Non-Parametric Time Series Classification

Scott Lenser and Maneula Veloso
Carnegie Mellon University
Pittsburgh, PA
{slenser,mmv}@cs.cmu.edu

Abstract

We present a new state-based prediction algorithm for time series. Given time series produced by a process composed of different underlying states, the algorithm predicts future time series values based on past time series values for each state. Unlike many algorithms, this algorithm predicts a multi-modal distribution over future values. This prediction forms the basis for labelling part of a time series with the underlying state that created it given some labelled examples. The algorithm is robust to a wide variety of possible types of changes in signals including changes in mean, amplitude, amount of noise, and period. We show results demonstrating that the algorithm successfully segments signals for a wide variety of example possible signal changes.

1 Introduction

Segmentation of time series into discrete classes is an important problem in many fields. We approach the problem from the field of robotics where time series generated by sensors are readily available. We are interested in using these signals to identify sudden changes in the robot's environment allowing the robot to respond intelligently. For this application, the signal segmentation must be performed in real time and on line, which requires algorithms that are amenable to on-line use. Usually a mathematical model of the process that generates the sensor signal is unavailable as are the number of possible states in this process. Therefore, we focus on techniques that require little a priori knowledge and few assumptions.

In previous work [1, 2], we developed a technique for segmenting a time series into different classes given labelled example time series. In [1], we showed that our algorithm can successfully segment signals from robotic sensors. In this work, we improve on our previous technique by replacing a windowed approach to signal classification with a recursive solution based on a simple HMM.

We have named our new algorithm for classifying time series the Probable Series Classifier (PSC). It is based on a time series prediction component which we will refer to as the Probable Series Predictor (PSP). Unlike many other methods, PSP predicts a *multi-model* probability density over next values. PSC uses several PSP modules to classify a time series into one of a set number of pre-trained states. PSC uses one PSP module per state. Each PSP module is pre-trained from an example time series generated by one state. PSP uses an internal non-parametric model trained from an example time series to make its predictions

PSC runs each PSP module on a time series to be classified and uses the one which best predicts the time series as the classification of the unknown time series.

There has been much interest in time series analysis in the literature due to the broad applicability of time series techniques. There have also been many approaches to time series predictions, most of which are focused on producing a single predicted value. For example, time series prediction has been done using AR, ARMA, IMA, and ARIMA models (e.g. [3]) and neural networks (NNs). All of these techniques produce a single estimated next value in the time series. These techniques can be converted into predicting a distribution over values by assuming a Gaussian deviation around the predicted value. This approach is used by Petridis and Kehagias for NNs [4, 5] and Penny and Roberts for HMM-AR models [6]. A related approach is that of using Gaussian Processes [7] for prediction. Unfortunately, this technique requires the inversion of an nxn matrix which takes $O(n^3)$ time for n observations. In contrast, our approach takes $O(n \log(n))$ time in our current implementation. The chief advantages of our approach over these previous approaches are that we are capable of predicting a multi-modal distribution over values and that our method is amenable to on-line training as more data from a particular state becomes available.

There are a wide variety of algorithms based on change detection, particularly in the domain of fault detection and identification (FDI). These FDI algorithms (e.g. [8, 9]) are usually specialized for the case of two states, one for normal system operation and one for failure cases. Because data can only be collected about the normal state of the system, these algorithms are attempting to solve a different problem and are generally more specialized for this task. We take a very general approach where we detect a wide variety of types of changes to the signal which sets PSC apart from these other techniques.

There has also been a lot of interest in HMMs and switching state-space models, e.g. [6, 10]. These techniques require an a priori knowledge of the underlying structure of the system or extensive off-line training. PSC requires no knowledge about the system structure, as we only require labelled time series examples. PSC also requires negligible training time since almost all of the work is done at query time.

2 Probable Series Classifier Algorithm

Consider a time series of values $\vec{x}_0, \vec{x}_1, \ldots, \vec{x}_t$ created by a generator with k distinct states. At each point in time, one of the states is active and generates the next data value in the time series based upon the previous time series values. Also assume that the frequency of switching between states is relatively low, such that sequential values are likely to be from the same state. We are interested in using the time series of values to recover which state was active at each point in time using only example time series created by each state.

The belief state at time j for state i is the probability of it being active at time j:

$$\begin{aligned} B(s_j = i) &= P(s_j = i | \vec{x}_j, \vec{x}_{j-1}, \ldots, \vec{x}_0) \\ &= \frac{P(\vec{x}_j | \vec{x}_{j-1}, \ldots, \vec{x}_0, s_j = i) * P(s_j = i | \vec{x}_{j-1}, \ldots, \vec{x}_0)}{P(\vec{x}_j | \vec{x}_{j-1}, \ldots, \vec{x}_0)} \end{aligned}$$

We are interested in finding the state i that maximizes this probability. Note that $P(\vec{x}_j | \vec{x}_{j-1}, \ldots, \vec{x}_0)$ is just a normalizing constant and thus doesn't affect which $s = i$ has the maximum likelihood. Furthermore, we will make the m^{th}-order Markov assumption that values $> m$ time steps ago are negligible, given more current readings. This assumption simplifies $P(\vec{x}_j | \vec{x}_{j-1}, \ldots, \vec{x}_0, s_j = i)$ to $P(\vec{x}_j | \vec{x}_{j-1}, \ldots, \vec{x}_{j-m}, s_j = i)$.

$$\begin{aligned}
&P(s_j = i|\vec{x}_{j-1}, \ldots, \vec{x}_0) \\
&= \sum_l P(s_j = i, s_{j-1} = l|\vec{x}_{j-1}, \ldots, \vec{x}_0) \\
&= \sum_l P(s_j = i|s_{j-1} = l, \vec{x}_{j-1} \ldots \vec{x}_0) P(s_{j-1} = l|\vec{x}_{j-1} \ldots \vec{x}_0) \\
&= \sum_l P(s_j = i|s_{j-1} = l) * B(s_{j-1} = l)
\end{aligned}$$

Here we have assumed the current state is independent of old observations (before time j) given the previous state. These assumptions simplify the problem to finding the state i that maximizes the following equations providing a recursive solution:

$$\begin{aligned}
&B(s_j = i) \\
&\propto P(\vec{x}_j|\vec{x}_{j-1}, \ldots, \vec{x}_0, s_j = i) * P(s_j = i|\vec{x}_{j-1}, \ldots, \vec{x}_0) \\
&\approx P(\vec{x}_j|\vec{x}_{j-1}, \ldots, \vec{x}_{j-m}, s_j = i) * P(s_j = i|\vec{x}_{j-1}, \ldots, \vec{x}_0) \\
&= P(\vec{x}_j|\vec{x}_{j-1} \ldots \vec{x}_{j-m}, s_j = i) \sum_l P(s_j = i|s_{j-1} = l) B(s_{j-1} = l)
\end{aligned}$$

This belief update equation is useful for segmentation and classification. Our Probable Series Classifier algorithm uses the update equation for classification by finding the state that maximizes the probability of an unknown time series (using PSP for some key probability calculations). We assume a uniform distribution over the initial state of the generator. We also assume that $P(s_j = i|s_{j-1} = l) = .999$ if $i = l$ and a uniform distribution of the remaining probability over other states. The transition probability does not effect the most likely state at any given time much since the probability is dominated by the sensor readings. Our algorithm runs in real time on a Athlon XP 2700 processing data at 125Hz.

3 Probable Series Predictor Algorithm

We need a prediction of the likelihood of new time series values based upon previous values and the current state i.

$$P(\vec{x}_j|\vec{x}_{j-1}, \ldots, \vec{x}_{j-m}, s_j = i)$$

Note, that state i is known in this case, so we know what state we are predicting for. Assume we have previous time series values generated by this state. We can use these previous examples to generate an estimate at time j given the previous values of the time series. We will focus on the case where $m = 1$ and $\vec{x}$ is a single dimensional value.

We have: a set of value pairs $\vec{x}_i$,$\vec{x}_{i-1}$ and a value at time $j-1$ ($\vec{x}_{j-1}$). We need to generate a probability for each possible $\vec{x}_j$. We can use non-parametric techniques with a locally weighted approach. The problem is visualized in Figure 1. We need to introduce some terminology to more easily discuss the problem.

base value(s) Those value(s) used in generating a predicted value. These are the time series values on which the output is conditioned. In the case of $m = 1$, this is just $\vec{x}_{j-1}$. The conditioning on the state is accomplished by having a separate model for each state.

output value The value output by prediction.

model points Points in base/output space in the training data for a state. These points form the model for this state. Each point is a pair of values: an output value $\vec{x}_j$ and associated base value(s) $\vec{x}_{j-1}, \ldots, \vec{x}_{j-m}$.

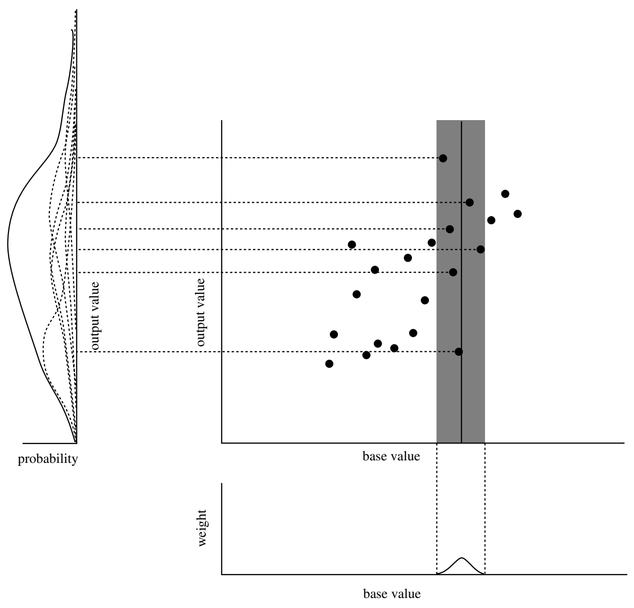

Figure 1: Data prediction. The dots in the main graph show the data available for use in prediction. The grey bar shows the range of values used in the prediction. The bottom graph shows the weight assigned to each model point. The left graph shows the contribution of each point to the predicted probability of a value at time t as dotted curves. The final probability assigned to each possible value at time t is shown as a solid curve.

prediction query A query of the model which provides $\vec{x}_{j-1}, \ldots, \vec{x}_{j-m}$ as input and generates a probability density over $\vec{x}_j$ as output.

We will generate a probability density by generating a weighted set of output value predictions, one from each model point. A kernel is used that assigns more weight to model points with base value(s) near the query base value(s). The predicted output values must then be smoothed to form a continuous probability density.

We use a bandwidth limited kernel over base value(s) to weight model points for speed reasons. The kernel used is the tri-weight kernel:

$$K_t(x,h) = \begin{cases} (1-(x/h)^2)^3 & \text{if } |x/h| <= 1, \\ 0 & \text{otherwise} \end{cases}$$

This kernel is a close approximation to a Gaussian but is much cheaper to compute and reaches zero in a finite bandwidth. The finite bandwidth allows some points to be eliminated from further processing after this step. The bandwidth h is a smoothing parameter that must be selected that controls the amount of generalization performed. From non-parametric statistics, it is known that in order for the prediction to converge to the true function, as $n \to \infty$ (the number of model points), the following two properties must hold: $h \to 0$ and $nh \to \infty$. These properties ensure that each estimate uses more data from a narrower window as we gather more data. We use a ballooning bandwidth for our bandwidth selection. A ballooning bandwidth chooses the bandwidth as a function of the distance to the k^{th} nearest neighbor. Since the average distance between neighbors grows as $1/n$, we choose a bandwidth equal to the distance to the $\sqrt{n}$ nearest neighbor, ensuring that the bandwidth grows as $1/\sqrt{n}$ which satisfies the required statistical properties. Each model point is assigned a weight by the base kernel K_t which is used to scale its prediction in the next stage.

Table 1: Probable Series Predictor algorithm.

```
**Procedure** PredictOutput(*generator_model*,*base_values*)
    **let** $OP \leftarrow$ *generator_model.model_points*
    **let** $D \leftarrow$ dist(*OP.base_values*,*base_values*)
    Choose *base_dist* equal to the $\lceil\sqrt{n}\rceil$th smallest $d \in D$.
    **let** $h_b \leftarrow$ *base_dist* + **noise_base**
    **let** *pred* $\leftarrow \{z.output\_value \mid z \in OP \wedge$
            $\text{dist}(z.base\_values, base\_values) < h_b\}$
    Perform correlation correction on *pred*.
    **let** *base* $\leftarrow \{z.base\_values \mid z \in OP \wedge$
            $\text{dist}(z.base\_values, base\_values) < h_b\}$
    Choose $h_o$ that minimizes $M(h_o)$ over *pred*.
    Return probability density equal to
        $\text{pdf}(z) = \sum_i K_g(pred_i - z, h_o) *$
            $K_t(base_i - base\_values, h_b)$
```

Figure 1 illustrates the PSP algorithm. The dark circles represent model points that have already been seen. The x axis shows the base value. The y axis shows the output value. The dark vertical line shows the query base value. The grey bar shows the range of values that fall within the non-zero range of the base kernel. The graph underneath the main graph shows the weight assigned to each model point based on its distance from the query base value. A prediction is made based on each model point that is simply equal to its output value (we will refine this estimate later). The dotted lines leading from each model point used in the prediction shows these predicted output values. PSP is described in pseudo-code in Table 1.

We need to smooth the predicted output values to get a continuous probability density. We will once again turn to non-parametric techniques and use a tri-weight kernel centered over each point. Because this kernel has a finite bandwidth, it may assign a zero probability to some points. This assignment is undesirable since we never have enough training data to be absolutely sure the data could not have occurred in this state. Hence, we assign a .0001 probability that the time series value is generated from a uniform distribution and a .9999 probability that it is generated according to the estimated distribution.

We need a method for selecting a bandwidth for K_g, the output kernel. We use a modified form of the ballooning method. For each output value prediction, we assign a bandwidth proportional to the distance to the $\sqrt{n}^{\text{th}}$ nearest output value with a minimum bandwidth. We used a proportionality constant of 0.5. We also experimented with selecting a bandwidth using the pseudo-likelihood cross validation measure [11, 12]. This alternative bandwidth selection had similar performance but took about 10 times as long to run.

As exemplified in Figure 1, there is usually a strong correlation between the time series value at time t and the value at time $t-1$. This correlation causes a natural bias in predictions. Model points with base values below the query base value tend to predict an output value which is too low and model points with base values above the query base value tend to predict an output value which is too high. We can correct for this bias by compensating for the correlation between x_t and x_{t-1}. We calculate a standard least squares linear fit between x_{t-1} and x_t. Using the slope of this linear fit, we can remove the bias in the predicted output values by shifting each prediction in both base value and output value until the base value matches the query base value. This process can shift the predicted output value a substantial amount, particularly when using points far from the query base value. This process improves the accuracy of the algorithm slightly. This correlation removal was used in all the tests performed in this paper.

4 Evaluation

We tested the PSC using simulated data, allowing us to know the correct classification ahead of time. It also allowed us to systematically vary different parameters of the signal to see the response of the classification algorithm.

We used PSC to segment a signal generated from a 2 state generator. One state was a fixed baseline signal. The other state was a variation of the baseline signal formed by varying one parameter of the signal. The algorithm was graded on its ability to correctly label segments of the signal that it hadn't been trained on into the 2 states. We tested the performance of PSC by varying the following test parameters:

Training window size The number of data points used to train on each state. The smaller the window size the harder the problem.

Parameter value The value of the parameter used in the modified signal. The closer the parameter to the baseline value the harder the problem.

Parameter modified The parameter chosen for modification. We tested changes to the following parameters: mean, amplitude/variance, observation noise, and period.

For each test we generated a time series with 4000 data points. The baseline state (class 1) generated the first and third quarter of the data signal and the modified state (class 2) generated the second and fourth quarter of the data. We chose to use a sine wave for the underlying signal for the baseline. We added uniform observation noise to this underlying signal to better reflect a real world situation. The signal is parameterized by amplitude, mean, period, and noise amplitude resulting in 4 parameters. The standard signal used an amplitude of $5 * 10^5$, a mean of 0, a period of 20 observations, and a noise amplitude of $5 * 10^4$, resulting in $\pm 10\%$ noise on each observation relative to the signal range.

For each test, we trained a model of each state on an example signal from that state. This example signal was not included in the testing data. The length of the example signal (training window) was varied from 5 data points to 100 data points to show the effect on the results. We tested with 14 different example signals for each state and averaged the results. For each test, we calculated the fraction of time the most likely state matched the actual state of the system. This fraction is reported on all of the result figures. A value of 1 indicates a perfect segmentation of the test segments into classes. A value of 0.5 is the performance expected from randomly guessing the class. This metric equals the probability of getting the correct labelling using a random training example signal for each class.

We summarized our test results in a series of figures. The y axis shows the fraction of correct labellings achieved by PSC. In each figure, there is a point at which the performance of PSC falls to chance levels (0.5). These points occur where the states for class 1 and class 2 have the same parameters and are hence generate the same signal. Since a signal cannot be segmented from itself, we expect the algorithm's performance to fall to chance levels at these points. We gathered results pertinent to applications where a constant amount of training data is available as would occur in on-line labelling applications.

Figure 2 shows the performance of PSC with respect to changes in the amplitude of the signal. PSC performs quite well even for small training window sizes. This type of change to the signal would not be detected by algorithms that are only sensitive to mean shifts as a change in amplitude does not change the mean but only the variance.

Figure 3 shows the performance of PSC with respect to mean shifts. PSC performs well and is able to detect small mean shifts. This type of change to the signal can be detected by algorithms capable of detecting mean shifts. Many mean shift algorithms would have trouble with a periodic signal like this sine wave, though, unless the data was averaged over a full period which would slow detection time.

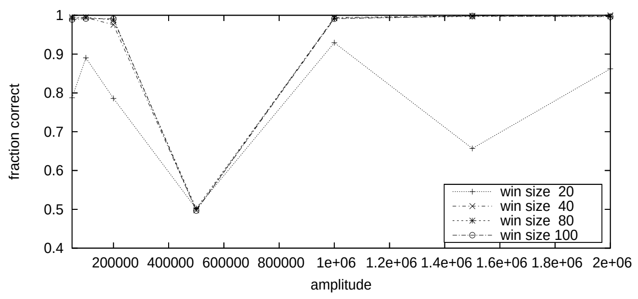

Figure 2: Detection of changes to the signal amplitude with a fixed training window size. The x axis shows the factor by which the amplitude was multiplied.

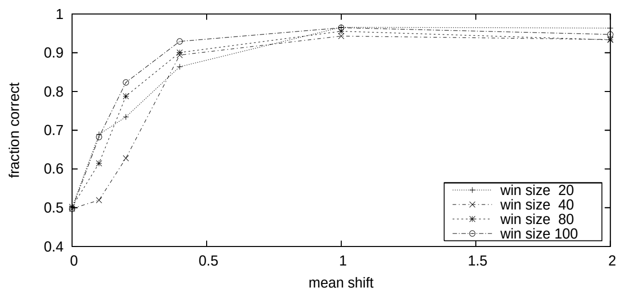

Figure 3: Detection of changes to the signal mean with a fixed training window size. The x axis shows the mean shift as a fraction of the signal amplitude.

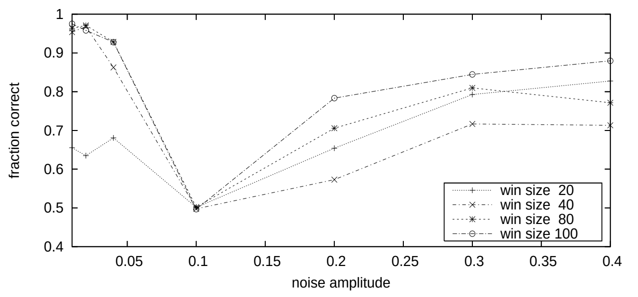

Figure 4: Detection of changes to the observation noise with a fixed training window size. The x axis shows the observation noise as a fraction of the signal amplitude.

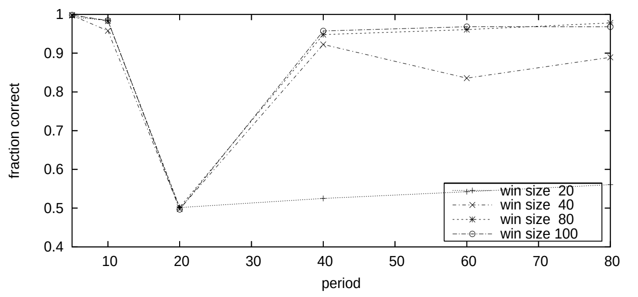

Figure 5: Detection of changes to the period with a fixed training window size. The x axis shows the period of the signal.

Figure 4 shows the performance of PSC with respect to changes in observation noise. This type of change is very difficult to detect as it produces no mean shift in the signal and a negligible variance change. Nevertheless, PSC is still able to detect this type of change effectively. Many algorithms fail to detect this kind of change to a signal.

Figure 5 shows the performance of PSC in detecting changes in the period of the signal. PSC performs well at detecting a change of the period most of the time. PSC fails to detect the change in the period for the case where the period is longer than the training sequence but in this case the training signal is not fully representative of the full signal for the state.

5 Conclusion

We have presented an algorithm for generating predictions of future values of time series. We have shown how to use that algorithm as the basis for a classification algorithm for time series. We proved through testing that the resulting classification algorithm robustly detects a wide variety of possible changes that signals can undergo including changes to mean, variance, observation noise, period, and signal shape. The algorithm runs in real-time and is amenable to on-line training.

References

[1] Scott Lenser and Manuela Veloso. Classification of robotic sensor streams using non-parametric statistics. In *it To appear: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS)*, 2004.

[2] Scott Lenser and Manuela Veloso. Automatic detection and response to environmental change. In *Proceedings of ICRA-2003*, 2003.

[3] Kan Deng, Andrew Moore, and Michael Nechyba. Learning to recognize time series: Combining arma models with memory-based learning. In *IEEE Int. Symp. on Computational Intelligence in Robotics and Automation*, volume 1, pages 246–250, 1997.

[4] V. Petridis and A. Kehagias. Modular neural networks for MAP classification of time series and the partition algorithm. *IEEE Transactions on Neural Networks*, 7(1):73–86, 1996.

[5] V.Petridis and A.Kehagias. *Predictive modular neural networks: applications to time series*. Kluwer Academic Publishers, 1998.

[6] William Penny and Stephen Roberts. Dynamic models for nonstationary signal segmentation. *Computers and Biomedical Research*, 32(6):483–502, 1999.

[7] A. Girard, C. E. Rasmussen, J. Quionero-Candela, and R. Murray-Smith. Multiple-step ahead prediction for non linear dynamic systems — a gaussian process treatment with propagation of the uncertainty. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15, pages 529–536. MIT Press, 2003.

[8] Michèle Basseville and Igor Nikiforov. *Detection of Abrupt Change - Theory and Application*. Prentice–Hall, Englewood Cliffs, N.J., 1993.

[9] Masafumi Hashimoto, Hiroyuki Kawashima, Takashi Nakagami, and Fuminori Oba. Sensor fault detection and identification in dead-Reckoning system of mobile robot: Interacting multiple model approach. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS 2001)*, pages 1321–1326, 2001.

[10] Zoubin Ghahramani and Geoffrey E. Hinton. Switching state-space models. Technical report, 6 King's College Road, Toronto M5S 3H5, Canada, 1998.

[11] J. D. F. Habbema, J. Hermans, and K. van den Broek. A stepwise discrimination analysis program using density estimation. In *Proceedings of Computational Statistics (COMPSTAT 74)*, 1974.

[12] R. P. W. Duin. On the choice of smoothing parameters of Parzen estimators of probability density functions. In *Proceedings of IEEE Transactions on Computers*, volume 25, pages 1175–1179, 1976.