Series and Forecasting*, Springer-Verlag, New York, 1996.

[3] Choi, B. "Automata for Learning Sequential Tasks," *New Generation Computing: Computing Paradigms and Computational Intelligence*, Vol. 16 No. 1, pp. 23-54, 1998.

[4] Choi, B., "Inductive Inference by Using Information Compression," *Computational Intelligence*, 19 (2), 164-185, 2003.

[5] Clements, M.P., Franses, P.H., & Swanson, N.R. "Forecasting economic and financial time-series with non-linear models," *International Journal of Forecasting*, Volume 20, Issue 2, Pages 169-183, April-June 2004.

[6] Dietterich, T.G., "Machine Learning for Sequential Data: A Review," Book Series *Lecture Notes in Computer Science*, ISSN 0302-9743, Volume 2396, Pages 227-246, 2002.

[7] Dinda, P. "The Statistical Properties of Host Load," *Scientific Programming*, 7:3-4, Fall, 1999.

[8] Dinda, Peter A. "Resource Signal Prediction and Its Application to Real-time Scheduling Advisors," Ph.D. Dissertation. Carnegie-Mellon Univ. Pittsburgh Pa School of Computer Science, May, 2000.

[9] Dinda, Peter A.; Lowkamp, Bruce; Kallivokas, Loukas F.; & O'Hallaron, David R. "The Case for Prediction-based Best-effort Real-time Systems," *Proceedings of the 7th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS 1999)*, pp. 309-318, San Juan, April, 1999.

[10] Firoiu, Laura & Cohen, Paul R. "Segmenting Time Series with a Hybrid Neural Networks - Hidden Markov Model." Eighteenth national conference on Artificial intelligence, pp. 247– 252, Alberta, Canada, 2002

[11] Fu, Tak-chung, Chung, Fu-lai, Ng Vincent, & Luk Robert. "Pattern discovery from stock time series using self-organizing maps," *KDD 2001 Workshop on Temporal Data Mining*, August 2001.

[12] Groschwitz, N. C., & Polyzos, G. C. "A time series model of long-term NSFNET backbone traffic," *In Proceedings of the IEEE International Conference on Communications (ICC'94)* vol. 3, pp. 1400–1404, May 1994.

[13] Kalpakis, Konstantinos; Gada, Dhiral; & Puttagunta, Vasundhara; "Distance Measures for Effective Clustering of ARIMA Time-Series," *In the Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM'01)*, pp. 273-280, San Jose, November 29-December 2, 2001.

[14] Kemp, K.W. "The use of cumulative sums for sampling inspection schemes," *Applied Statistics*, 11, pp.16-31, 1962.

[15] Kobayashi, T., "Speech recognition based on Hidden Markov Models", *Trans Institute of Electrical Engineers of Japan*, 113-C, 5, pp.259-301, May 1993.

[16] Last, M., Klein Y. & Kandel A. "Knowledge Discovery in Time Series Databases," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 31, issue 1, pp. 160-169, Feb 2001.

[17] MacDonald, I.L. & Zucchini, W. *Hidden Markov and Other Models for Discrete-valued Time Series*, ISBN 0412558505, CRC Press, 1997.

[18] Neumaier A. & Schneider. T., "Estimation of parameters and eigenmodes of multivariate autoregressive models," *ACM Trans. Math. Software*, 27, pp. 27–57, 2001.

[19] Perng, C.S., Wang, H., Zhang S.R., & Parker, D.S. "Landmarks: a new model for similarity-based pattern querying in time series databases," *16th International Conference on Data Engineering*, pp. 33 – 4, 29 Feb.-3 March 2000.

[20] Schneider T. & Neumaier A. "Algorithm 808: ARfit - A Matlab package for the estimation of parameters and eigenmodes of multivariate autoregressive models," *ACM Trans. Math. Software*, 27, pp. 58–65, 2001.

[21] Shalizi, Cosma Rohilla; Shalizi, Kristina Lisa; & Crutchfield, James P. "An Algorithm for Pattern Discovery in Time Series," *Journal of Machine Learning Research*, 2002.

[22] Taylor, Wayne A. "Change-Point Analysis: A Powerful New Tool for Detecting Changes," (http://www.variation.com/cpa/tech/changepoint.html), *National Computer*, 2000.

[23] Wang, Q., Liu, J., & Zheng, N.A. "New neural network algorithm for adaptive pattern recognition," *International Conference on Industrial Electronics, Control and Instrumentation (IECON '91)*, vol.2, pp. 1471 – 1473, 28 Oct.-1 Nov. 1991.

[24] Yamato, J., Ohtani, J., & Ishii, K. "Recognizing human action in time-sequential image using Hidden Markov Models," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '92)*, 15-18 June 1992.