A Nearest Trajectory Strategy for Time Series Prediction*

James McNames

Information Systems Laboratory, Stanford University
Stanford, CA 94305-9510
E-mail: *mcnames@simoon.stanford.edu*

Abstract-A method of local modeling for predicting time series generated by nonlinear dynamic systems is proposed that incorporates a weighted Euclidean metric and a novel ρ-steps ahead cross-validation error to assess model accuracy. The tradeoff between the cost of computation and model accuracy is discussed in the context of optimizing model parameters. A fast nearest neighbor algorithm and a novel modification to find neighboring trajectory segments are described.

I. Introduction

This paper proposes a nonparametric forecasting method for univariate time series that contain little or no noise. For practical purposes it is assumed that the time series is generated by a nonlinear dynamic system governed by the following equations,

$$\begin{aligned} z_{t+1} &= h(z_t, u_t) \\ y_t &= g(z_t) \end{aligned} \tag{1}$$

where $h(z_t, u_t)$ and $g(z_t)$ are nonlinear time-invariant functions with continuous derivatives, $u_t \in \mathbb{R}^{d_s}$ is a deterministic function of t, $y_t \in \mathbb{R}^1$ and $z_t \in \mathbb{R}^{d_s}$. It is assumed that $h(z_t, u_t)$, $g(z_t)$, u_t, and d_s are unknown and the prediction must be based solely on a time series of points, $[y_1, y_2, \ldots, y_n]$, generated by the system.

II. Discussion on Local Modeling

Local models generate predictions by finding local portions of the time series that closely resemble a portion of the points immediately preceding the point to be predicted. The prediction is an estimate of the average change that occurred immediately after these similar portions of points.

Previous studies have shown that forecasting methods based on local models produce predictions that are better than or comparable to competing models and they have a number of favorable properties not shared by other methods [5, 8, 9, 25, 34].

To use local models for time series prediction there are many decisions that one must make. For example, how should *local* be defined mathematically, how should the input vector be constructed, what should the model be designed to predict, what type of local model should be used, and how should the model accuracy be evaluated.

Individually, researchers have offered suggestions on how to make each of these decisions. However, a comprehensive method that combines the successful innovations of various researchers has not emerged and the tradeoff between model accuracy and the computational costs has often been overlooked. This paper attempts to fill these gaps and offers some new suggestions on how to use local models for time series prediction.

A. Computational Limitations

Virtually all nonlinear models (local and global) have parameters that must be chosen before the model can be constructed or tested. When working with black box systems nothing is known about the process that could help the analyst choose values for these parameters. Ideally, the parameter values would be chosen so as to maximize some measure of model accuracy. Although computation is cheaper and more widely available than ever before, a global maximization over all of the model parameters is almost never viable. The natural alternative, nonlinear optimization, is usually not feasible because of numerous local minima[1].

In some instances efficient methods may exist that find good values for some of the parameters. Existing theory may also provide good estimates or, more often, bounds on parameter values. Unfortunately, these options are virtually never applicable to all of the model parameters.

In most cases the accuracy of the model will be much less sensitive to some parameters than others. An experienced analyst will often be able to choose good val-

*Published in *Proceedings of the International Workshop on Advanced Black-Box Techniques for Nonlinear Modeling*, Katholieke Universiteit Leuven, Belgium, 112–128, July, 1998

†Permanent email: mcnames@alumni.stanford.org

[1]The model accuracy is often convex in some of the parameters and, if the remaining parameters are fixed, the model accuracy can be efficiently maximized. However, the problem of numerous local minima usually still exists for the remaining parameters.

ues for these parameters. Unfortunately, the process by which an analyst chooses the values for parameters is often subjective and does not admit of a general purpose recipe. Consequently the accuracy of a model may greatly depend upon the skill and experience of the analyst[2].

Often a coarse optimization is performed by measuring the model accuracy for several values of the model parameters. The extent of this optimization is limited by computational resources and, as a result, model accuracy is sacrificed. The remainder of this discussion suggests how parameters of local models may be chosen in the context of this tradeoff.

B. Embedding: Reconstructing the System State

Takens has shown that the state of many dynamic systems can be accurately reconstructed by a finite window of the time series [39]. This window is called a time delay embedding,

$$x_t \triangleq [y_t, y_{t-\tau}, \ldots, y_{t-(m-1)\tau}] \tag{2}$$

where m is the embedding dimension and τ is the embedding delay. Takens' work was later generalized and shown to apply to a broader class of systems by Sauer *et al.* [35].

Time delay embeddings are widely used as the input vector to dynamic models, both linear and nonlinear. Takens' theorem provides a sound theoretical basis for this approach and has been applied successfully in many applications.

The theorem assumes that the time series is infinitely long and noise-free; in practice, these conditions are never met and the selection of τ and m may critically affect how accurately the embedding reconstructs the state of the system. Many researchers have recognized this problem and proposed methods to find τ and m for *finite* time series with and without noise [2, 3, 4, 21]. The goal of these methods is usually to find the values of these parameters that minimize the embedding dimension m without sacrificing the accuracy of the reconstruction. Although a compact reconstruction is efficient computationally, it does not necessarily maximize accuracy.

Although Takens' theorem only applies to infinitely long time series there is evidence that the accuracy of the reconstruction is not sensitive to the value of m for *finite* noise-free time series so long as m is sufficiently large. Kugiumtzis makes a strong case that the most important consideration in choosing the embedding parameters, m and τ, is the window length [21].

$$w = m\tau \tag{3}$$

As long as w is sufficiently large and m is not too small, a wide range of values of τ and m will create an accurate reconstruction of the system state.

C. Choosing a Metric

The local models discussed in this paper are constructed using only the nearest points on the k nearest neighboring trajectory segments.

Choosing an appropriate measure of nearness, or metric[3], is an important decision that is usually overlooked. The most common metric is the square of the Euclidean distance[4] between an input vector x_t and the query vector q.

$$d_E(q_t, x_t) = (q_t - x_t)^\mathsf{T}(q_t - x_t) \tag{4}$$

In this case the only parameter is m, the embedding dimension. Choosing a more general measure with more parameters can drastically effect model accuracy [13, 29, 40]. For example, a weighted Euclidean distance,

$$d_\Lambda(q_t, x_t) = (q_t - x_t)^\mathsf{T}\Lambda(q_t - x_t) \tag{5}$$

could be used where $\Lambda \in \mathbb{R}^{m\times m}$ is any positive semidefinite matrix. However, short time series are often too short to estimate m^2 parameters and optimizing over so many parameters is computationally impractical for longer time series.

Murray proposed using a exponentially weighted diagonal metric[5]

$$\Lambda(i,j) \triangleq \begin{cases} \lambda^{i-1} & i = j \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where $0 < \lambda < 1$ [29].

This metric is intuitively appealing because the components of x_t closest in time to the prediction are given exponentially more weight. It is especially appropriate for chaotic systems where neighboring states are known to diverge exponentially with time. Since the standard Euclidean metric, d_E, is a subset of d_λ an appropriate choice of λ will not decrease model accuracy and is likely increase it.

The metric d_λ is defined by the parameters m and λ. If m is sufficiently large and λ is sufficiently small the influence of the most distant components of x_t will be negligible. Thus, in principle, if a model is constructed with m sufficiently large, only λ will affect model accuracy.

[2] See Friedman for a more thorough discussion on *expert bias* [10].

[3] In this paper *metric* means a measure of nearness between two points. It may not have the usual metric properties.

[4] The square of the Euclidean distance is used to reduce computation. The same nearest neighbors would be found with the Euclidean distance.

[5] This metric is of a different form than that proposed by Murray but it is mathematically equivalent.

Since the model accuracy is sensitive to λ it is generally worth the computational cost to find the best value. This is a critical component of the method proposed here. For many nonlinear models, such as neural networks, the nonlinear optimization over the adaptive parameters (weights) typically consumes too much computation to enable a global optimization over structural parameters, such as the number of neurons in a hidden layer. In contrast, the model described in this paper can be constructed and evaluated efficiently enough to enable a global optimization over several of the model parameters which makes the choice of the parameters to be optimized especially crucial.

C.1. Other Metrics: Previous Work

Although the choice of the metric can drastically affect the accuracy of local models, very little has been published on this topic in the field of time series prediction. The only publications known to the author are briefly summarized in this section.

Farmer and Sidorowich suggested a fixed $\lambda = e^{-h}$, where h is the metric entropy, which they report is in some sense linearly optimal [8].

In addition to the metric used here Murray also investigated a two-parameter metric that used a tridiagonal weighting matrix [29].

Kugiumtzis compared L_2, L_1 and L_∞ norms, mostly for the purpose of estimating correlation dimension [22]. Kugiumtzis also performed a brief analysis using local linear models for prediction and reported that there were not significant differences in the prediction errors between each of the three norms.

Casdagli and Weigend reported that model accuracy is very sensitive to the choice of the embedding dimension when the Euclidean metric is used [6].

Tanaka *et al.* developed an elegant approximation to an optimal metric with a rank-1 weighting matrix [40]. This metric has no user-specified parameters, is computationally inexpensive, and reportedly is more accurate than models that use the Euclidean metric or Murray's metric. The same metric was also developed by Garcia *et al.* [13]. Unfortunately there is no obvious means of using this metric with existing nearest neighbor algorithms. If this metric requires a brute force nearest neighbor algorithm, it is computationally impractical for moderate to large-sized data sets.

C.2. Choosing the Embedding Parameters

A good value for the window length, w, can often be chosen by visual inspection of the time series. Typically, noise-free univariate time series generated by the dynamic system in Equation (1) will contain periodic oscillations of roughly the same period. w should be chosen to span several of these oscillations. Longer windows usually require more computation with little increase in accuracy. Longer windows may also significantly reduce the number of points,

$$n_c = n - m\tau \tag{7}$$

that may be used to construct the model for short time series. Conversely, choosing a window that is too short will reduce model accuracy.

Once w has been selected, the embedding delay, τ, must be chosen. One justification for choosing the smallest possible value for τ is that it increases the accuracy of d_λ. For example, if the time series was sampled at a rate T from a continuous time system, d_λ approximates the weighted integrated squared error, ISE_λ.

$$\mathrm{ISE}_\lambda(q_t, x_t) \triangleq \int_{tT-(m-1)\tau T}^{tT} \lambda^{v-tT} (q_v - x_v)^2 \, dv \tag{8}$$

$$d_\lambda(q_t, x_t) = \sum_{i=0}^{m-1} \lambda^i (q_{t-i\tau} - x_{t-i\tau})^2 \tag{9}$$

$$\approx \frac{1}{T} \mathrm{ISE}_\lambda(q_t, x_t) \tag{10}$$

For a fixed w, the choice of τ governs the accuracy of the estimated ISE_λ. Small values of τ will increase the accuracy of the estimated ISE_λ, though not necessarily the model accuracy. However, small values of τ will require a larger value of m and thereby increase the computational cost. In light of this tradeoff, τ should be chosen as small as possible within the limits of the computational budget.

D. Iterated versus Direct Prediction

Suppose we are given a time series, $[y_1, y_2, \ldots, y_n]$, and asked to predict p steps ahead. *Direct prediction* is the method by which a model is built to directly predict y_{n+p} from the input vector x_n,

Iterated prediction is the method by which a model is built to predict one step ahead p times. The model estimates $\hat{y}_{n+1}$ from x_n which is used in turn to estimate the first component of the input vector x_{n+1}.

$$\hat{x}_{n+1} = \left[\hat{y}_{n+1}, y_{n+1-\tau}, \ldots, y_{n+1-(m-1)\tau}\right] \tag{11}$$

This process is iterated for p steps finally producing the prediction $\hat{y}_{n+p}$.

There has been much debate about which method is superior [4, 5, 8, 9, 34]. There is strong empirical evidence that iterated prediction performs better on short term forecasts for a variety of nonlinear models [4, 5, 8]. Sauer has suggested averaging the direct and iterated predictions to reduce variance [34].

Direct prediction is questionable because for nonlinear dynamic systems a function that maps p steps into the future will usually be more complicated than one that predicts a single step into the future [9].

Iterated prediction has the disadvantage that it does not take into account the accumulated errors in the input vector. After a model has been iterated further than it is capable of accurately predicting, the input vector will no longer be an accurate reconstruction of the state of the system being modeled. In this case the best least squares prediction is simply the mean of the signal, $\bar{y}$.

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{12}$$

Iterated prediction, however, will produce the same prediction regardless of the accuracy of the input vector. As a result, predictions many steps ahead will have the same variance as predictions a few steps ahead and will consequently have a significantly larger squared error than simply predicting $\bar{y}$ or using direct prediction[6].

For global models direct prediction is more expensive computationally because p models,

$$\begin{aligned} \hat{y}_{n+1} &= f_1(x_n) \\ \hat{y}_{n+2} &= f_2(x_n) \\ &\vdots \\ \hat{y}_{n+p} &= f_p(x_n) \end{aligned}$$

must be constructed to predict $1, 2, \ldots, p$ steps ahead. Iterated prediction requires that only one model be constructed. However, for local models direct prediction models can use the same k nearest trajectories as the basis for all p predictions [8]. Since finding the nearest trajectories is the most computationally expensive operation for local models, direct prediction is much less expensive than iterated prediction, which requires that the nearest trajectories be found for p different input vectors.

Despite its computational cost, iterated prediction is almost always used because of its superior short-term accuracy. However, one should use this method with caution since medium to long-term forecasts can be worse than predicting $\bar{y}$.

E. A Nearest Trajectory Algorithm

One of the differences between a regression problem and a time series prediction problem is that it is possible to increase the number of data points by up-sampling, or interpolating, the time series; more data can be generated without adding any new information. This is problematic because a signal that is sampled at a high enough rate will have all k of its nearest neighbors adjacent to each other in the time series.

[6]To the best of the author's knowledge, this disadvantage of iterated prediction has not been recognized in previous discussions on this topic.

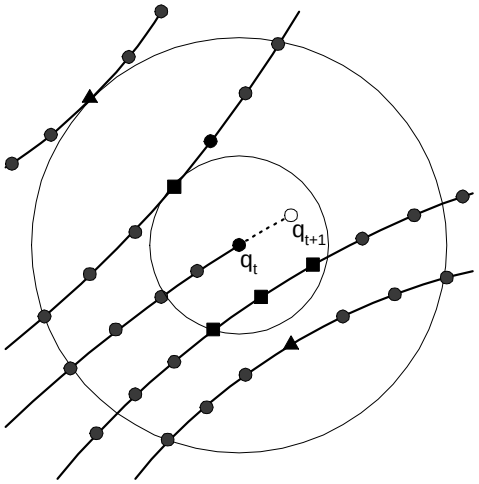

Figure 1: Four trajectory segments and a predicted trajectory in a two-dimensional embedding space. The four nearest neighbors of q_t are shown by squares. The inner circle shows the distance to the fourth nearest neighbor and the outer circle shows the distance to the fourth nearest trajectory.

For example, in Figure 1 many of the points on the nearest trajectory segment are closer than the points on the fourth nearest trajectory segment.

This problem can be solved by finding the nearest *trajectory segments* instead of the nearest neighbors [8]. Fortunately, it is possible to modify existing nearest neighbor algorithms to find the nearest trajectory segments. This is advantageous because much research has gone into the development of efficient nearest neighbor algorithms (Section F.).

A novel nearest trajectory modification is as follows: suppose a nearest neighbor algorithm has found a point x_i that is closer than the k nearest points found by the algorithm so far.

1. Calculate the distance to the points preceding x_i, $\{x_{i-1}, x_{i-2}, \ldots\}$, until the nearest local minimum is found. Repeat this procedure for the points succeeding x_i. The local minimum that is found by this procedure, x_{min}, is the closest point in the trajectory segment.

2. Calculate the distance to the points preceding x_{min} until either a maximum is found or the distance becomes greater than the distance to the kth nearest neighbor found so far. Call this point x_{max}. Eliminate all of the points between x_{min} and x_{max} from consideration by the nearest neighbor algorithm.

3. Repeat the previous step for the points succeeding x_{min}.

4. Replace the kth nearest neighbor found so far with x_{min} and continue with the nearest neighbor algorithm.

F. Nearest Neighbor Algorithms

Many nearest neighbor algorithms have been proposed to overcome the high computational cost of the naïve brute force approach[7]. Almost all of these algorithms can be divided into two categories: axis-partitioning algorithms and triangle inequality-based algorithms. Axis-partitioning algorithms divide the m-dimensional input space into hypercubes and establish a lower bound on the distance from the query point, q, to all points contained in each hypercube. If the lower bound is greater than the distance to the kth nearest neighbor found so far, all of the points contained by the hypercube can be eliminated without explicitly calculating the distance to each point [11, 20, 30, 32, 38].

Triangle inequality-based algorithms use the triangle inequality to find lower bounds on the distance to points. If the lower bounds is greater than the distance to the kth nearest neighbor found so far, the point can be eliminated from consideration without explicitly calculating the distance to that point. Subsets of points can also be eliminated by finding a lower bound on the distance to an enclosing hypersphere [1, 12, 17, 18, 27, 42].

In low-dimensional spaces both types of nearest neighbor algorithms have been shown to perform drastically better than the brute force approach. In low dimensions most of these algorithms achieve $O(\log n_c)$ search time and require only $O(n_c \log n_c)$ preprocessing time and storage, where n_c is the number of points in the data set.

In high-dimensional spaces, say $m > 15$, the bounding techniques of both types of algorithms are ineffectual and the performance is much worse. In fact, the computational cost of these algorithms can be significantly higher than the brute force approach due to overhead imposed for ordering the search of the input space and for calculating the lower bounds. Generally, the search time increases exponentially with m up to a limit where the distance is calculated for nearly all of the points in the data set.

If all of the points in the data set lie in a low-dimensional subspace of $\mathbb{R}^m$, the triangle inequality algorithms have an advantage over axis-partitioning algorithms—they can achieve search times comparable to low-dimensional data sets. If the dimension of the subspace is held constant, the search time grows roughly linearly with m. This is especially relevant when the data points are taken from a time series generated by a nonlinear dynamic system because the points will often lie on a low-dimensional manifold [14, 35, 39]. This feature of triangle inequality-based algorithms has not been widely recognized in either the nearest neighbor algorithm literature or in the time series prediction literature, with the exception of Micó *et al.* [28].

F.1. Another Nearest Neighbor Algorithm

In this section a new nearest neighbor algorithm, ANNA[8], is described that is competitive with the most efficient nearest neighbor algorithms in low-dimensional spaces and nearly as efficient as the brute force approach in high-dimensional spaces. Since this algorithm is based on the triangle-inequality, it is much faster than the brute force approach when used on data sets generated low-dimensional nonlinear dynamic systems.

As with other triangle inequality algorithms, ANNA requires that a metric be chosen before construction. It is assumed in this section that the metric is the square root of the metric given in Equation (5), d_Λ. The square root is necessary for the triangle inequality to hold.

$$d(q, x_i) \triangleq d_\Lambda(q, x_i)^{1/2} \tag{13}$$

ANNA consists of two stages, a construction phase and a search phase. During the construction phase a search tree is recursively constructed. Initially all of the points in the data set are subdivided into c distinct groups. Each subset is further subdivided into c subsets and so on until the size of a subset becomes less than or equal to n_T.

Any clustering algorithm could be used to divide each subset into further subsets. The following algorithm is used in ANNA: suppose a set of n_p points are to be divided into c distinct subsets.

1. Pick a point x_0 at random.
2. Find the point in the data set furthest from x_0. This point will be the first cluster center.
3. Find the point in the data set that has the maximum distance to the nearest of the cluster centers found thus far. This point will be the next cluster center.
4. Repeat the previous step until c cluster centers have been found.
5. Assign each point to the nearest cluster center. Calculate the center of mass for each cluster of points.

[7] See Nene and Nayar for a recent concise review of existing algorithms [30].

[8] ANNA is an improved version of the triangle inequality-based algorithm that was proposed by Fukunaga and Narendra [12].

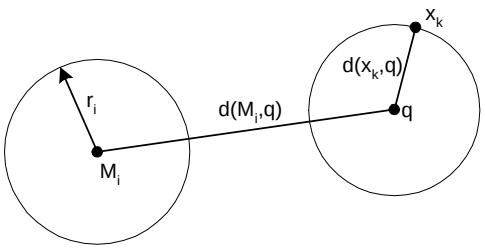

Figure 2: This figure illustrates Rule 1. The circle around M_i encloses all of the points, S_i, assigned to the node i. The circle around the query point, q, encloses the k nearest neighbors found so far by the algorithm. Rule 1 effectively states that if the circles do not intersect, none of the points contained in S_i can be one of the k nearest neighbors of q.

6. Find the c points, $\{M_1, M_2, \ldots, M_c\}$, that are closest to the center of mass in each cluster of points.

7. Assign each point in the data set to the nearest center point, M_i. The set of points assigned to a center point M_i will be denoted by S_i

8. For each set of points, S_i, calculate the distance, r_i, to the point that is furthest from the center point, M_i.

$$r_i = \max_{x_i \in S_i} d(x_i, M_i) \tag{14}$$

9. For all sets, S_i, that contain less than n_T points, calculate the distance from the center point, M_i, to each point in the set, $d(M_i, x_i)\ \forall x_i \in S_i$.

Despite the large number of steps in this clustering algorithm the computation is $O(n_p)$.

During the search phase a depth-first branch and bound search algorithm is used. First, the algorithm calculates distance from the query point to each center point, $d(M_i, q)$, in the top level of the search tree. Second, the successor nodes, which represent smaller subsets of points, are searched in order of increasing $d(M_i, q)$. This process is repeated until a terminal node is reached.

After k distances have been calculated the following rule may enable the algorithm to avoid searching a node.

Rule 1: No point in S_i can be the kth nearest neighbor of q if

$$d(x_k, q) < d(M_i, q) - r_i$$

where x_k is the kth nearest neighbor found so far. This rule is illustrated in Figure 2.

When a terminal node is reached, Rule 2 can be used eliminate some of the points without explicitly calculating the distance from each point, $x_i \in S_i$, to the query point, q.

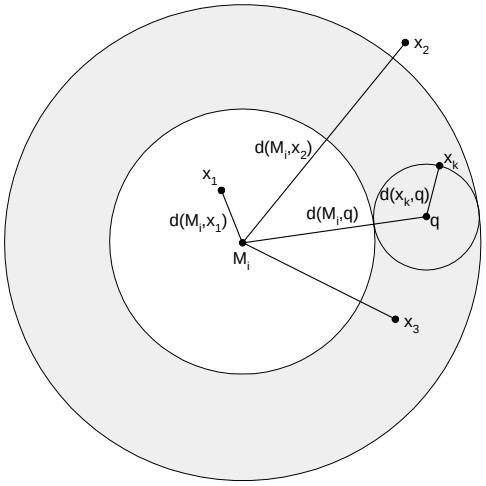

Figure 3: This figure illustrates Rule 2, which effectively states that any point outside the gray region cannot be a nearest neighbor of q. This rule can be used to eliminate points x_1 and x_2, but not x_3 though it is not one of the k nearest neighbors of q.

Rule 2: No point x_j in the terminal node i can be the kth nearest neighbor of q if[9]

$$d(x_k, q) < |d(M_i, q) - d(M_i, x_j)|$$

Several additional rules have been proposed to eliminate more points than the rules described here [18]. However, Larsen and Kanal have shown these additional rules are rarely invoked, especially for high-dimensional problems [23]. Other improvements have been suggested by various researchers but the additional overhead imposed by more extensive rule checking often outweighs the savings [16, 26, 32].

F.2. Nearest Neighbors in Time Series

The search time of ANNA can be reduced when the query points are taken from consecutive points in the time series. For example, if the k nearest neighbors for an input vector x_t have been found,

$$[x_{v(1)}, x_{v(2)}, \ldots, x_{v(k)}] \tag{15}$$

where $v(i)$ is the index of the ith nearest neighbor, then the set of points

$$[x_{v(1)+1}, x_{v(2)+1}, \ldots, x_{v(k)+1}] \tag{16}$$

will usually be very close to x_{t+1}. If the distance to these points is calculated before searching the tree then the lower bounds used in Rules 1 and 2 will be much tighter, more points will be eliminated, and the computation will be reduced.

[9] This is a slight extension of the original rule proposed by Fukunaga and Narendra and was originally suggested by Kamgar-Parsi and Kanal [12, 18].

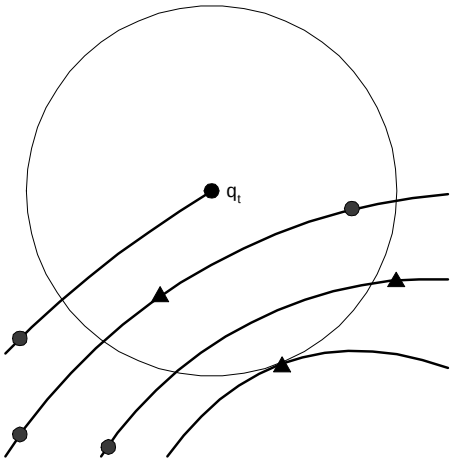

Figure 4: Three coarsely sampled trajectories in a two-dimensional embedding space. The nearest point on each trajectory is shown by a triangle. Note that the nearest point on the second nearest trajectory is further away than the nearest point on the third nearest trajectory.

G. Interpolation

As discussed in Section E, the nearest trajectory algorithm finds the closest point on each of the k neighboring trajectory segments. If the time series is coarsely sampled from a continuous-time dynamic system, the nearest point on a neighboring trajectory can only be found with coarse precision. If a higher sampling rate is used or, equivalently, if a time series generated by a discrete-time dynamic system is interpolated, the nearest point will be found with greater precision and the prediction will generally be more accurate.

In severe cases where the sampling rate is nearly small enough to incur aliasing, the wrong trajectories may be found by the nearest trajectory algorithm, as shown in Figure 4.

In practice, interpolation can significantly increase accuracy if the raw time series is coarsely sampled. However, effectively increasing the number of points in a time series may proportionally increase the computational cost of the nearest trajectory algorithm. A good rule of thumb is to interpolate, if necessary, so that the effective sampling rate is 5–10 times faster than the Nyquest rate.

H. Local Function Approximation

Once the k nearest trajectories have been found a local model must be constructed to make the prediction. This is a standard function approximation problem with the important exception that the dimension of the input vector, m, is often large compared to the number of data points, k, that are used to construct the model. Most nonlinear modeling methods are not viable under these conditions because they contain too many free parameters and require more than k data points to construct. Models that are computationally expensive to construct are also not viable since a different model must be constructed for every query.

H.1. Local Linear Models

Local linear models have been a popular choice because they can be constructed very quickly using small data sets. Some authors have proposed projecting the input vectors onto a subspace before constructing the linear model. Most often, singular value decomposition is used to find the l-dimensional subspace in which the input vectors exhibit the greatest variance[10]. The linear model is then constructed and the prediction is generated.

One of the commonly mentioned disadvantages of local models is that they are discontinuous [5, 24, 25]. A slight perturbation in the input vector can change the kth nearest trajectory which can, in turn, result in a very different model. This problem can be remedied by giving the furthest trajectories less influence on the local model. For example local linear models can be constructed using weighted linear regression where the furthest trajectories receive the smallest weights [8, 34].

Various weighting functions have been widely researched in the context of kernel regression [36, 43]. It is generally accepted in this field that model accuracy is insensitive to the type of weighting function used and it is reasonable to assume that this is true for the models described here as well. A good choice is the biweight function,

$$w(i) = \left(1 - \frac{d_i^2}{d_k^2}\right)^2 \tag{17}$$

where d_i is the distance to the ith nearest neighbor. One of the advantages of the biweight function is that the kth nearest trajectory is given a weight of zero which makes the local model continuous[11].

H.2. Local Constant Models

Local constant models are a slightly less common choice. Like local linear models, they can be made continuous by using a smooth weighting function. Because constant models have only one degree of freedom they have less variance than linear models but more bias for the same number of points, k. If the optimal k is used, neither is consistently more accurate. Linear models are slightly more expensive com-

[10] This is commonly called principal components analysis.

[11] This is not true in general. If a sufficient number of points are equidistant to q the local model may be discontinuous at that point. However, for $q \in \mathbb{R}^m$ the probability of this happening is virtually zero.

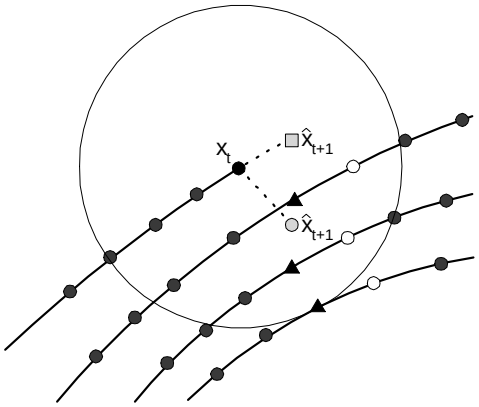

Figure 5: Three trajectory segments and a predicted trajectory in a two-dimensional embedding space. The three nearest trajectory points to the query point, x_t, are shown as triangles. The predicted next point, $\hat{x}_{t+1}$, using averaged prediction is shown as a gray circle. This is just the average vertical value of the white circles. For integrated prediction, shown by the gray square, the predicted next point is the current position plus the average change in the nearest trajectories (the vertical change from each triangle to the corresponding circle).

putationally, but in most cases the additional computation is insignificant compared to the computational cost of finding the k nearest trajectories. However, linear models generally require a larger value of k which is significantly more expensive computationally.

Although local constant models are reasonably accurate at interpolation they are generally poor at extrapolation. Since a local model consists of a weighted average, the output will never be greater than or less than any of the points that make up the average. In high-dimensional spaces this can be a significant disadvantage since nearly every point is an outlier [10]. For example, in Figure 5 the gray circle shows a prediction generated by averaging the next point of the three nearest trajectories.

One way to fix this problem is to predict the change, or residual, $y_{t+1} - y_t$, rather than y_{t+1}. The final prediction can then be found by integrating from the current state.

Figure 5 illustrates why predicting the residual and integrating (*integrated prediction*) may be better than predicting the average of the neighboring trajectory segments (*averaged prediction*)[12].

For large data sets these methods have similar performance. In brief empirical comparisons, integrated prediction was slightly more accurate for short term predictions and averaged prediction was more accurate for longer term predictions. This is probably because averaged prediction is less sensitive to accumulated errors in the input vector.

I. Number of Neighboring Trajectories

Many researchers have reported that model accuracy is sensitive to the number of neighboring trajectories, k, which makes this parameter a good candidate for global optimization [5, 6, 14, 37]. As discussed in Section C, this is a crucial decision because a global optimization can be performed for only a few parameters.

Optimizing k is much more efficient than optimizing the metric parameter λ for two reasons. First, the nearest neighbor search can be performed for different values of k without having to reconstruct the search tree used by ANNA. This is not true for λ which requires that a new search tree be constructed for each value.

Second, the optimal value of k is generally small for noise-free time series, as was shown by Casdagli and Weigend [6]. Only a small number of different values for k need to be considered to find the optimal value. Since $\lambda \in \mathbb{R}^1$, many more values must be considered in order to find the optimal value of λ.

Smith has suggested locally varying the value of k to minimize an estimate of the predicted absolute error [37]. This method has the disadvantage of making local models discontinuous, but certainly warrants further investigation.

J. Model Accuracy

Many methods of evaluating model accuracy have been proposed. A popular choice for local models is leave-one-out cross-validation. This method begins by constructing a model to predict one step ahead with all but one of the n_c points. The error in predicting the omitted point is calculated and the process is repeated for all n_c points. Any of a variety of measures of error could be used. Absolute error, squared error, and the coefficient of correlation are common choices.

The cross-validation error is an estimate of the expected error[13]. For example, if squared error is used the cross-validation error is defined as

$$\mathrm{MSE}_1 \triangleq \frac{1}{n_c}\sum_{i=1}^{n_c}\left(y_{i+1} - f_{i+1}^{-(i+1)}(x_i)\right)^2 \quad (18)$$

$$\approx E_t\left[\left(y_{t+1} - f_{t+1}^{-(t+1)}(x_t)\right)^2\right] \quad (19)$$

$$\approx E_t\left[\left(y_{t+1} - f_{t+1}(x_t)\right)^2\right] \quad (20)$$

where $f_{i+1}^{-(i+1)}(x_i)$ is the model constructed to predict

[12] Integrated prediction has been widely used for time series prediction of nonstationary stochastic processes [7]. However, this typically includes using residuals for the embedding, which is different from the approach taken here.

[13] See Friedman for a discussion of issues surrounding cross-validation and other model selection criteria [10].

one step ahead with the $(i+1)$th point left out of the construction set.

This method is especially attractive for local models because it can be efficiently computed without constructing n_c different models. Instead, during the search for the k nearest neighbors, the omitted point can simply be ignored. A further advantage can be gained by performing one search for the largest value of k that is being considered. One can then construct local models for smaller values of k without performing additional searches which further reduces the computational cost of finding the optimal value of k.

The ability to efficiently compute the cross-validation error is a significant advantage of local models. Many global modeling strategies, such as neural networks, require substantial computation to build a single model; for these strategies building n_c different models is computationally infeasible[14].

A disadvantage of using one-step ahead cross-validation error (OSCV) as a measure of model accuracy is that it does not take into account the model sensitivity to errors in the input vector that occur with iterated prediction; the parameter values that minimize the OSCV error are generally not the same values that minimize the p-steps ahead cross-validation error.

It is better to choose an error measure that represents the cost of making poor predictions in the application in which the model is going to be used. In many cases an average (possibly weighted) model accuracy over p-steps ahead is appropriate,

$$\mathrm{MSE}_{1,p} \triangleq \frac{1}{pn_p} \sum_{i=1}^{n_p} \sum_{j=1}^{p} \left(y_{i+j} - f_{i+j}^{-(i+1,i+p)}(x_i) \right)^2 \tag{21}$$

where $n_p = n_c - p + 1$ and $f_{i+j}^{-(i+1,i+p)}(x_i)$ is the model output j-steps ahead constructed with the points $(i+1)$ to $(i+p)$ left out. Kantz and Jaeger have argued that multi-steps ahead cross-validation (MSCV) also reduces bias [19].

Unfortunately, MSCV is more expensive computationally than OSCV for two reasons. First, p predictions must be made by each model instead of one. Second, a single search for all of the different values of k cannot be done with MSCV because, for example, the prediction two steps ahead depends on the previous prediction, which is different for each value of k. However, the search tree in ANNA still does not need to be reconstructed in order to calculate $\mathrm{MSE}_{1,p}$; the omitted points can simply be ignored during the search.

To reduce the computation required to calculate $\mathrm{MSE}_{1,p}$, fewer than $n_c - p + 1$ terms could be used to approximate the expected error. For example, windows of p-points could be used that are spaced s points apart.

$$\mathrm{MSE}_{1,p} \approx \frac{1}{pn_v} \sum_{i=1}^{n_v} \sum_{j=1}^{p} \left(y_{v_i+j} - f_{v_i+j}^{-(v_i+1,v_i+p)}(x_{v_i}) \right)^2 \tag{22}$$

where $v_1 = 1$, $v_2 = s$, $v_3 = 2s$, and so on. Here n_v is the number of windows, or validation sets.

For large values of p and short time series, removing p values from the data set may significantly affect the model that is constructed. In this case, the cross-validation error will not be representative of the expected error using all n_c points. A novel approach to reduce this effect is to omit only the trajectory segment that surrounds the point being predicted, $y_{v(i)+\rho}$, instead of all p points.

III. Empirical Results

Two data sets are used to illustrate the method proposed here. The first is simulated data from the Lorenz equations. The second data set is the time series used in the competition that was part of this workshop.

A. Measure of Accuracy

The accuracy of models generated for both data sets was measured by the multi-steps ahead cross-validation method described in Section J. The normalized mean squared error, NMSE_ρ, was calculated for each validation set and is defined as

$$\mathrm{NMSE}_\rho \triangleq \frac{\frac{1}{n_v} \sum_{i=1}^{n_v} \left(y_{v_i+\rho} - f_{v_i+\rho}^{-(v_i+\rho)^\star}(x_{v_i}) \right)^2}{\frac{1}{n} \sum_{i=1}^{n} (y_i - \bar{y})^2} \tag{23}$$

where $y_{v_i+\rho}$ is the $(v_i+\rho)$th point in the time-series, v_i is the index of the first point in the ith validation set, $\bar{y}$ is the average value of y_t given in Equation (12), and $f_{v_i+\rho}^{-(v_i+\rho)^\star}(x_{v_i})$ is the prediction at time $v_i+\rho$ of a model that has been iterated ρ times. As described in Section J, only the points that make up the trajectory segment surrounding the point $y_{v_i+\rho}$ were left out to generate the prediction, $f_{v_i+\rho}^{-(v_i+\rho)^\star}(x_{v_i})$.

B. Lorenz Data

The Lorenz equations are as follows:

$$\dot{x} = \sigma(y - x) \tag{24}$$

$$\dot{y} = rx - y - xz \tag{25}$$

$$\dot{z} = xy - bz \tag{26}$$

The standard values of $\sigma = 10.0$, $r = 28.0$, and $b = 8/3$ were used to generate the series. Figure 6 shows a plot of the first 500 points in the series.

[14] Vapnik has recently proposed an efficient method of optimizing parameters for a large class of models that requires much less computation [41].

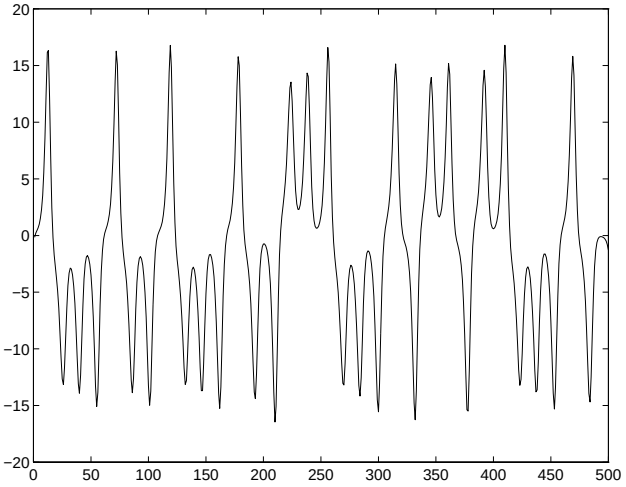

Figure 6: The first 500 points in the Lorenz time series.

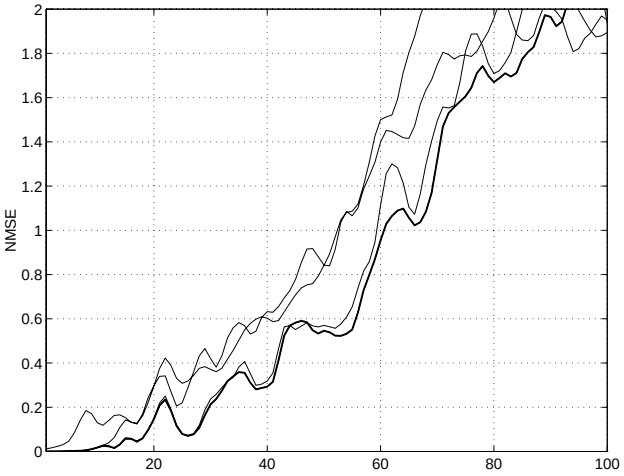

Figure 7: The NMSE_ρ versus ρ, the number of steps ahead, for the Lorenz time series. The user-selected parameters are shown in Table 1. The heavy line shows the NMSE_ρ for $k = 2$ and $\lambda = 0.48$, the values that minimized $\text{NMSE}_{1,60}$. The thin lines show the NMSE_ρ for the other values of k, 3–5, and the best value of λ for each.

A 2000-point time series was used to build the model. The user-selected parameters are shown in Table 1.

The cross-validation error was calculated for 31 different values of λ, $\{0.40, 0.42, \ldots, 1.00\}$, and 4 different values of k, $\{2\text{–}5\}$, [15]

Figure 7 shows the average prediction error as a function of the number of steps predicted ahead. This figure demonstrates the sensitivity of the model accuracy to value of k. It can also be seen that the average prediction horizon, the number of steps for which the model prediction is better than predicting $\bar{y}$, for the best values of k and λ is about 60 steps ahead. To determine the best value of λ and k, the average NMSE_ρ was taken over the first 60 values of ρ.

$$\text{NMSE}_{1,60} \triangleq \frac{1}{60} \sum_{\rho=1}^{60} \text{NMSE}_\rho \tag{27}$$

Parameter	**Value**
Model Type	Constant
Prediction Type	Averaged
Prediction Method	Iterated
Embedding Dimension(m)	25
Embedding Delay (τ)	1
Upsample Rate	5
Validation Sets (n_v)	258

Table 1: User-selected model parameters that were used to predict the Lorenz time series.

Figure 8 illustrates the sensitivity of the $\text{NMSE}_{1,60}$ to the choice of k and λ. The values $k = 2$ and $\lambda = 0.48$ had the smallest $\text{NMSE}_{1,60}$ (0.2830). The lack of smoothness emphasizes the importance of using a large number of validation sets to estimate $\text{NMSE}_{1,60}$ and optimize k and λ.

Figure 9 shows the average NMSE_ρ for the values of k and λ that minimize $\text{NMSE}_{1,60}$. The large variance in error, shown by the gray region, also illustrates the importance of using multiple validation sets to assess model accuracy. If only one validation set had been used, the prediction horizon might have appeared to be anywhere from 10 to well beyond 100.

After upsampling the time series, there were 9871 points in the construction set. Using the parameters $c = 4$ and $n_T = 5$, on average only 234 distances had to be calculated to find the two nearest trajectories to a query point. Thus, for this data set ANNA is approximately 40 times faster than the brute force approach.

Figures 10 and 11 show two 100-steps ahead predictions for the model with $k = 2$ and $\lambda = 0.48$. Neither of these series were used to construct the model or to optimize the values of k and λ.

C. Workshop Competition

The workshop time series consisted of 2000 points. By inspection, the data set appears to be noise-free (Figure 13). A plot of the power spectrum (Figure 12) indicates that the series had been sampled at approximately five times the Nyquest frequency.

[15] $k = 2$ means only the nearest trajectory is used since the kth trajectory is given a weight of zero (Section H.).

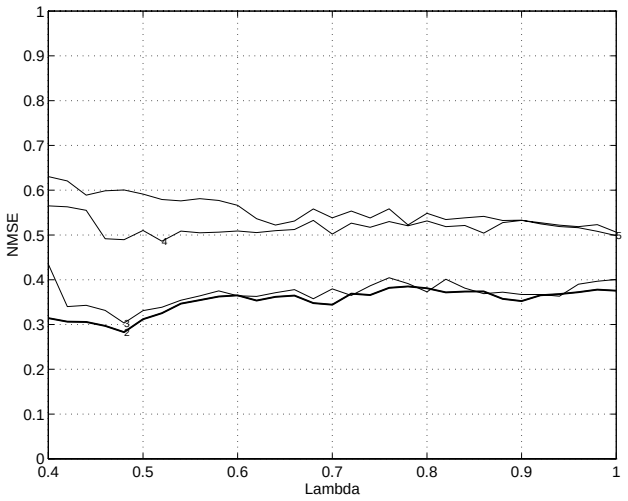

Figure 8: $\mathrm{NMSE}_{1,60}$ versus λ. The user-selected parameters are shown in Table 1. The heavy line shows the NMSE_{ρ} for $k = 2$. The thin lines show the $\mathrm{NMSE}_{1,60}$ for the other values of k which are labeled on the plot.

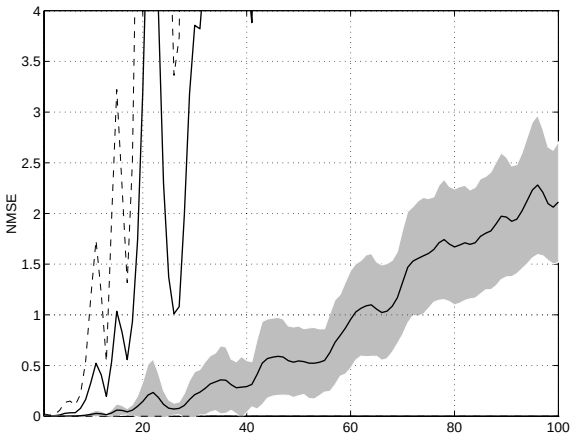

Figure 9: The NMSE_{ρ} versus ρ, the number of steps ahead, for $k = 2$ and $\lambda = 0.48$. The user-selected parameters are shown in Table 1. The line in the center of the gray region shows the *average* NMSE_{ρ} and the gray region shows three standard deviations of the average NMSE_{ρ}. The upper broken line shows the largest NMSE_{ρ} of all the validation sets and the line outside of the gray region shows three standard deviations of NMSE_{ρ}. The smallest NMSE_{ρ} of all the validation sets is too small to be visible on this plot.

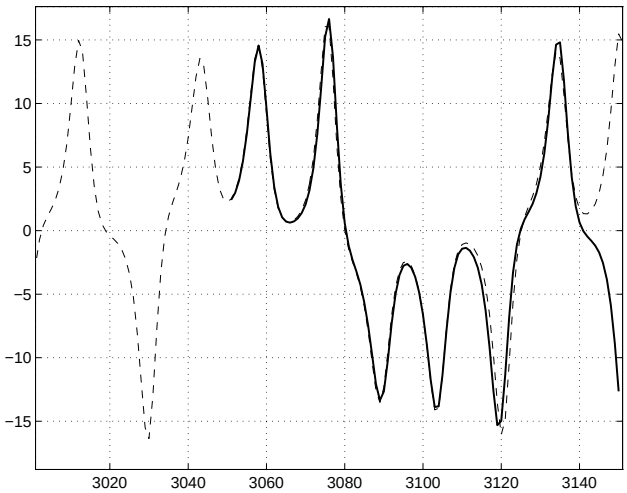

Figure 10: A 100-steps ahead prediction for the Lorenz time series. The predicted points are shown by the heavy line and the actual series is shown by the broken line. In this case the model was able to accurately predict approximately 90 steps ahead, which is better than the average prediction horizon (60 steps ahead).

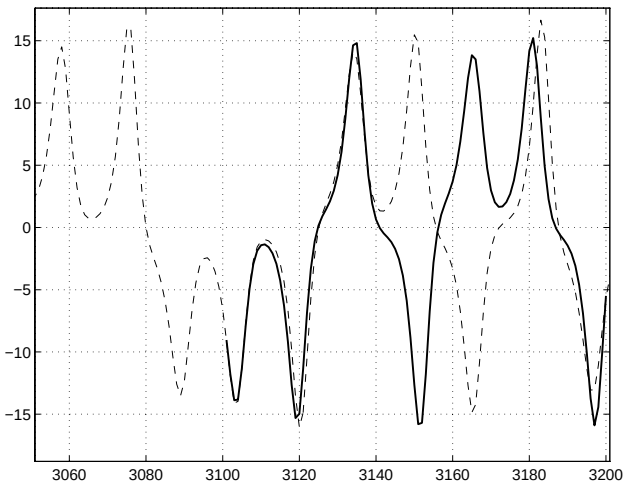

Figure 11: A 100-steps ahead prediction for the Lorenz time series. The predicted points are shown by the heavy line and the actual series is shown by the broken line. In this case the model was able to accurately predict approximately 40 steps ahead, which is worse than the average prediction horizon (60 steps ahead).

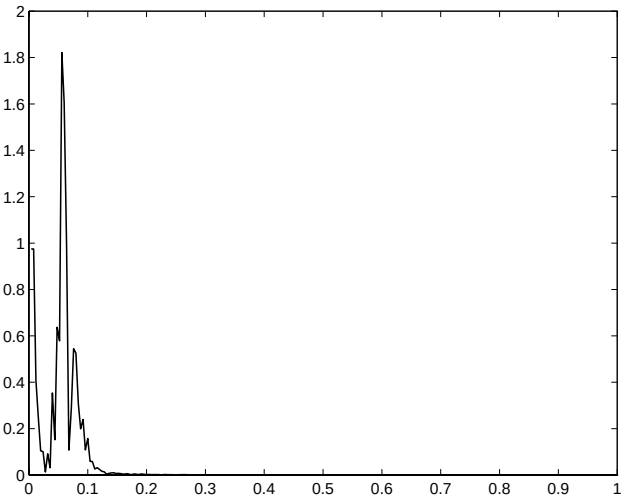

Figure 12: The estimated power spectrum of the workshop time series. The spectrum was estimated using Bartlett's method of periodogram averaging with four nonoverlapping windows [15]. The x-axis has been scaled so that 1 represents half the Nyquest frequency.

C.1. Exploiting Symmetry

Upon further inspection, the series appears to be chaotic with three unstable equilibria at approximately -0.25, 0, and 0.25, as shown in Figure 13. The period and growth of the oscillations about the upper and lower equilibria appear to be the same. This observation poses the possibility that the system that generated this time series may be symmetrical; that is, it may just as likely have generated the time series reflected about the horizontal axis.

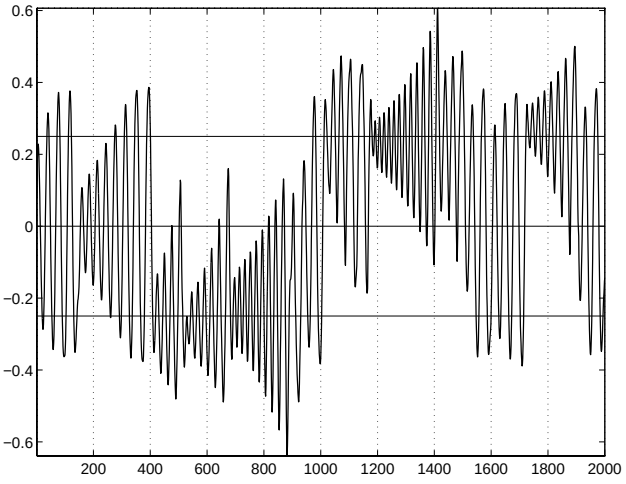

Figure 13: The competition time series. The three horizontal lines show the approximate location of the three unstable equilibria in the series.

There is further evidence that the time series is symmetrical in delay plots of the series. For example, the delay plot in Figure 14, which was created with the raw time series, is visually indistinguishable from the trajectory plot in Figure 15, which was created with the time series reflected about the horizontal axis.

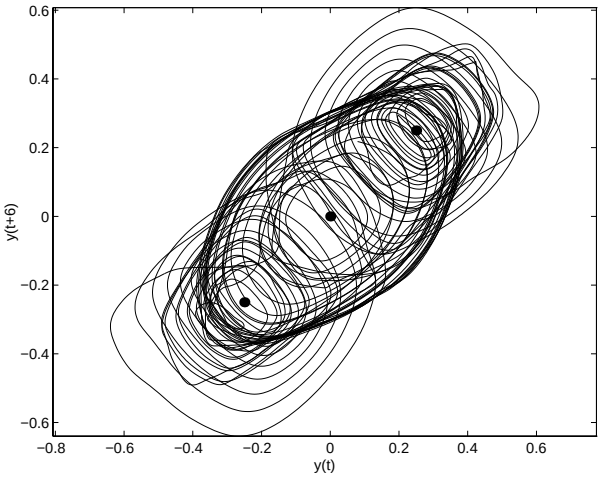

Figure 14: The workshop time series data embedded in a two-dimensional space: y_{t+6} verses y_t. The time series was upsampled by a factor of 10 to clearly show the continuous path of the trajectory. The three dots represent the approximate locations of the equilibria.

If the system that generated the original time series could have generated the reflected time series, a more accurate model could be built using the combination of both time series. To determine if this was reasonable for the purpose of prediction, the cross-validation error was compared for two models: one built with the original time series and the other built with the combined time series. Table 2 shows the user-selected parameters that were used for the comparison.

Parameter	**Value**
Model Type	Constant
Prediction Type	Integrated
Prediction Method	Iterated
Embedding Dimension (m)	50
Embedding Delay (τ)	2
Validation Sets (n_v)	229

Table 2: User-selected model parameters for analyzing the effect of combining the original time series with the reflected time series.

Figure 17 shows the NMSE_ρ as a function of ρ using only the original data to construct the model. The average prediction horizon is approximately 60 steps with the best values of λ and k. Figure 16 shows the NMSE_ρ using the combined time series. The aver-

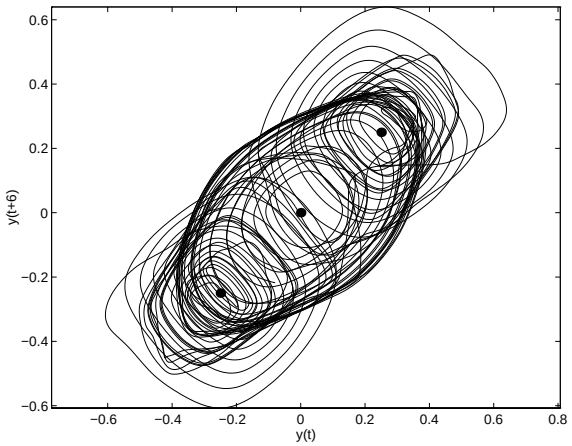

Figure 15: The workshop time series data *reflected* about the horizontal axis and embedded in a two-dimensional space: y_{t+6} verses y_t. The time series was upsampled by a factor of 10 to clearly show the continuous path of the trajectory. The three dots represent the approximate locations of the equilibria.

age prediction horizon with the combined time series is approximately 80 steps. The model built with the combined time series has an average NMSE_ρ over the first 80 steps (0.4712),

$$\text{NMSE}_{1,80} \triangleq \frac{1}{80} \sum_{\rho=1}^{80} \text{NMSE}_\rho \tag{28}$$

that is approximately 16% better than the model built with just the original time series (0.5637).

C.2. Sensitivity to Optimized Parameters

Parameter	**Value**
Model Type	Constant
Prediction Type	Integrated
Prediction Method	Iterated
Embedding Dimension (m)	50
Embedding Delay (τ)	2
Combined Time Series	Yes
Validation Sets (n_v)	229

Table 3: User-selected model parameters for analyzing the sensitivity to the values of k and λ.

To measure the sensitivity of the model accuracy to the parameters k and λ, the NMSE_ρ was calculated for 31 different values of λ, $\{0.40,\ 0.42,\ \ldots,\ 1.00\}$, and 5 different values of k, $\{2\text{–}5\}$. Figure 18 shows the $\text{NMSE}_{1,80}$ as a function of λ for a model with the user-selected parameters shown in Table 3. The large variation in $\text{NMSE}_{1,80}$ illustrates a central point in this

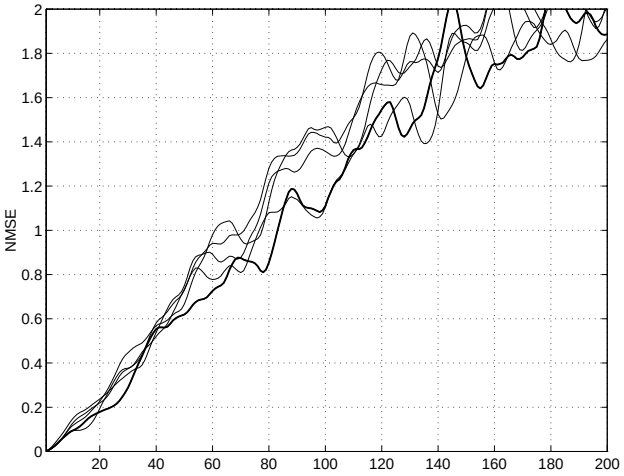

Figure 16: NMSE_ρ as a function of ρ for a model built with the combined time series. The user-selected parameters are shown in Table 2. The heavy line shows the NMSE_ρ for $k = 2$ and $\lambda = 0.74$. The other lines show the NMSE_ρ for four other values of k, 3–6, and the best λ for each.

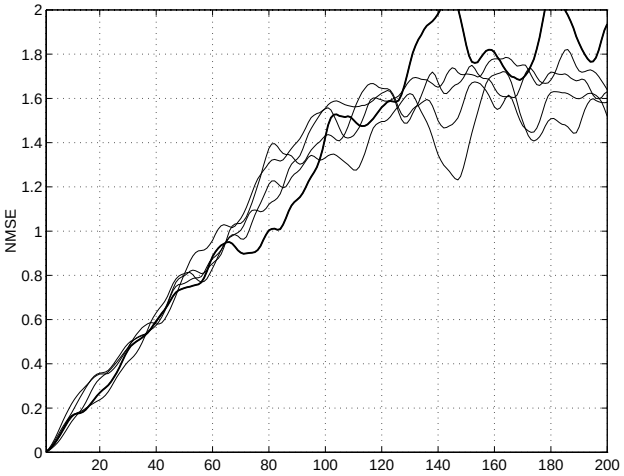

Figure 17: The NMSE_ρ versus ρ for a model built with the original time series. The user-selected parameters are shown in Table 2. The heavy line shows the NMSE_ρ for $k = 2$ and $\lambda = 0.74$. The other lines show the NMSE_ρ for four other values of k, 3–6, and the best λ for each.

paper—the model accuracy is sensitive to the choices of k and λ.

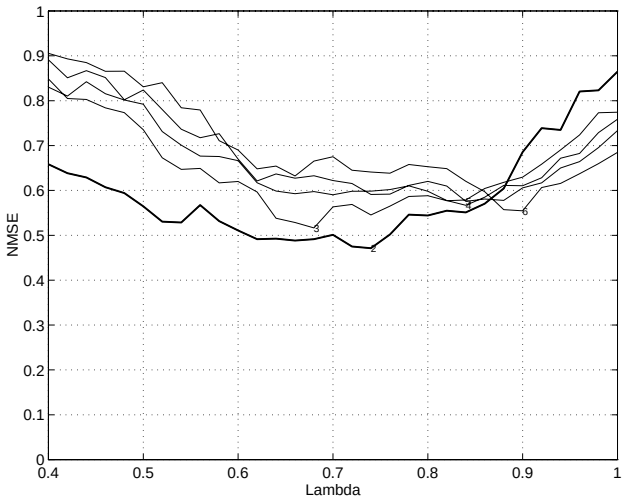

Figure 18: $\text{NMSE}_{1,80}$ versus λ. The user-selected parameters are shown in Table 3. The heavy line shows the NMSE_ρ for $k = 2$. The thin lines show the NMSE_ρ for other values of k, 3–6.

C.3. Performance of ANNA

For this time series, the construction set consisted of 3800 points. Using the parameters $c = 4$ and $n_T = 5$, on average only 180 distances were calculated to find the two nearest trajectories to a query point q. For this data set, ANNA was approximately 20 times faster than the brute force approach.

C.4. Prediction Horizon

Figure 19, which shows the average NMSE_ρ for the values of k and λ that minimized $\text{NMSE}_{1,80}$, illustrates three important points. First, it shows one of the disadvantages of iterated prediction—if the model is trying to predict too many steps ahead, the average error will be larger than simply predicting $\bar{y}$. As discussed in Section D, this problem is caused by the accumulated errors in the input vector. If the average prediction error is the appropriate measure of accuracy for the application, the model prediction should be replaced with $\bar{y}$ for points beyond the prediction horizon.

Second, Figure 19 illustrates the sensitivity of the prediction error to the region of the time series in which the prediction is being made. This can be seen by observing the large variance in the prediction errors. The worst NMSE_ρ exceeds 1 after 4 steps; the best NMSE_ρ is too small to be observed on this plot. Essentially this illustrates that the prediction horizon is very sensitive to the state of the system, which has been observed by other researchers[16] [8, 31, 33].

[16] See Nese for a good discussion on this topic [31].

Third, this figure illustrates the importance of using a large number of validation sets, n_v, to assess the average prediction horizon. This can be observed by noting that the thickness of the gray region, which can be loosely interpreted as a 99.8% confidence region[17], is inversely proportional to $\sqrt{n_v}$.

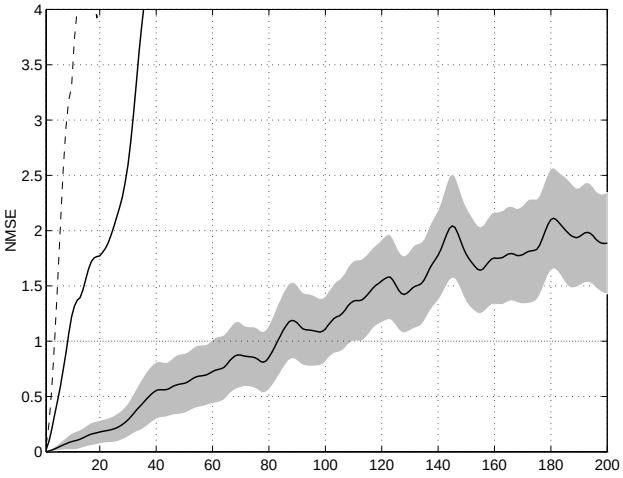

Figure 19: The average NMSE_ρ versus ρ for the parameters shown in Table 3, $k = 2$ and $\lambda = 0.74$. The gray region shows three standard deviations in the estimate of the *mean* error. The upper broken line shows the worse-case error and the upper solid line shows three standard deviations in the error. The minimum error is not visible on this scale.

Figure 20 shows the 200-steps ahead predictions for the five smallest values of $\text{NMSE}_{1,80}$, which are given in Table 4 along with the corresponding values of k and λ. There is agreement among the predictions for the first 80 steps, which is consistent with the average prediction horizon that was found by cross-validation.

k	λ	$\text{NMSE}_{\mathbf{1,80}}$
2	0.74	0.4712
2	0.72	0.4750
2	0.66	0.4882
2	0.68	0.4913
2	0.62	0.4914

Table 4: The smallest five values of $\text{NMSE}_{1,80}$ and the corresponding values of k and λ.

[17] If the errors in the numerator summation of Equation (23) are independent, then the central limit theorem applies and the distribution of NMSE_ρ is approximately normal for n_v suitably large. The assumption of independence is appropriate if the validation sets are spaced sufficiently far apart and the system is chaotic.

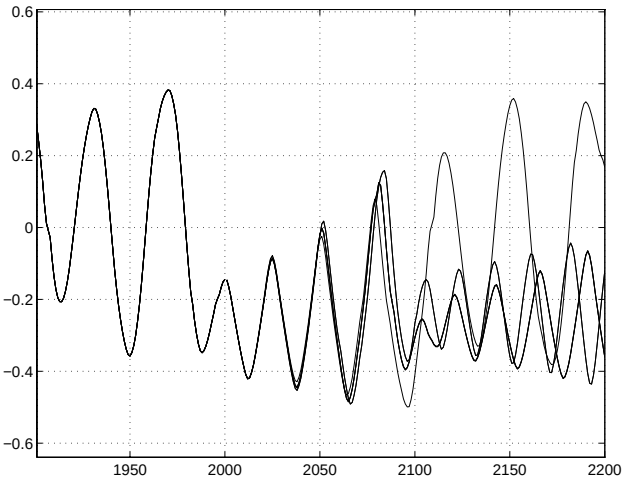

Figure 20: Five 200-steps ahead predictions from the five best models. The user-selected parameters are shown in Table 3. The five smallest values of $\mathrm{NMSE}_{1,80}$ are shown in Table 4.

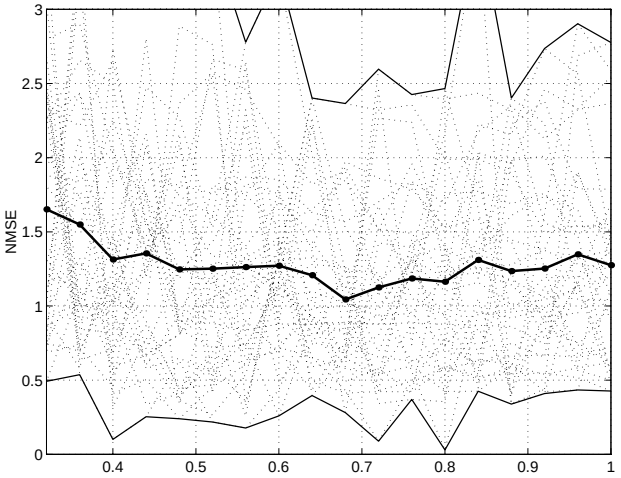

Figure 21: $\mathrm{NMSE}_{1,200}$ as a function of λ for $k = 2$. The error for each of the 33 validation sets is shown by the broken lines. The lines at the top and bottom of the plot show the maximum error and minimum error, respectively, over all 33 validation sets.

C.5. Competition Entry

As in many practical situations, there was relatively little time to construct a model for the competition. Consequently, a relatively narrow range of different models could be considered, which compromised model accuracy. The $\mathrm{NMSE}_{1,200}$ was calculated for 31 different values of λ, $\{0.40, 0.42, \ldots, 1.00\}$, and 5 different values of k, $\{2\text{–}5\}$. Invariably $k = 2$ had the smallest error. Only local constant models using integrated prediction were investigated for the competition. Local linear models or local constant models using averaged prediction may be more accurate.

The user-selected parameters are shown in Table 5. Only 33 validation sets were used, {201–400, 250–450, ..., 1801–2000}, to reduce the computational cost. Unlike the previous sections, leave-200-out cross-validation was used instead of omitting out only the local trajectory segment.

Parameter	**Value**
Model Type	Constant
Prediction Type	Integrated
Prediction Method	Iterated
Embedding Dimension (m)	50
Embedding Delay (τ)	2
Combined Time Series	Yes
Validation Sets (n_v)	33

Table 5: User-selected model parameters that were used for the competition entry.

Figure 21 shows the average $\mathrm{NMSE}_{1,200}$ as a function of λ. The minimum $\mathrm{NMSE}_{1,200}$, 1.044, was obtained with $\lambda = 0.68$ and $k = 2$. As discussed in section D, the best prediction to minimize *average* error after the average prediction horizon is the constant $\bar{y}$. However, the prediction horizon for any given region of the time series may be longer or shorter than the average, as is illustrated in Figure 21. Since similar entries were likely to be submitted in the competition, it was probable that at least one of them would have a longer prediction horizon than the average, consequently having a smaller error than an entry which predicted $\bar{y}$ after the average prediction horizon. For this reason, the model output was used for all 200 pre-

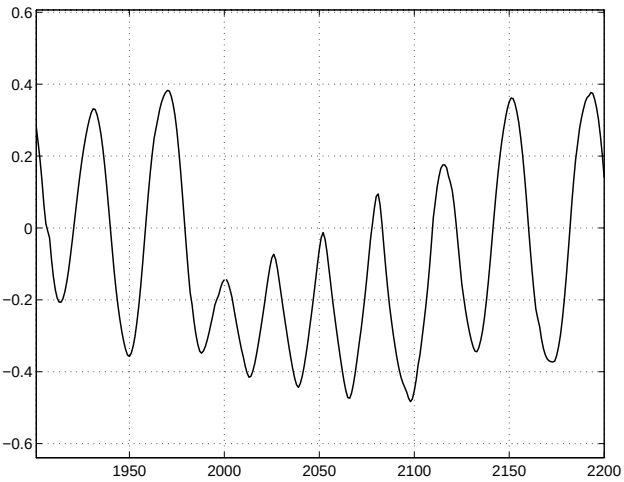

Figure 22: The last 200 points shows the model prediction that was entered in the competition.

dicted points even though this approach is worse on average. The competition entry is shown in Figure 22.

References

[1] Stelios G. Bakamidis. An exact fast nearest neighbor identification technique. In *1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages 658–661, April 1993.

[2] Th. Buzug and G. Pfister. Comparison of algorithms calculating optimal embedding parameters for delay time coordinates. *Physica D*, 58:127–137, 1992.

[3] Liangyue Cao. Practical method for determining the minimum embedding dimension of a scalar time series. *Physica D*, 110:43–50, 1997.

[4] Martin Casdagli. Nonlinear prediction of chaotic time series. *Physica D*, 35:335–356, 1989.

[5] Martin Casdagli, Deirdre Des Jardins, Stephen Eubank, J. Doyne Farmer, John Gibson, and James Theiler. Nonlinear modeling of chaotic time series: Theory and applications. In Jong Hyun Kim and John Stringer, editors, *Applied Chaos*, pages 335–380. John Wiley & Sons, Inc., 1992.

[6] Martin C. Casdagli and Andreas S. Weigend. Exploring the continuum between deterministic and stochastic modeling. In Andreas S. Weigend and Neil A. Gershenfeld, editors, *Time Series Prediction*, Santa Fe Institute Studies in the Sciences of Complexity, pages 347–366. Addison-Wesley, 1994.

[7] Chris Chatfield. *The Analysis of Time Series, An Introduction*. Texts in Statistical Science. Chapman & Hall, fifth edition, 1996.

[8] J. D. Farmer and John J. Sidorowich. Exploiting chaos to predict the future and reduce noise. In Yee Chung Lee, editor, *Evolution, Learning and Cognition*, pages 277–330. World Scientific, 1988.

[9] J. Doyne Farmer and John J. Sidorowich. Predicting chaotic time series. *Physical Review Letters*, 59(8):845–848, August 1987.

[10] Jerome H. Friedman. An overview of predictive learning and function approximation. In Vladimir Cherkassky, Jerome H. Friedman, and Harry Wechsler, editors, *From Statistics to Neural Networks*, volume 136 of *Computer and Systems Sciences*, pages 1–61. Springer-Verlag, 1991.

[11] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, September 1977.

[12] Keinosuke Fukunaga and Patrenahalli M. Narendra. A branch and bound algorithm for computing k-nearest neighbors. *IEEE Transactions on Computers*, C-24:750–753, July 1975.

[13] P. García, J. Jiménez, A. Marcano, and F. Moleiro. Local optimal metrics and nonlinear modeling of chaotic time series. *Physical Review Letters*, 76(9):1449–1452, February 1996.

[14] Neil A. Gershenfeld and Andreas S. Weigend. The future of time series: Learning and understanding. In Andreas S. Weigend and Neil A. Gershenfeld, editors, *Time Series Prediction*, Santa Fe Institue Studies in the Sciences of Complexity, pages 1–70. Addison-Wesley, 1994.

[15] Monson H. Hayes. *Statistical Digital Signal Processing and Modeling*. John Wiley & Sons, Inc., 1996.

[16] Qiyuan Jiang and Wenshu Zhang. An improved method for finding nearest neighbors. *Pattern Recognition Letters*, 14:531–535, July 1993.

[17] Iraj Kalantari and Gerard McDonald. A data structure and an algorithm for the nearest point problem. *IEEE Transactions on Software Engineering*, SE-9(5):631–634, September 1983.

[18] Behrooz Kamgar-Parsi and Laveen N. Kanal. An improved branch and bound algorithm for computing k-nearest neighbors. *Pattern Recognition Letters*, 3:7–12, January 1985.

[19] Holger Kantz and Lars Jaeger. Improved cost functions for modelling of noisy chaotic time series. *Physica D*, 109:59–69, 1997.

[20] Baek S. Kim and Song B. Park. A fast k nearest neighbor finding algorithm based on the ordered partition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):761–766, November 1986.

[21] D. Kugiumtzis. State space reconstruction parameters in the analysis of chaotic time series — the role of the time window length. *Physica D*, 95:13–28, 1996.

[22] D. Kugiumtzis. Assessing different norms in nonlinear analysis of noisy time series. *Physica D*, 105:62–78, 1997.

[23] Soren Larsen and Laveen N. Kanal. Analysis of k-nearest neighbor branch and bound rules. *Pattern Recognition Letters*, 4:71–77, April 1986.

[24] D. Kugiumtzis Lillekjendlie and N. Christophersen. Chaotic time series. part ii. system identification and prediction. *Modeling, Identification and Control*, 15(4):225–243, October 1994.

[25] Zhong Liu, Xiaolin Ren, and Zhiwen Zhu. Equivalence between different local prediction methods of chaotic time series. *Physics Letters A*, 227:37–40, March 1997.

[26] Philip Lockwood. A low cost dtw-based discrete utterance recogniser. In *Eighth International Conference on Pattern Recognition*, pages 467–469, 1986.

[27] Luisa Micó, José Oncina, and Rafael C. Carrasco. A fast branch & bound nearest neighbor classifier in metric spaces. *Pattern Recognition Letters*, 17:731–739, 1996.

[28] María Luisa Micó, José Oncina, and Enrique Vidal. A new version of the nearest-neighbor approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements. *Pattern Recognition Letters*, 15:9–17, January 1994.

[29] Daniel B. Murray. Forecasting a chaotic time series using an improved metric for embedding space. *Physica D*, 68:318–325, 1993.

[30] Sameer A. Nene and Shree K. Nayar. A simple algorithm for nearest neighbor search in high dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9):989–1003, September 1997.

[31] Jon M. Nese. Quantifying local predictability in phase space. *Physica D*, 35:237–250, 1989.

[32] H. Niemann and R. Goppert. An efficient branch-and-bound nearest neighbor classifier. *Pattern Recognition Letters*, 7:67–72, February 1988.

[33] Liming W. Salvino, Robert Cawley, Celso Grebogi, and James A. Yorke. Predictability in time series. *Physics Letters A*, 209:327–332, December 1995.

[34] Tim Sauer. Time series prediction by using delay coordinate embedding. In Andreas S. Weigend and Neil A. Gershenfeld, editors, *Time Series Prediction*, Santa Fe Institue Studies in the Sciences of Complexity, pages 175–193. Addison-Wesley, 1994.

[35] Tim Sauer, James A. Yorke, and Martin Casdagli. Embedology. *Journal of Statistical Physics*, 65(3):579–616, 1991.

[36] Jeffrey S. Simonoff. *Smoothing Methods in Statistics*. Springer Series in Statistics. Springer-Verlag, 1996.

[37] Leonard A. Smith. Local optimal prediction: exploiting strangeness and the variation of sensitivity to initial condition. In *Philosophical Transactions of the Royal Society*, volume 348 of *A*, pages 371–381, 1994.

[38] S. C. Tai, C. C. Lai, and Y. C. Lin. Two fast nearest neighbor searching algorithms for image vector quantization. *IEEE Transactions on Communications*, 44(12):1623–1628, December 1996.

[39] F. Takens. Detecting strange attractors in turbulence. In D. A. Rand and L. S. Young, editors, *Dynamical Systems and Turbulence*, volume 898 of *Lecture Notes in Mathematics*, pages 336–381. Springer-Verlag, 1981.

[40] Naoki Tanaka, Hiroshi Okamoto, and Masayoshi Naito. An optimal metric for predicting chaotic time series. *Japanese Journal of Applied Physics*, 34(1):388–394, January 1995.

[41] Vladimir Naumovich Vapnik. *The nature of statistical learning theory*. Springer-Verlag, 1995.

[42] Enrique Vidal. An algorithm for finding nearest neighbors in (approximately) constant average time. *Pattern Recognition Letters*, 4:145–157, July 1986.

[43] M. P. Wand and M. C. Jones. *Kernel Smoothing*. Number 60 in Monographs on Statistics and Applied Probability. Chapman & Hall, 1995.