Classification of Robotic Sensor Streams Using Non-Parametric Statistics

Scott Lenser and Manuela Veloso
Computer Science Department
Carnegie Mellon University
Pittsburgh, USA
Email: slenser,mmv@cs.cmu.edu

***Abstract*— We extend our previous work on a classification 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 improve upon the speed of our previous approach and show the utility of the algorithm for discriminating between different states of the robot/environment from robotic sensor signals.**

I. 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 [1], we developed a technique for segmenting a time series into different classes given labelled example time series. In that work, we broke the time series into windows and used distance metrics over probability densities to determine from which class each window was generated. In subsequent work [2], we introduced a new algorithm that allows for smaller window sizes, puts the segmentation on a strong probabilistic foundation, and takes into account the conditional dependence of time series points on points in the recent past. In this work, we improve on the speed of the algorithm. We also show that the newer algorithm works for a wide variety of signals generated from robotic sensors.

We have named the 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 likely future time series values. PSP is typically used to predict the next value in a running time series based on recent observed values, as a new observation is obtained 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. 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 from a generator classes. To classify an unknown time series, PSC runs each PSP module on it and chooses the most predictive one as the predicted class label.

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, such as the various ARIMA models (e.g. [3]). 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. [4], [5]) 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. [6], [7]. 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.

II. Probable Series Classifier Algorithm

A complete description of the PSC algorithm can be found in our previous work [2]. An abbreviated description is provided here for easy reference.

Consider a time series of values $\vec{x}_0, \vec{x}_1, \ldots, \vec{x}_t$ generated by k distinct Markov generators. 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. 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 this equation for classification by finding the generator class that maximizes the probability of an unknown time series (using PSP for some key 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.

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

III. 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. Assume we have previous time series values generated by this generator. Using these examples and recent time series values, we can estimate the time series value at time j. 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) The time series value(s) used in generating a predicted value, i.e. those 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) $\vec{x}_{j-1}, \ldots, \vec{x}_{j-m}$.

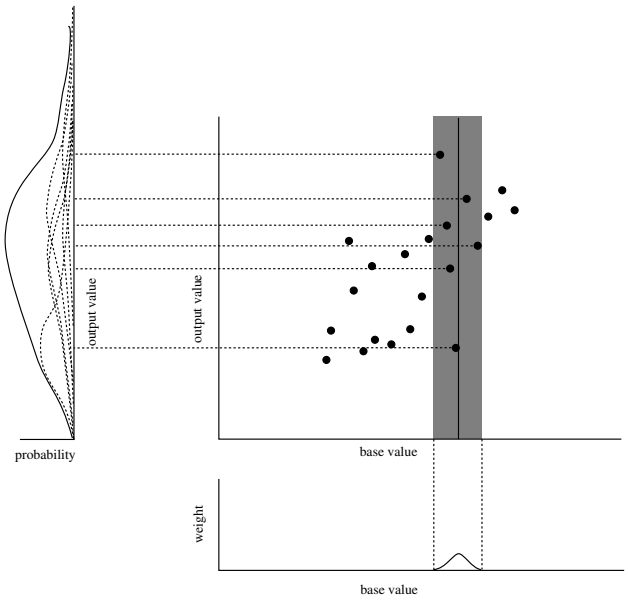

Fig. 1. Data prediction. The dots in the main graph show the data available for use in prediction. The grey bar shows the range of values used in the prediction. The bottom graph shows the weight assigned to each model point. The left graph shows the contribution of each point to the predicted probability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

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 approximates a Gaussian but is much cheaper to compute and reaches zero in a finite bandwidth. The finite bandwidth allows most 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 $n * h \to \infty$. 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 $1/\sqrt{n}$ satisfying 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.

We additionally constrain the approximation to use no more than $\sqrt{n}$ points by only using the $\sqrt{n}$ nearest points to the query base value(s). In the case of points that are the same distance from the query base value(s), we use newer model points in preference to older model points. Constraining the number of points used is very important for the speed of the algorithm because the bandwidth selection takes time $O(m^2)$ where m is the number of points used. By enforcing $m = \sqrt{n}$, the total time for bandwidth selection becomes $O(\sqrt{n}^2) = O(n)$. This new constraint drastically improves the speed of the algorithm over our previous work ($\approx 25x$).

Figure 1 illustrates the PSP algorithm. The dark circles represent model points. The x/y axes shows the base/output values, respectively. 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 I.

The predicted output values must be smoothed to get a continuous probability density. We will 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, some regions would have zero probability. These zero probability regions would make entire sequences have zero probability for

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* $\leftarrow \{z.\textit{output\_value} \mid z \in OP \wedge$
    $\text{dist}(z.\textit{base\_values}, \textit{base\_values}) < h_b\}$
  Perform correlation correction on *pred*.
  let *base* $\leftarrow \{z.\textit{base\_values} \mid z \in OP \wedge$
    $\text{dist}(z.\textit{base\_values}, \textit{base\_values}) < h_b\}$
  Choose h_o that minimizes $M(h_o)$ over *pred*.
  Return probability density equal to
    $\text{pdf}(z) = \sum_i K_g(\textit{pred}_i - z, h_o) *$
    $K_t(\textit{base}_i - \textit{base\_values}, h_b)$

TABLE I
PROBABLE SERIES PREDICTOR ALGORITHM.

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 choose a metric for determining the quality of a bandwidth. The error we are interested in is the ratio of predicted to actual probabilities. We chose to use the pseudo-likelihood cross validation measure [8], [9]. This method is known to minimize the Kullback-Leibler distance(for many classes of probability densities [10]), a close proxy for the desired error. The pseudo-likelihood cross validation method maximizes the likelihood of the data predicting itself over all possible bandwidths under leave-one-out cross-validation. The pseudo-likelihood cross validation measure is defined as:

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

PSP searches over all possible bandwidths from one corresponding roughly to expected measurement noise to one corresponding to the range of possible values. The pseudo-likelihood measure seems to be unimodal (at least over our data). We have used this to speed up the search and have noticed no degradation in performance over an exhaustive search. The search is done by starting with a range over all possible bandwidths. Five bandwidths are selected from this range (one at each end plus three in the middle) such that they have a constant *ratio*. The maximum pseudo-likelihood measure amongst these points is found. A new range is chosen equal to the range between the sample point smaller than the maximum and the sample point larger than the maximum. The process is repeated until the ratio between the top of the range and the bottom is less than a constant γ (we used $\gamma = 1.2$). The bandwidth with the maximum pseudo-likelihood is then chosen as the bandwidth. This new recursive bandwidth selection scheme makes the algorithm run in 60% of the time taken by our previous exhaustive search.

As exemplified in Figure 1, there is usually a strong correlation between the time series values at time t and $t-1$. This correlation causes a natural bias in predictions. Model points with base values below/above the query base value tend to predict an output value which is too low/high, respectively. We can remove 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 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.

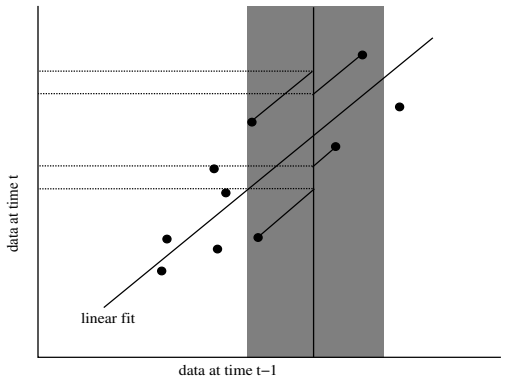

Fig. 2. Correlation removal. A linear fit to the model points (shown as dots) is shown. The grey vertical bar shows the range of values actually used in the prediction. The short solid lines show the effect of shifting the model points to match the base value of the query while taking into account the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

IV. Evaluation

We evaluated the Probable Series Classifier (PSC) using data logged by our robot as it performed various tasks. The data was hand classified as a baseline for comparison with the automatic classification. We used a standard Sony AIBO ERS-210 for gathering all of our data.

A. Methodology

We generated a set of data series from the sensors on our robot. We used two different sensors, a CMOS camera and an accelerometer. Since PSC currently only supports single dimensional data, we reduced each data series down to a single dimensional data series. Each camera image was reduced to an average luminance value (a measure of brightness), resulting in a 25Hz luminance signal. The accelerometer data is inherently three dimensional with accelerations along three axes. We chose to use the axis oriented towards the front of the robot (the other axes gave similar results). The accelerometer data has a frequency of 125Hz. For each task, PSC was trained on a segment of data for each possible class. PSC used a window of data to generate each classification starting at the data item to be classified and extending backwards in time, i.e. only data that would be available in an on-line scenario was used for classification. The PSC generated label was compared to a hand generated label to ascertain accuracy. In some of the signals, there were segments of the test signal that did not correspond to any of the trained classes. These segments were not used in calculating accuracy. We considered four different classification tasks, two using each sensor stream.

B. Results

Each figure show the results from one task. The bottom part of each figure shows the raw data signal used for testing. Each of the other figures corresponds to one of the trained classes. The class to which it corresponds is labelled to the left of each graph. The thick black line running through parts of each class graph indicates when this class is the correct class according to the human generated labelling. The small black dots show the probability that

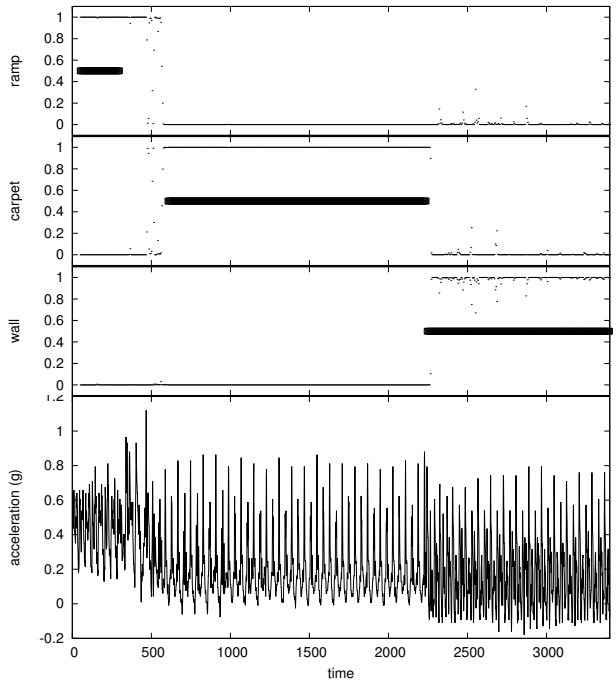

Fig. 3. Use of accelerometer data to distinguish between walking down a ramp, walking across a carpet, and walking into a wall.

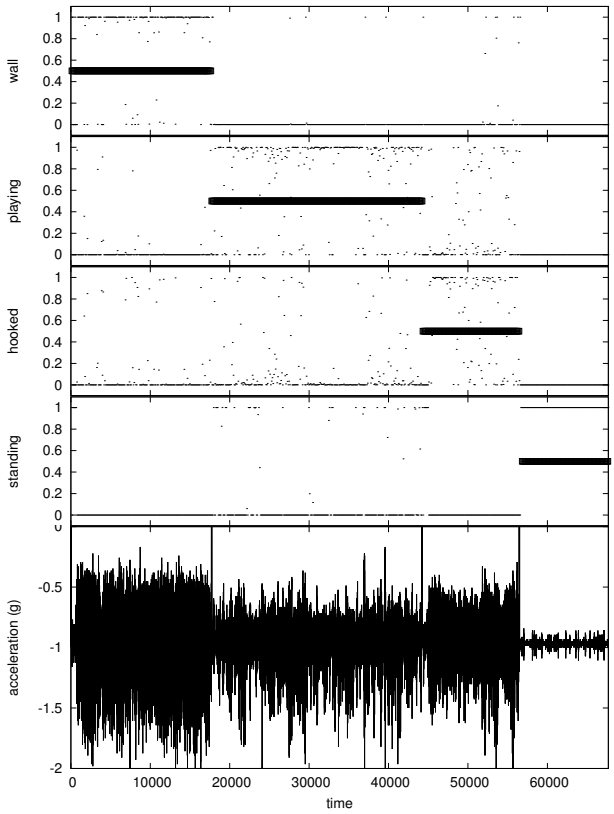

Fig. 4. Use of accelerometer data to distinguish between playing soccer, walking into a wall, walking with one leg caught on an obstacles, and standing still.

PSC assigned to this class at each point in time (based on a window of data prior to this time point). Ideally, the probability would be 1.0 when the thick black bar is present and 0.0 otherwise. In sections where the test data series does not correspond to any of the trained classes, the indicator bar is absent and the output of PSC for each class is irrelevant. Table II summarizes the results achieved by PSC.

TABLE II
ACCURACY OF PSC IN VARIOUS TEST CLASSIFICATION TASKS.

Task	Sensor	Accuracy
Walk stability	Accelerometer	99.19%
Walk interference	Accelerometer	78.49%
Lights playing	Camera	64.69%
Lights standing	Camera	93.77%

Figure 3 shows the results from the first accelerometer task distinguishing between walking down a metal ramp, across a soft carpet, and into a low wooden wall. This task is labelled as “walk stability” in the results summary table. PSC was trained on one example sequence and tested on a completely separate sequence. Each class was trained using 500 examples, except 400 examples were used for training the “ramp” class. PSC was tested on windows of size 50 (.4 seconds of data). PSC was tested after every 5 data points, i.e. the window was moved 5 data points at a time. As the graphs show, PSC does an excellent job of distinguishing between these different walking conditions achieving and accuracy of 99.19%.

Figure 4 shows the results from the second accelerometer task distinguishing between playing soccer, walking into a wall, walking with one leg hooked on an obstacle, and standing still. The playing class includes a wide variety of signals including walks in several different directions and full body kicking motions such as diving on the ball. This task is labelled as “walk interference” in the results summary table. PSC was trained on example sequences from the test sequence. In other tests, we did not observe a noticeable difference between testing on training data and testing on separate testing data. Each class was trained using 5000 examples. PSC was tested on windows of size 125 (1 second of data), and was tested after every 100 data points. PSC performed well overall, correctly classifying 78.49% of the test windows. PSC performed perfectly on the standing still data. It had the most problems identifying hooked on an obstacle (59.84% accurate), often confusing it with playing (69% of errors for this class).

Figure 5 shows the results from the first camera task distinguishing between bright, medium, dim, and off lights while the robot is playing soccer. This task is labelled as “lights playing” in the results summary table. PSC was trained on one example sequence and tested on a completely separate sequence. Each class was trained using 1000 examples. PSC was tested on windows of size 50 (2 seconds of data), and was tested after every 25 data points. PSC performed fair overall, correctly classifying 64.69% of the test windows. Most of the errors were due to problems distinguishing between bright lights and medium lights. Confusion errors between these two classes accounted for 54% of the errors the algorithm made during this test.

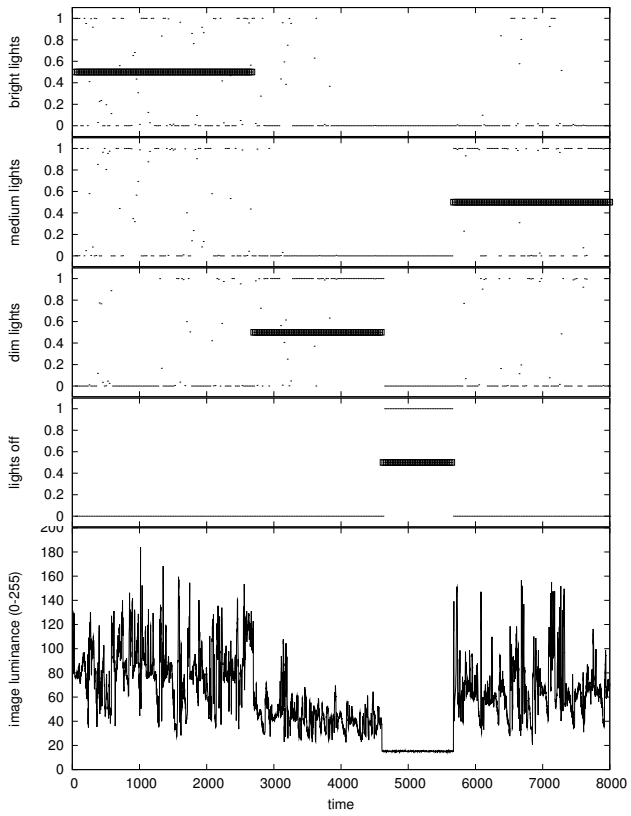

Fig. 5. Use of average luminance from images to distinguish between bright, medium, dim, and off lights while playing soccer.

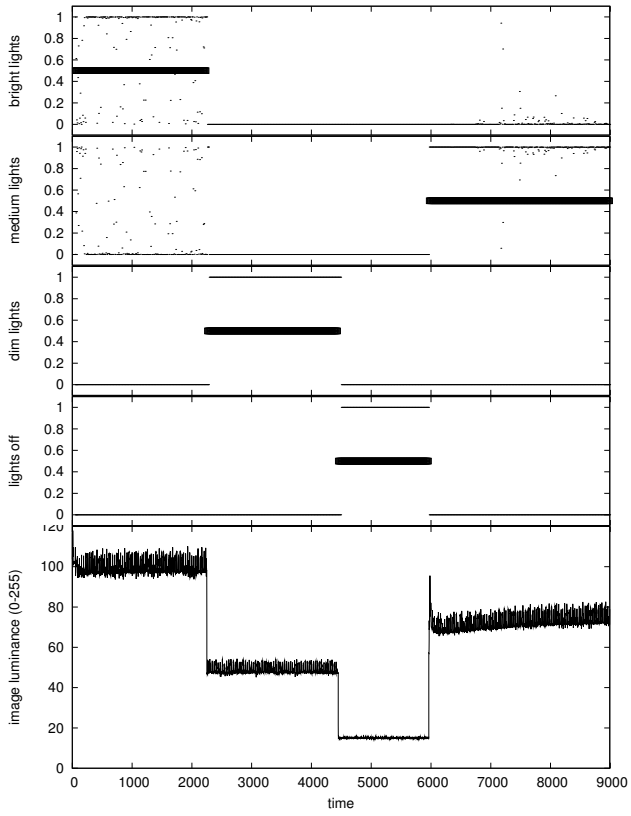

Fig. 6. Use of average luminance from images to distinguish between bright, medium, dim, and off lights while standing still.

Figure 6 shows the results from the second camera task distinguishing between bright, medium, dim, and off lights while the robot is standing still. The robot moved its head to look at different objects. This task is labelled as "lights standing" in the results summary table. PSC was trained on one example sequence and tested on a completely separate sequence. Each class was trained using 150–200 examples. PSC was tested on windows of size 50 (2 seconds of data), and was tested after every 10 data points. As the graphs show, PSC excels at distinguishing between these different lighting conditions, achieving an accuracy of 93.77%.

V. Conclusion

We have presented an algorithm for generating predictions of future values of time series and shown how to use that algorithm as the basis for a classification algorithm for time series. The algorithm is general in that it is able to detect a wide range of changes to a signal. 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. We proved through testing on robotic sensor data that the resulting classification algorithm can be used to differentiate between semantically different signal classes.

Acknowledgment

This research was sponsored by the United States Army under Grant No. DABT63-99-1-0013. The content of the information in this publication does not necessarily reflect the position or the policy of the Defense Advanced Research Projects Agency (DARPA), the US Army or the US Government, and no official endorsement should be inferred.

References

[1] S. Lenser and M. Veloso, "Automatic detection and response to environmental change," in *Proceedings of ICRA-2003*, 2003.

[2] ——, "Time series classification using non-parametric statistics," in *Under submission*.

[3] K. Deng, A. Moore, and M. Nechyba, "Learning to recognize time series: Combining arma models with memory-based learning," in *IEEE Int. Symp. on Computational Intelligence in Robotics and Automation*, vol. 1, 1997, pp. 246–250.

[4] M. Basseville and I. Nikiforov, *Detection of Abrupt Change - Theory and Application*. Englewood Cliffs, N.J.: Prentice–Hall, 1993. [Online]. Available: http://www.irisa.fr/sigma2/kniga/

[5] M. Hashimoto, H. Kawashima, T. Nakagami, and F. 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)*, 2001, pp. 1321–1326.

[6] W. Penny and S. Roberts, "Dynamic models for nonstationary signal segmentation," *Computers and Biomedical Research*, vol. 32, no. 6, pp. 483–502, 1999.

[7] Z. Ghahramani and G. E. Hinton, "Switching state-space models," 6 King's College Road, Toronto M5S 3H5, Canada, Tech. Rep., 1998. [Online]. Available: citeseer.nj.nec.com/ghahramani96switching.html

[8] 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.

[9] R. P. W. Duin, "On the choice of smoothing parameters of Parzen estimators of probability density functions," in *Proceedings of IEEE Transactions on Computers*, vol. 25, 1976, pp. 1175–1179.

[10] P. Hall, "On Kullback-Leibler loss and density estimation," in *The Annals of Statistics*, vol. 15, 1987, pp. 1491–1519.