Time Series Classification Using Non-Parametric Statistics

keywords: time series, prediction, classification, non-parametric

Abstract

We present a new class-based prediction algorithm for time series. Given time series produced by different underlying generating processes, the algorithm predicts future time series values based on past time series values for each generator. Unlike many algorithms, this algorithm predicts a distribution over future values. This prediction forms the basis for labelling part of a time series with the underlying generator that created it given some labelled examples. The algorithm is robust to a wide variety of possible types of changes in signals including mean shifts, amplitude changes, noise changes, period changes, and changes in signal shape. 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. By identifying these changes in the sensor signals, the robot can intelligently respond to changes in its environment as they occur. For this application, the signal segmentation must be performed in real time and on line. Therefore, we are focused on algorithms which are amenable to on-line use. Also, usually mathematical models of the processes that generate the sensor signal are unavailable as are the number of possible generating processes. Therefore, we are focused on techniques that require very little a priori knowledge and very few assumptions. In particular, we are focused on techniques where the number of generating processes (or classes of signals) is unknown in advance and where generator models for each class are unavailable.

In previous work (Lenser & Veloso, 2003), Lenser and Veloso developed a technique for segmenting a time series into different classes given labelled example time series. In that work, they broke the time series into windows and used distance metrics over probability densities to determine from which class each window was generated. In this work, we improve on their technique by allowing for smaller window sizes, putting the segmentation on a strong probabilistic foundation, and taking into account the conditional dependence of time series points on points in the recent past.

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). It generates predictions based upon an internal non-parametric model which is trained from an example time series. PSP uses this model and the most recent values from a time series to predict the future values that are likely to be seen. PSP is typically used to predict the next value in a running time series based on recent observed values, a new observation is obtained, and the process is repeated. Unlike many other methods, PSP does not predict a single next value for the time series, but instead predicts a *probability density* over next values. Because of the way this prediction is done, PSP is capable of making multi-model predictions which is important in order to represent realistic time series. PSC uses several PSP modules to classify a time series into one of a set number of pre-trained generator classes. PSC uses one PSP module per generator class. Each PSP module is pre-trained from an example time series generated by one of the generator classes. PSC runs each PSP module on a time series to be classified and uses the one which best predicts the time series to classify 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

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

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. (Deng et al., 1997)). All of these techniques produce a single estimated next value in the time series. In contrast, we generate a distribution over next values. These ARIMA models are also not class-based, which makes them better suited for time series produced by a single underlying process or underlying processes that vary continuously. PSC, on the other hand, is tuned for situations where the time series is produced by a set of *discrete* underlying processes. These differences make these other algorithms suited for a different class of problems than PSC.

There are also a wide variety of algorithms based on classes, particularly in the domain of fault detection and identification (FDI). These FDI algorithms (e.g. (Basseville & Nikiforov, 1993; Hashimoto et al., 2001)) are usually specialized for the case of two classes, one which represents normal operation of the system and one which represents a failure. Because it is difficult to gather data on failure cases, these algorithms focus on confirming/denying the hypothesis that the system is working correctly. This focus results in algorithms that have a one-sided test where the decision of working/failure is based entirely on the properties of the normal case which results in less knowledge being needed at the cost of some resolution power. Also, most of the algorithms are further specialized for detecting particular types of changes in the signal. Detecting mean shifts in the signal is a particularly common specialization while other algorithms specialize in variance changes.

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. (Penny & Roberts, 1999; Ghahramani & Hinton, 1998). These techniques require an a priori knowledge of the underlying structure of the system, which is not available for the robotic signals we are interested in. PSC does not require as much knowledge about the system structure, as we only require labelled time series examples.

2. Probable Series Classifier Algorithm

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

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

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

We take a maximum likelihood approach, and are interested in finding c_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 c_i has the maximum likelihood. Furthermore, we will make the simplifying assumption that the effective information found in time series values more than m time steps in the past is negligible, given more current readings. This assumption simplifies $P(\vec{x}_j|\vec{x}_{j-1}, \ldots, \vec{x}_0, c_{i,j})$ to $P(\vec{x}_j|\vec{x}_{j-1}, \ldots, \vec{x}_{j-m}, c_{i,j})$.

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

Here we have assumed that $c_{i,j}$ is independent of observations before time j given $c_{l,j-1}$ for all l.

These assumptions simplify the problem to finding the c_i that maximizes the following equations providing a recursive solution:

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

This belief update equation is useful for segmentation and classification. Our Probable Series Classifier algorithm uses the update equation for classification by finding the generator class that maximizes the probability of an unknown time series (using PSP for some key probability calculations). The probability of the

unknown time series of length w for a generator c_i can be calculated using the following equations, where we assume that $P(c_{i,j}|c_{l,j-1}) = 0$ for $i \neq l$ and the initial beliefs over all generator classes are equal.

$$
\begin{aligned}
& B(c_{i,j}) \\
& \propto P(\vec{x}_j|\vec{x}_{j-1}\ldots\vec{x}_{j-m}, c_{i,j}) \sum_l P(c_{i,j}|c_{l,j-1}) B(c_{l,j-1}) \\
& = P(\vec{x}_j|\vec{x}_{j-1}, \ldots, \vec{x}_{j-m}, c_{i,j}) * B(c_{i,j-1}) \\
& = B(c_{i,j-w}) \prod_{l=j-(w-1)}^{i} P(\vec{x}_l|\vec{x}_{l-1}, \ldots, \vec{x}_{l-m}, c_{i,l}) \\
& \propto \prod_{l=j-(w-1)}^{i} P(\vec{x}_l|\vec{x}_{l-1}, \ldots, \vec{x}_{l-m}, c_i)
\end{aligned}
$$

3. Probable Series Predictor Algorithm

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

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

Note, that c_i is known in this case, so we know what generator we are predicting for. Assume we have previous time series values generated by this generator. 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 $\vec{x}_i$,$\vec{x}_{i-1}$
- 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 generator is accomplished by having a separate model for each generator.

output value The value output by prediction.

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

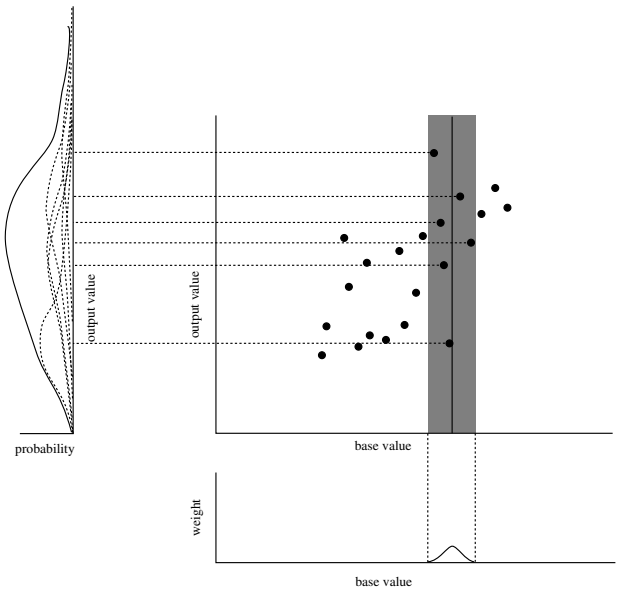

Figure 1. Data prediction. The dots in the main graph show the data available for use in prediction. The grey bar shows the range of values used in the prediction. The bottom graph shows the weight assigned to each model point. The left graph shows the contribution of each point to the predicted probability 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.

model point base value(s) The base value(s) in a model point.

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.

query base value(s) The base value(s) in the prediction query.

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

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
    $\text{pdf}(z) = \sum_i K_g(pred_i - z, h_o) *$
    $K_t(base_i - base\_values, h_b)$

Table 1. Probable Series Predictor algorithm.

that controls the amount of generalization performed. We need to select a bandwidth h for this kernel. 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 $n * h \to \infty$. 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. We add a small constant a_n to this bandwidth to ensure that a non-zero number of points have non-zero weight. This constant is chosen equal to the minimum amount of base value change that is considered meaningful. Each model point is assigned a weight by the base kernel K_t which is used to scale its prediction in the next stage.

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 Gaussian kernel centered over each point. A Gaussian kernel is used because it assigns a non-zero probability to every possible outcome. If we chose a bandwidth limited kernel, we would have locations with zero probability. These zero probability regions would make entire sequences have zero probability for some classes, a form of overfitting. The Gaussian kernel used is:

$$K_g(x,h) = \frac{1}{h\sqrt{2\pi}} * e^{-(x/h)^2/2}$$

We need a method for selecting a bandwidth for K_g, the output kernel. We can't reuse the ballooning method because it would result in an invalid probability density function which changes as you query it. This output bandwidth can be found by a simple search if we first develop a metric for determining the quality of a bandwidth. The error we are interested in is the ratio of predicted probabilities to actual probabilities, not the distance between these two probabilities. We chose to use the pseudo-likelihood cross validation measure (Habbema et al., 1974; Duin, 1976). This method is known to minimize the Kullback-Leibler distance between the estimated probability densities and the actual probability density (for many classes of probability densities (Hall, 1987)). The Kullback-Leibler distance $(\int f(x) * \ln(f(x)/g(x))dx)$ has a strong dependence on the ratio of the two probability densities. The pseudo-likelihood cross validation method maximizes the likelihood of the data predicting itself over all possible bandwidths. We use leave-one-out cross-validation where each point is excluded from its own prediction. The pseudo-likelihood cross validation measure is defined as:

$$M(h) = \prod_i \sum_{j \neq i} K_g(\frac{x_i - x_j}{h}, h)$$

PSP does a search over all possible bandwidths starting from one corresponding roughly to expected measurement noise and ending with one corresponding to the range of possible values. The search is done by starting at the low value and increasing the bandwidth each time by a constant *factor*. The bandwidth with the maximum pseudo-likelihood cross validation measure is chosen as the bandwidth.

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

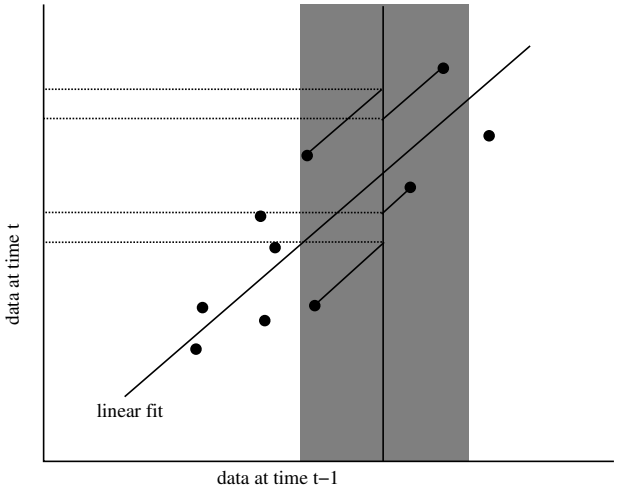

Figure 2. Correlation removal. A linear fit to the model points (shown as dots) is shown. The grey vertical bar shows the range of values actually used in the prediction. The short solid lines show the effect of shifting the model points to match the base value of the query while taking into account the correlation. The hollow squares show the predicted output values.

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 is shown in Figure 2, where we can see that the predicted output value can shift a substantial amount, particularly when using points far from the query base value. 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.

4.1. Methodology

We used PSC to segment a signal generated from 2 classes. One generator class was a fixed baseline signal. The other generator 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 classes. 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 generator. The smaller the window size the harder the problem.

Testing window size The number of data points used to classify each unknown time series segment. The smaller the window 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
- Period
- Signal Shape

For each test we generated a time series with 4000 data points. The baseline generator(class 1) generated points 0–999,2000–2999 and the modified generator(class 2) generated points 1000–1999,3000–3999. 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. We did not try adding process noise. The signal is parameterized by amplitude, mean, period, and noise amplitude resulting in 4 parameters. In addition, we ran some tests where the modified signal was generated from a triangle wave with parameters as per the sine wave as shown in Figure 3. The standard signal used an amplitude of $5 * 10^5$, a mean of 0, a period of 20 observations, and a noise amplitude of $5*10^4$. This results in $\pm 10\%$ noise on each observation point relative to the range of the signal.

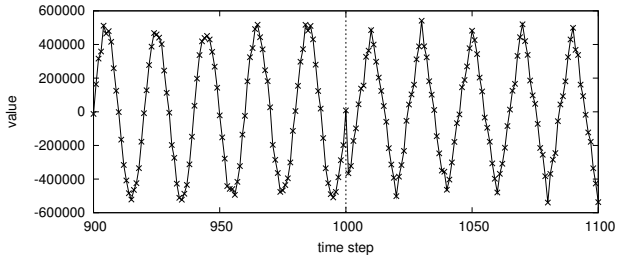

Figure 3. Example signal. The first half of the figure shows the baseline sine wave and the second half shows the triangle wave.

For each test we subdivided the 4000 data points into 100 data point long segments. We used segments starting at $0, 200, 400, \ldots, 3800$ for training models of classes. We used segments starting at $100, 300, 500, \ldots, 3900$ for testing the ability of the

models to correctly label new data. For each possible combination of testing segment, training segment for baseline signal (class 1), and training segment for modified signal (class 2), we evaluated the likelihood of the test segment coming from class 1 or 2 based upon the trained models. If the more likely class matched the actual class for the test segment, we counted that trial a success. We used the fraction of correct labellings as the metric for evaluating the performance of the algorithm. This metric is shown in most of the graphs. 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 segment for class 1, a random training segment for class 2, and a random testing segment.

4.2. Results

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 generators for class 1 and class 2 have the same parameters and are hence 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 considered two main usage scenarios for the availability of training/testing data. In the first scenario, we considered applications where the training and testing windows are of the same size as might occur in clustering applications. In the second scenario, we considered applications where a constant, large amount of training data is available as would occur in on-line labelling applications.

4.2.1. Equal Size Training and Test Windows

Figure 4 shows the performance of PSC with respect to changes in the amplitude of the signal. When the signals are the same, the performance drops to the chance level of 0.5. The rest of the time, PSC performs quite well even for extremely small window sizes. With a window size of just 100 data points, PSC is successfully able to segment between even very small changes of amplitude. Even with only 5 data points for training and another 5 for testing, PSC correctly labels most of the test sequences. Using 5 data points corresponds to using data from only 1/4 of the period of the signal. PSC also matches intuition well in performing better for larger training/testing window sizes and larger changes in the signal. 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.

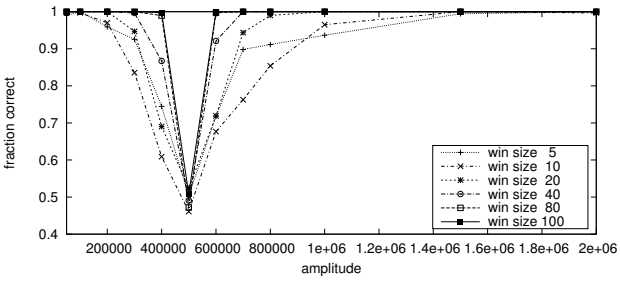

Figure 4. Detection of changes to the signal amplitude with equal sized training and testing windows. The x axis shows the factor by which the amplitude was multiplied.

Figure 5 shows the performance of PSC with respect to mean shifts. PSC performs well and is able to detect small mean shifts. Detecting mean shifts requires more data points than some of the other changes, as multiple periods worth of data are desirable to confirm a mean shift. 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.

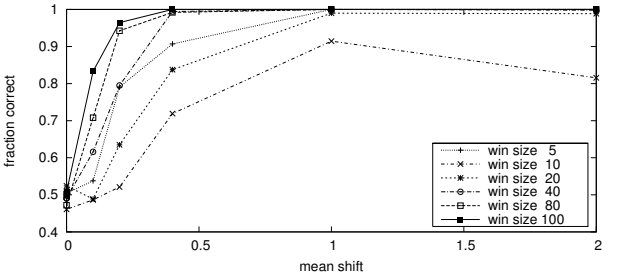

Figure 5. Detection of changes to the signal mean with equal sized windows. The x axis shows the mean shift as a fraction of the signal amplitude.

Figure 6 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 very effectively. With a window size of 100, PSC almost perfectly segments the data for a small 2x change in noise. We are not aware of any other algorithms that can detect this kind of change to a signal.

Figure 7 shows the performance of PSC in detecting changes in the period of the signal. PSC performs well at detecting a shortening of the period for a wide

range of window sizes, with larger windows providing better performance. PSC also performs well at detecting lengthening of the period for the larger window sizes.

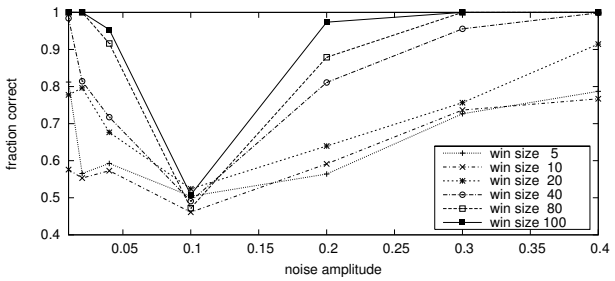

Figure 6. Detection of changes to the observation noise with equal sized windows. The x axis shows the observation noise as a fraction of the signal amplitude.

The performance at detecting lengthening of the period for small window sizes is erratic and often poor. This poor performance is caused by a misalignment between the part of the period of the signal used for training and the part used for testing. Since the training/testing windows for the smaller window sizes are only a small part of one period, it is often the case that the training window and the testing window come from different parts of the signal period. Naturally, the predictions during the testing phase are poor when tested on a part of the signal that was never seen during training. This effect accounts for the poor performance of PSC for long periods and short window sizes. PSC performs well in the cases where the training data and testing data contain the same parts of the signal. Usually, algorithms which detect period changes are highly specialized to period changes and are incapable of detecting other changes.

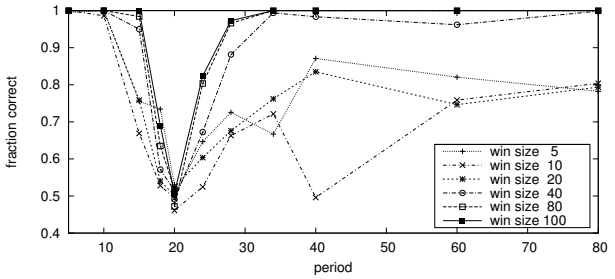

Figure 7. Detection of changes to the period with equal sized windows. The x axis shows the period of the signal.

4.2.2. Constant Size Training Windows

Figures 8, 9, 10, and 11 show the equivalent tests to Figures 4, 5, 6, and 7 respectively, except that in all cases the training window size is fixed at 80. As can be

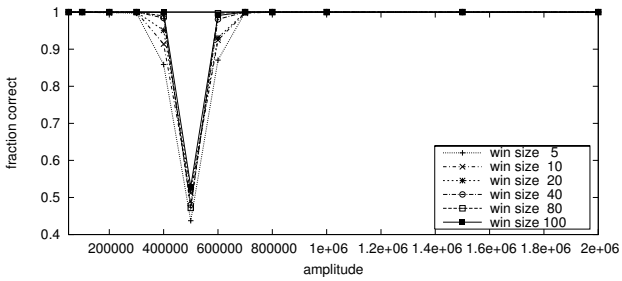

Figure 8. Detection of changes to the signal amplitude with a fixed training window size. The x axis shows the factor by which the amplitude was multiplied.

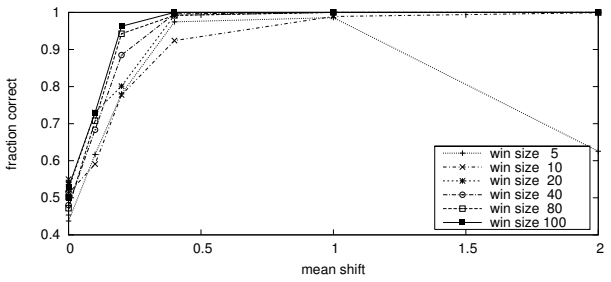

Figure 9. Detection of changes to the signal mean with a fixed training window size. The x axis shows the mean shift as a fraction of the signal amplitude.

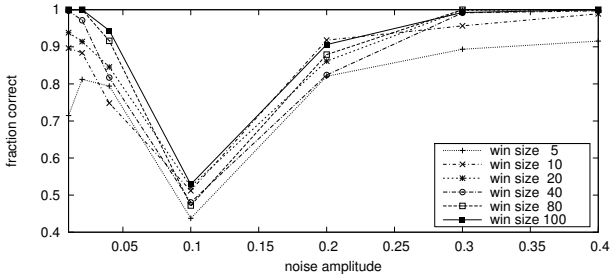

Figure 10. Detection of changes to the observation noise with a fixed training window size. The x axis shows the observation noise as a fraction of the signal amplitude.

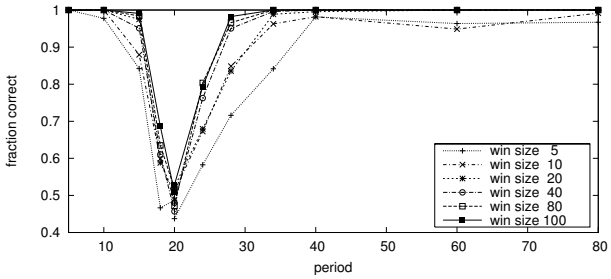

Figure 11. Detection of changes to the period with a fixed training window size. The x axis shows the period of the signal.

seen from the graphs, the extra training data results in an across the board improvement in the performance of PSC. The graphs show that even very small window sizes are sufficient for detecting most signal changes. The most dramatic improvement is in detection of period changes. With the longer training window size, small testing window sizes are now sufficient for detecting changes in the period of the signal. This improvement is caused by the guarantee of the testing window overlapping the same phase of the signal as at least some of the training window. PSC correctly selects the relevant part of the training data and can detect even small changes in period.

Figure 12 shows the ability of PSC to segment signals based upon the shape of the signal. We tested using two classes with identical parameters to the base signal except that for class 2 the signal is based upon a triangle wave instead of a sine wave. The y axis shows the difference in the log probability of the two classes for a window of 25 data points. As can be seen in the graph, the most likely class is also the correct class in the vast majority of cases. We trained PSC on a window of 100 data points for each class. We also ran the same experiment with a few different training window sizes and testing window sizes. In all cases, we observed that increasing the amount of available data resulted in larger margins in the classification (results not shown).

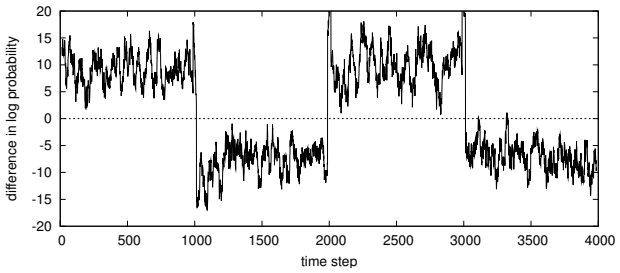

Figure 12. Segmentation between a sine wave and a triangle wave. The y axis shows the log probability of the sine wave minus the log probability of the triangle wave. A value above 0 indicates the signal is probably a sine wave while a value below 0 indicates that a triangle wave is more likely. Each data point shows the difference in log probability for the two possible signals based on a window of 25 data points centered at the x coordinate of the point. The actual signal was a sine wave from times 0–999 and 2000-2999 and a triangle wave the rest of the time.

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 has the nice property that the performance of the algorithm improves when more training data is available, when more data is available before a class label must be chosen, and when the difference between the signals becomes greater. The algorithm can be used to replace a collection of algorithms tuned to detecting particular changes in signals with one algorithm which can detect any change to the signal.

References

Basseville, M., & Nikiforov, I. (1993). *Detection of abrupt change - theory and application*. Englewood Cliffs, N.J.: Prentice–Hall.

Deng, K., Moore, A., & Nechyba, M. (1997). Learning to recognize time series: Combining arma models with memory-based learning. *IEEE Int. Symp. on Computational Intelligence in Robotics and Automation* (pp. 246–250).

Duin, R. P. W. (1976). On the choice of smoothing parameters of Parzen estimators of probability density functions. *Proceedings of IEEE Transactions on Computers* (pp. 1175–1179).

Ghahramani, Z., & Hinton, G. E. (1998). *Switching state-space models* (Technical Report). 6 King's College Road, Toronto M5S 3H5, Canada.

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

Hall, P. (1987). On Kullback-Leibler loss and density estimation. *The Annals of Statistics* (pp. 1491–1519).

Hashimoto, M., Kawashima, H., Nakagami, T., & Oba, F. (2001). Sensor fault detection and identification in dead-Reckoning system of mobile robot: Interacting multiple model approach. *Proceedings of the International Conference on Intelligent Robots and Systems (IROS 2001)* (pp. 1321–1326).

Lenser, S., & Veloso, M. (2003). Automatic detection and response to environmental change. *Proceedings of ICRA-2003*.

Penny, W., & Roberts, S. (1999). Dynamic models for nonstationary signal segmentation. *Computers and Biomedical Research*, *32*, 483–502.