Large-scale Automated Forecasting using Fractals

Deepayan Chakrabarti
Center For Automated Learning and Discovery
deepay@cs.cmu.edu

April 20, 2002
CMU-CALD-02-101

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Forecasting has attracted a lot of research interest, with very successful methods for periodic time sequences. Here, we propose a fast, automated method to do non-linear forecasting, for both periodic as well as *chaotic* time sequences. We use the technique of *delay coordinate embedding*, which needs several parameters; our contribution is the automated way of setting these parameters, using the concept of 'intrinsic dimensionality'. Our operational system has fast and scalable algorithms for preprocessing and, using R-trees, also has fast methods for forecasting. The result of this work is a black-box which, given a time series as input, finds the best parameter settings, and generates a prediction system. Tests on real and synthetic data show that our system achieves low error, while it can handle arbitrarily large datasets.

Keywords: Time Series Forecasting, Fractals, Lag Plots, State Space Models

Contents

1 Introduction

Scientific research on physical systems is often based on the measurement of observables over time. Such data is the basis of characterization of the system and prediction of its future behaviour. If there are known underlying deterministic equations, they can be analyzed and system properties can be determined analytically or through simulations. But, in many real-world cases, the underlying equations for the system are unknown. Under such circumstances, regularities in the past have to be used as guides to understanding the system and forecasting the future.

To describe the time series prediction problem in a concrete fashion, some terminology is required.

Terminology We will follow the terminology used Abarbanel ([Aba95]). Formally, the terms involved are defined as follows:

- *Dynamical System:* This is a physical system which evolves over time.
- *System State:* This is the underlying state of the dynamical system under consideration, and is an unknown quantity.
- *Observable:* This is a function of the system state which can be be measured directly, or can be calculated from other measurements. If the system returns to the same state later, the observable will have the same value.
- *Time Series:* This is a list of values taken by the observable over time.

Using this terminology, we can give the following problem definition.

Problem Definition Given the sequence of values taken by an observable over time, a forecasting system attempts to predict the observable's values over some future time period. More formally, the precise definition is as follows:

Problem (Predict-1). Given *a time sequence* $x_1, x_2, \ldots, x_t$, **predict** *the value of* x_{t+1}.

This problem definition can be generalised to the following problem.

Problem (Predict-n). *The training set (also called the historical data, for example, stocks) is a set of time sequences generated by the* same *physical system over separate periods of time.*

$$TS = X_1, \ldots, X_N \tag{1}$$

where $X_i = x_{t_i}, x_{t_i+1}, \ldots, x_{t_i+(l_i-1)}$. *Here,* x_t *is the value of the time series at time* t, *and* l_i *is the length of sequence* X_i. *Also, the forecasting system is provided with the query sequence* $Y = y_1, \ldots, y_l$ *from the same physical system, which must be forecasted; that is, we need to find* $y_{l+1}, y_{l+2}, \ldots$ *and so on. Thus, we have several training sequences, and one query sequence.*

Some example time series are shown in Figure 1.

Applications Analysis of time series for prediction purposes is a rich field, with a wide variety of applications. The following gives a snapshot of the areas where time series have shown up ([AN93, FBC95]):

- *Physics:* Measurements of laser fluctuations.
- *Biology:* Physiological data, such as heart rate, blood oxygen concentration, and EEG state, of patients with sleep apnea.
- *Economics:* High-frequency currency exchange rate data, and daily stock quote data.
- *Astrophysics:* Intensity variations of white dwarf stars, prediction of solar activity.
- *Geophysics:* Measurements of magnetic storm data.
- *Transportation:* Highway usage-over-time data.

One application area which is of increasing interest are sensor networks ([GKS01, BGS01, BW01]). This envisages a large number of sensors being distributed in the environment. Each sensor will be collecting data independently, and performing Each sensor will be collecting time sequence data, and it will be doing forecasting, and thus be able to spot outliers and find patterns. In such an environment, human intervention and manual setting of parameters is out of the question, because sensors may be operating in a human-hostile environment, with low communication bandwidth for shipping data and getting human feedback.

Overview of Article Many techniques have been suggested, but they are either too simplistic (such as linear models like ARMA), or require long training times and large number of parameters (such as Neural Networks). Our approach uses a forecasting method called "Delay Coordinate Embedding". This has the advantage of being firmly grounded in theory, and can handle periodic as well as chaotic datasets. Its disadvantage was that its parameters had to be set manually. Our F4 (**F**ractal **FORE**casting) system provides automatic methods to do this, without any human intervention. This results in a black-box which, given any time series, can automatically find the optimal parameters and build a prediction system. Since there is no

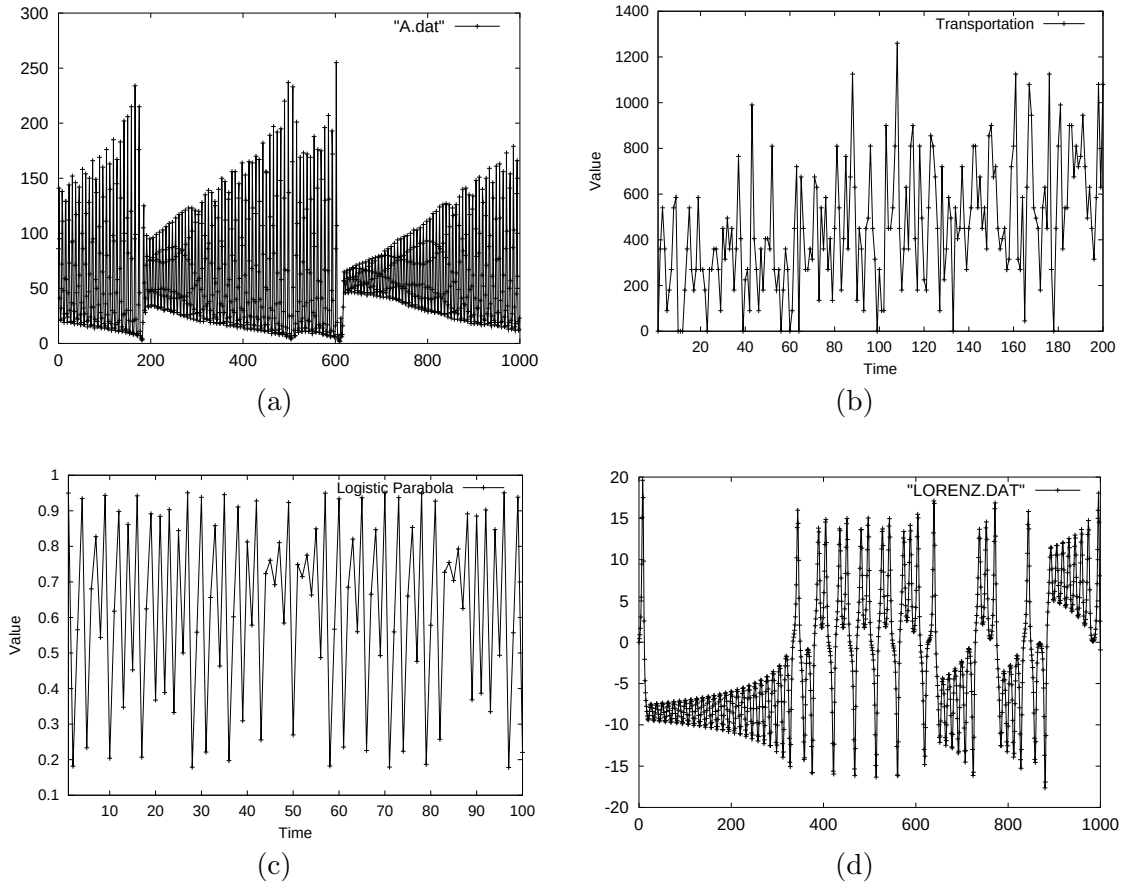


Figure 1: *Example time series*: Plot (a) shows fluctuations in a laser over time. Plot (b) shows the number of vehicles crossing a point on a California highway, over time. Plot (c) is the *Logistic Parabola* dataset. Plot (d) shows the LORENZ dataset over some stretch of time. Most of these datasets will be discussed later.

need for human intervention in the entire prediction process, our method can be easily applied in the sensor network setting.

The rest of the article is as follows: we study the related work in Section 2. Then we give a background on the techniques which we use in our proposed method, such as lag plots and fractal dimensions, in Section 3. This is followed by a detailed description of the proposed method in Section 4, and the results are described in Section 5. We finally present our conclusions in Section 6.

2 Related Work

Time-series prediction is a well-studied problem. Prior to the 1920s, forecasting was carried out by extrapolating the series through a global fit in the time domain. This changed with the appearance of linear time series models. They had two very desirable features: they were easy to understand and analyse, and they were simple to implement. This led to their usage for a long period of time. The problem with linear models was the obvious one: they could not work well on time series generated by non-linear processes. Hence linear models have recently been replaced to some extent by non-linear models. While these are more successful at fitting complicated time sequences, understanding them is tougher. We give detailed descriptions of both linear and non-linear forecasting techniques below.

Apart from forecasting, time sequences have also been studied from other points of view. The databases community has primarily looked at methods for time series indexing for fast similarity searches. These methods turned out to suffer from the *curse of dimensionality*, which has prompted research to reduce dimensionality. All these are also surveyed below.

2.1 Linear Prediction

We take most of our material from Weigend and Gershenfeld ([AN93]). Literature on this field is vast, and some good references are Chatfield ([Cha89]) and Box and Jenkins ([BJ76]).

Linear models attempt to explain a time series as a linear combination of external inputs and/or internal memory. These lead to the several possible models, such as AR, MA, ARMA, ARIMA, ARFIMA and so on. The most representative of these are discussed below.

2.1.1 Moving Average (MA) models

These models assume that the system receives some external impulses which change the state of the system. Let the external input series be e_t and the output series be x_t.

$$x_t = \sum_{n=0}^{N} b_n e_{t-n} = b_0 e_t + b_1 e_{t-1} + \ldots + b_N e_{t-N} \tag{2}$$

This is called the Moving Average (MA) model. The origin of this terminology can be seen if one pictures a simple smoothing filter which averages the last few values of the series e. Engineers call this the *finite pulse response* (FIR) model, because the output is guaranteed to go to zero at N time steps after the input becomes zero.

MA filters operate in an open loop, without any feedback. To give the system some internal dynamics, some memory property is needed. This is done by the Autoregressive (AR) models.

2.1.2 Autoregresive (AR) models

Here, the time series x_t is described by the following equation:

$$x_t = \sum_{m=1}^{M} a_m x_{t-m} + e_t \tag{3}$$

This is called the Mth-order autoregressive model (AR(M)). It is also called an *infinite impulse response* (IIR) filter, because the output can continue after the input ceases. To generate a specific instance of a series, some initial conditions must be specified, usually the first M values of the series x_t. Still, the input term e_t is crucial for the life of an AR model. Without any input, the output will finally either decay to zero, or diverge, or oscillate periodically.

2.1.3 ARMA models

The obvious way to increase complexity of the models is by combining AR and MA models. This leads to the ARMA(M,N) model:

$$x_t = \sum_{m=1}^{M} a_m x_{t-m} + \sum_{n=0}^{N} b_n e_{t-n} \tag{4}$$

ARMA models dominated all areas of time series analysis and discrete-time signal processing for more than half a century.

To fit a linear model to a time series, the order of the model (that is, for example, the values M and N in an ARMA(M,N) model) had to be selected. Heuristics were developed to do this, the most famous one being the Akaike Information Criterion ([Aka70]) also known as the AIC.

Another technique is to use cross-validation, by holding back some of the training data to test the performance of competing models.

The major failure of linear models was the assumption of linearity they made about the data. Linear models can fail even for simple non-linearities. This lead to the development of non-linear models, which are discussed below.

2.2 Non-Linear Prediction

Non-linear models relax the strong assumption of linearity of the data. This allows them to explain a wide variety of data. The price for this is the increase in number of parameters, as well as lack of clear interpretations of the model at times. But with the increase in computer processing power, model-fitting for such complicated models has become feasible.

There are a variety of non-linear models in use today. We survey some of the more widespread ones. Our descriptions follow that of Weigend and Gerschenfeld ([AN93]), who describe the entries in the Time Series Prediction competition at Santa Fe in 1992. We look at some of these techniques now.

2.2.1 Artificial Neural Networks

Artificial neural networks is a huge field in itself, and we will not attempt to go into its details. We shall only describe modifications that need to be made to the basic backpropagation network to make it applicable to time series prediction. For background knowledge on neural networks, the interested reader is referred to Rumelhart's work ([RDGC93]).

Neural networks have been applied to time series forecasting by several researchers ([Wer74, Wer88, LF87, WHR90]). Here we describe an approach by Wan ([Wan93]). This was the entry that gave the best predictions on Data Set A in the Santa Fe time series prediction competition. Wan modified the standard neural network design so that each network weight and neuron input is a vector instead of a scalar. The vector neuron input encodes previous values of the time series. The vector dot product $\mathbf{w}_{ij}^l.\mathbf{x}_i^l(k)$ is used instead of a scalar product (neuron i is connected to neuron j in layer l, k is the current time). A vector generalization of the backpropagation algorithm is used for learning from such networks. Another network structure for time series prediction is called the *Time-Delay Neural Network* [LH88, Wai89]. In these too, embedded time delays are incorporated into the network structure. A taxonomy of neural network structures for time-series processing is provided in [Moz93]. But all these methods suffer from the traditional problems associated with neural networks: long training times and large number of parameters. In fact, in the case of Wan's algorithm ([Wan93]), there were $1,105$ parameters to fit $1,000$ data points. This means that the risk of *overfitting* is

very great. Overfitting is another of the problems that crop up when the size of the parameter set increases (that is, as the model gets weaker).

2.2.2 Hidden Markov Models

Hidden Markov Models(HMMs) have also been used [FD93]. The hidden states in this model are meant to represent the *internal state* of the dynamical system. Discrete HMMs are not suitable for problems involving continuous data, so, a modified class of models called mixed-state HMMs are used. But, either these models do not have non-linearities or the mathematics becomes too complicated to easily apply the forward-backward algorithm for finding model parameters. Also, the forward-backward algorithm for finding parameters in such models is $O(N^2)$, and hence not scalable to large datasets.

There are also several other methods for non-linear forecasting, which are not as popular. One is called the "Method of Analogues" ([KL93]). This approach is simple and has few free parameters, but works only for low-dimensional chaotic attractors, and only for predictions over a short period of time.

Later, in Section 3.2, we will describe the technique that we use in our work. There, we shall point out the advantages of our technique of those described in this Section.

2.3 Other Time Series Work

Time sequences have been studied from several points of view, other than prediction. Some recent surveys on time sequence similarity and indexing are by Gunopoulos ([GD01]) and Tsotras ([ST99]). One problem of particular relevance to the database community has been the storage and indexing of large time series datasets for supporting fast similarity search. Similarity search is an important aspect of our technique too, and hence is relevant for us. Since the sequences to be stored are typically high-dimensional data, spatial databases such as R-Trees ([Gut84]) and variants have been used. But such structures cannot efficiently handle very high-dimensional data. Hence, techniques have been sought to reduce dimensionality: Korn et al. ([KJF97]) used Singular Value Decomposition (SVD) to do this. Faloutsos et al. ([FRM94] use an approach called "Segmented Means" in which the time sequence is subdivided in equal-sized windows, and the means of the data in those windows are stored. The authors give techniques to get carry out exact k-nearest-neighbor and range queries in such situations. Chakrabarti et al. ([CM00]) attempt to find *local* correlations in the data, as opposed to global correlations, and then perform dimensionality reduction on the locally correlated clusters of data individually. In a similar vein, Keogh et al. ([KCMP01]) try to fit a set of constant value segments of varying lengths to the series, thus achieving data compression.

Another aspect of time series research has been the problem of finding rules/patterns from the series. One approach ([DLM+98]) was to form subsequences of the data, which were clustered, and then rule finding methods were used to obtain rules from the sequence of clusters.

3 Background

Our proposed method for time series forecasting uses several concepts, which we introduce in this section. We first describe the concept of a *lag plot*. These lag plots help explain the *Delay Coordinate Embedding* technique, which we use in our proposed method. We talk about some possible approaches to parameter-setting in this framework, and show their drawbacks. We finally finish this section with a tutorial on *fractals* and *fractal dimensions*, of which we make heavy use in our method.

3.1 Lag Plots

The idea of a *lag plot* is central to understanding our proposed method. We use an example to explain it. Figure 2 shows the step-by-step technique of building and using lag plots. Given a historical time series x_t, we can build a lag plot for lag $L = 1$ by plotting x_t vs x_{t-1}. This gives us a cloud of points. One interesting point is that since the values along all the axes are derived from the same time series, problems related to ill-scaled axes are eliminated. A lag plot for lag $L > 1$ simply a higher dimensional plot with $x_t, x_{t-1}, \ldots, x_{t-L}$ as the axes.

When we are given a new point (say, y_{t-1}) and need to predict the value at the next time step (say y_t), we first find the k-nearest-neighbors of y_{t-1} along the x_{t-1} axis. These nearest neighbors are then projected on to the x_t axis. Each of these projected values is an estimate for y_t. We interpolate between these values to form our final estimate for y_t. Thus, given y_{t-1}, we can forecast the value of y_t.

Extending this to higher lags ($L > 1$) is simple. In this case, we need to predict the future of a given time series, such as $y_{t-1}, y_{t-2}, \ldots, y_{t-L}$. This gives us a point on the $x_{t-1}, \ldots, x_{t-L}$ plane. We find the nearest neighbors on this plane, and project the corresponding points to the x_t plane. Interpolation of results is the same as for the $L = 1$ case.

Figure 3(a) shows the procedure over a real dataset: the *Logistic Parabola* time series. Let us draw its lag plot with a lag of $L = 1$, for a moment leaving aside the question of why $L = 1$ was chosen. The lag plot is shown in Figure 3(b). We see that a definite structure becomes apparent in the lag plot, which was hidden in the first plot. This can be exploited for prediction purposes. To predict x_{t+1} given x_t, we find all the points in the lag plot whose X-values are the k-nearest-neighbors of x_t (assuming a given k). We take the Y-values of these points,

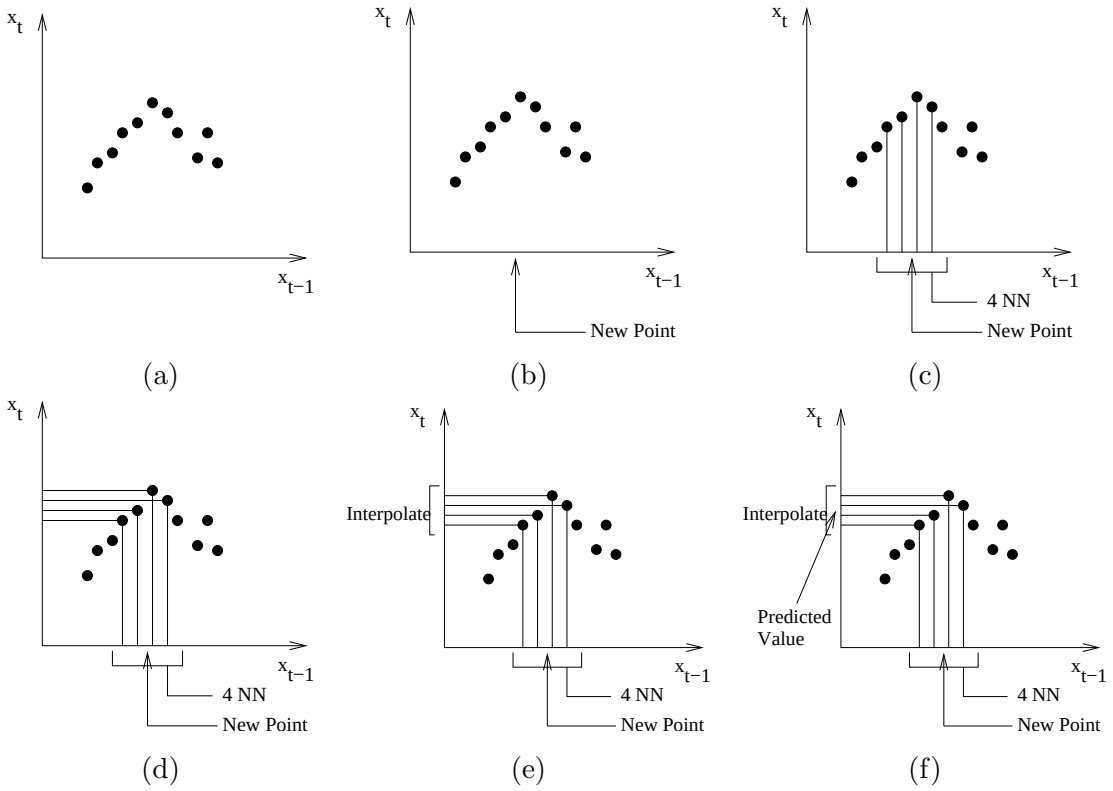
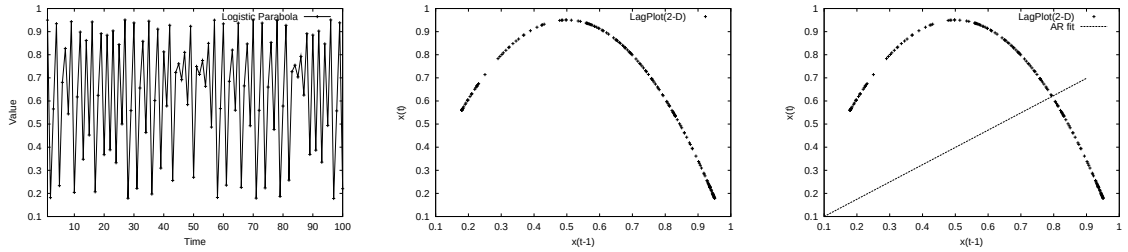


Figure 2: *Lag Plots explained:* Given a time series x_t, we generate a plot of x_t vs x_{t-1} as in (a). This is called the lag plot for lag $L = 1$. Now, the future of some new point (as shown in (b)) needs to be predicted. The k-nearest-neighbors of this point along the x_{t-1} axis are found, as in plot (c). Here, k is set to 4. The nearest neighbors are projected on to the x_t axis in (d), and the results interpolated in (e) to generate the prediction for the next value of the time series in (f).



(a) x_t vs t (b) Its lag-plot: x_{t+1} vs x_t (c) The AR fit

Figure 3: *The logistic parabola*: The equation generating this dataset is: $x_t = 3.8x_{t-1}(1 - x_{t-1}) + e_t$, where e_t is a $\mathrm{N}(0, 0.001)$ random variable. This is a chaotic dataset, a sample of which is shown in (a). This appears hard to predict, but on generating the two-dimensional lag-plot, shown in (b), we see the pattern in the data. This pattern can be used for prediction. Part (c) shows the AR fit, which is obviously not good.

and interpolate between them. The result is our predicted value of x_{t+1}. As against this, we show the fit obtained by using an autoregressive (AR) model in figure 3(c). AR attempts to minimize least-squares distance, which leads to very bad predictions on chaotic time sequences such as the Logistic Parabola, as shown.

3.2 Delay Coordinate Embedding

We first introduce certain concepts. Let the observation sequence be represented by the series x_t, which gives the value of the time series at time t. Let us define the following vector:

$$\mathbf{b} = [\mathbf{x_t}, \mathbf{x_{t-\tau}}, \mathbf{x_{t-2\tau}}, \ldots, \mathbf{x_{t-L\tau}}] \tag{5}$$

Here, τ is some real number greater than zero called the *time delay*, and L (called the lag length) is any integer greater than zero.

Definition 1. *Delay Coordinate Vector: The vector* $\mathbf{b}$ *is called a* delay coordinate vector *because its terms are the time-delayed data values from the time series.*

This allows us to define Lag Plots using this terminology.

Definition 2. *Lag Plot: Given a time series x_t and a lag L, we form all the delay coordinate vectors from x_t. A plot of this cloud of points in $(L+1)$-dimensional space is called the Lag Plot.*

There are two other closely related concepts. The time series is generated by observing the value of some observable over time. This value is a function of the internal state of the dynamical system, which is typically unknown.

Definition 3. *State Space: The space of possible values of the internal state of the system is called the State Space. The internal state may be a vector: for example, if we are forecasting levels of precipitation, then the internal state may be (pressure, temperature, humidity).*

As against this, we have the concept of *Phase Space*.

Definition 4. *Phase Space: The vector space of all the delay coordinate vectors is called the Phase Space. This is the same as the Lag Plot.*

In Section 3.1, we showed the basic method of using lag plots for time series prediction. This soundness of this method is based on a certain result (generally known as Takens' Theorem), which states that beyond some particular lag, any structure that exists in the state space is carried over faithfully into the delay-coordinate-vector space, and that it is a one-to-one transformation. Thus, properties of the state space, such as equilibrium, periodicity, Lyapunov exponents (for measuring the degree of chaos) can all be observed in delay-coordinate space. Thus, the delay coordinate vector can be a useful representation of the internal state of the dynamical system. If, at two instances along the time series, the delay coordinate vectors are same (or very similar), then the data values following the two instances should be the same (or very similar). There is predictive power in finding this one-to-one map.

For our purposes, the essential point to note is that choosing a lag greater than some particular (unknown) value is sufficient for lag plots to work. This might work for smaller dimensions also.

3.2.1 Parameter Setting Techniques

Hence, the problem of time series prediction can be divided into two subproblems:

1. Finding the right values for the lag length (L_{opt}) and number of nearest neighbors (k_{opt})
2. Prediction of future values given the current delay coordinate vector

A side-problem is that of choosing the time delay τ. There are existing methods for setting this value, but it is usually $\tau = 1$. To choose L_{opt} and k_{opt}, most current implementations resort to manual setting. This is obviously infeasible for a general-purpose forecasting system. In the following paragraphs, we outline some other possible techniques for setting L_{opt}.

False Nearest Neighbors Abarbanel ([Aba95]) suggests a method called *False Nearest Neighbours* to determinea L_{opt}. The technique can be used for periodic or chaotic systems. Suppose we have made a state-space reconstruction in dimension L. Consider a vector $\mathbf{v}$ in this space. We find its nearest neighbour, and call it $\mathbf{v}^{\mathbf{NN}}$. If $\mathbf{v}^{\mathbf{NN}}$ is a true neighbour of $\mathbf{v}$, then it arrived at the neighbourhood of $\mathbf{v}$ through dynamical origins. If it is a *false* neighbour, then it arrived by projection from a higher dimension, because the current dimension does not fully unfold the attractor. Incrementing the dimension to $d + 1$ should remove this false nearest neighbour from the neighbourhood of $\mathbf{v}$.

This suggests a technique for finding the "right" length of the delay-coordinate vector. Starting with a low dimension, we calculate the total number of false nearest neighbours for all vectors formed by the time series. As dimension increases, the number of false nearest neighbours should fall off till it reaches some constant value (which might not be zero because of noise). The minimum dimension at which this number stops falling significantly is a good estimate for the length of the delay-coordinate vector. We found this technique to be unsuitable for inclusion in a black-box prediction system, because it requires thresholding, and the results are sensitive to the threshold. Our proposed method solves this problem of choosing L_{opt}.

Cross-validation Another way to set L_{opt} is by using cross-validation. The historical data x_t is divided into a training set T and a holdout set H. The algorithm is as follows:

```
int cross-validate(D){
/* D is the entire historical dataset;
 * that is, D = x(t) over all t that we
 * have seen so far.
 * Return the estimated L(opt)
 */

  N = total number of points in D;
  Partition D into two sets T and H;
          /* T = Training set
           * H = Holdout set
           * Typically, both T and H have O(N)
           * points, and T is larger than H.
           */

  for lag L=1,2,3,... {
    Build lag plot for lag L using points
```

```
    from the training set T;
    Use this plot to predict the points
    in the holdout set H;
    Calculate error E(L) on H;
    Break the loop when E(L) has reached
    its first minimum;
  }
  return L such that E(L) is minimum;
}
```

As we describe in a later section, building up and storing a lag plot can be efficiently done by using a data structure called an R-Tree. Building up this data structure is at least an $O(N\log N)$ operation. Again, for each point that needs to be predicted in the holdout set, we need a k-nearest-neighbors query. This query is at least $O(\log N)$. Since the holdout set is $O(N)$, the total cost of prediction over the holdout set is at least $O(N\log N)$. Thus, for every step in the iteration, we spend at least $O(N\log N)$ time, perhaps more.

Even though this time complexity is superlinear, it is still acceptable. But the problem with the algorithm, as described above, is that it partitions the historical data into training and holdout sets only once. In such a case, the empirical error obtained over the holdout set is not a very good estimate of the true error; the variance is too high.

One method of reducing variance is to use the *leave-one-out cross-validation* procedure. This adds another loop in the previous algorithm, where each point in the historical data is made the holdout set, and the rest is the training set. Thus, for each lag L, we use N partitions of the data, and average the error over all the holdout sets. This leads to a better error estimate. The algorithm for this is as follows:

```
int leave-one-out-crossvalidate(D){
/* D is the entire historical dataset;
 * that is, D = x(t) over all t that we
 * have seen so far.
 * Return the estimated L(opt)
 */

  N = total number of points in D;
  for lag L=1,2,3,... {
    E(L) = 0;          /* Initialize error for lag L */
    for each t in D {
```

```
      H = {x(t)}       /* The holdout set */
      T = D - H        /* The training set */
      Build lag plot for lag L using points
      from the training set T;
      Use this plot to predict the points
      in the holdout set H;
      Calculate error E on H;
      E(L) += E;
    }

    /* Find the mean error for lag L */
    E(L) /= N;

    Break the loop when E(L) has reached
    its first minimum;
  }
  return L such that E(L) is minimum;
}
```

While the leave-one-out crossvalidation method gives us a better estimate of the true error, the extra loop increases the time complexity considerably. As we have shown in the analysis of the *cross-validate* algorithm above, the time complexity of the inner loop is $O(N\log N)$. Since we do this for each timestep t in the historical data D, the total cost for each lag is $O(N^2\log N)$. This is unacceptably high.

Hence, we see that using cross-validation can either lead to wrong results due to high variance, or give better results but at the cost of very high time complexity. Our proposed method will solve these problems by providing a means of finding the optimal lag L_{opt} in linear time on N. Also, we shall see that the by-products of our method give interesting information, which can be relevant for other data mining applications on the data.

3.3 Fractals

The fractal dimension of a cloud of points gives an estimate of the "intrinsic dimensionality" of the data in that space. For example, if we consider points along the diagonal of a cube, the intrinsic dimensionality of these points is 1 (because they actually lie along a straight line), but the extrinsic dimensionality is 3 (because the individual points are expressed in 3D-coordinates). Similarly, a parabola in two dimensions (as in Figure 3(b)) also has an intrinsic

dimensionality of 1.

There are several *fractal dimensions* existing in literature. Two of the most used ones are called the "correlation fractal dimension" and the "Hausdorff dimension". The correlation integral is obtained by plotting the number of pairs of points ($P(r)$) within a radius r, against r, on a log-log plot. For fractals, the plot consists of a horizontal part followed by an incline, and then another horizontal part. The correlation integral is the slope of the middle part. A faster way to compute it is through the Box-counting plot([BF95]). Figure 4 shows several example datasets, and their fractal dimensions. Plots (a) and (b) show the 'Circle' dataset, where the fractal dimension is found to be 1. This correctly captures the fact that given one dimension, the other dimension is automatically determined, so the intrinsic dimensionality is only 1. Plot (c) shows a randomly spread cloud of points, whose fractal dimension is found to be 2 from plot (d). Again, this is expected because the dimensions are mutually independent in this case. Plots (e) and (f) show the fractal dimension plot for a set of points called the "Sierpinski Triangle". Here, each dimension provides *some* information about the other, so the fractal dimension should be between 1 (for complete information) and 2 (for no information). It is actually found to be approximately 1.56. The reader is referred to [PJS92] for more details.

To conclude this section on fractals, there are two points of note:

1. There exist real-world datasets which have fractional dimension.
2. There are fast methods (typically $O(N)$) for calculating the fractal dimension of a cloud of N points.

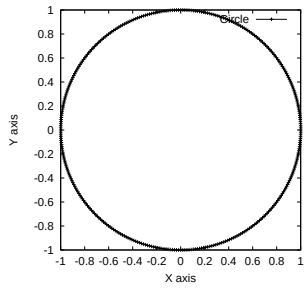
4 Proposed Method

We recap the problem definition:

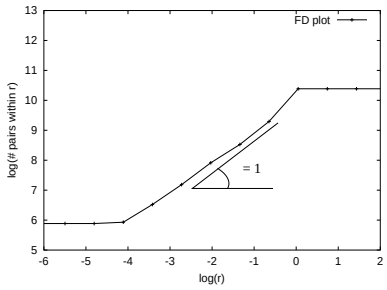
Problem (Predict-n). *The training set (also called the historical data, for example, stocks) is a set of time sequences generated by the* same *physical system over separate periods of time.*

$$TS = X_1, \ldots, X_N \tag{6}$$

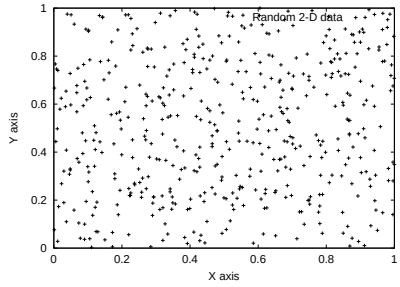
where $X_i = x_{t_i}, x_{t_i+1}, \ldots, x_{t_i+(l_i-1)}$. *Here,* x_t *is the value of the time series at time* t, *and* l_i *is the length of sequence* X_i. *Also, the forecasting system is provided with the query sequence* $Y = y_1, \ldots, y_l$ *from the same physical system, which must be forecasted; that is, we need to find* $y_{l+1}, y_{l+2}, \ldots$ *and so on. Thus, we have several training sequences, and one query sequence.*



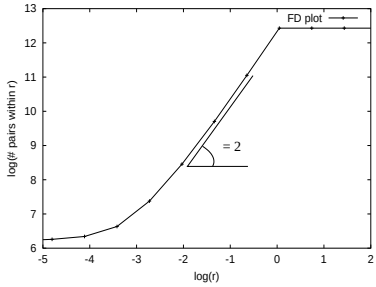
(a) A Circle dataset



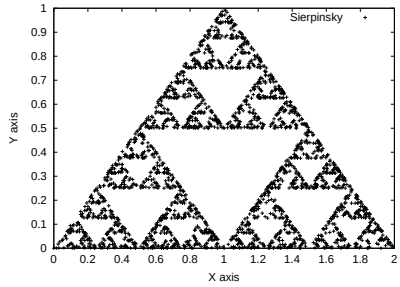
(b) Its fractal-dimension plot (slope=1)



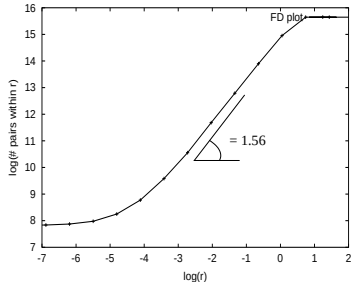
(c) A Random 2-D dataset



(d) Its fractal-dimension plot (slope=2)



(e) The "Sierpinski" triangle



(f) The fractal-dimension plot (slope=1.56)

Figure 4: *Examples of Fractal Dimension plots*: Figure (a) shows a circle of points; (b) gives its fractal-dimension plot. The slope of the graph is 1, which means that the fractal dimension of this dataset is 1. Figure (c) shows a dataset with points spread randomly in 2 dimensions. The slope from figure (d) is close to 2 as expected. Figure (e) is an example of a self-similar cloud of points. Figure (f) shows its fractal dimension to be approximately 1.56.

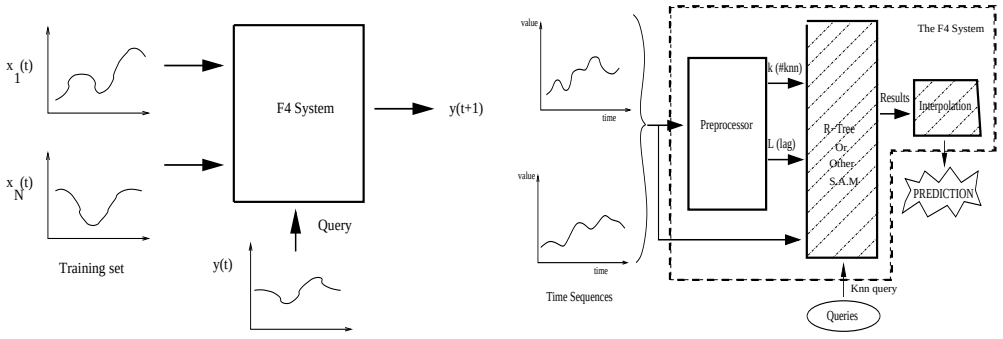


Figure 5: *The Proposed Forecasting Process*: The first figure shows the overview of the F4 system. Given a training set and a query, it forecasts the future of the query time sequence. The second figure expands the F4 system. The training set of time series is fed into a preprocessor, which outputs the "best" values of lag-length and number of nearest neighbors(k) to use. Then the time series are stored in an R-Tree using this lag-length. When queries come, the its k-nearest neighbors in lag space are retrieved. The points succeeding these neighbors are noted and interpolated to give the value for the prediction for the current query. The preprocessor itself is our novel contribution, and the shaded portions can be off-the-shelf components.

An overview of the forecasting process is shown in Figure 5. The pre-processing stage is our novel contribution. This involves setting the values for the parameters of the model. The technique must be fast, and it should give optimal or near-optimal values for the parameters.

Thus, we need to automatically set the values of the lag length L_{opt} and the number of nearest neighbors k_{opt}. Since we aim to use this method for forecasting a wide variety of time sequences, we should avoid any use of thresholding/bucketization techniques, which introduce arbitrariness into the system. Then, we shall have a black box, which, given a time series, will generate a similarity search system with all parameters set automatically.

4.1 Estimating the optimal lag length L_{opt}

Given a cloud of points in an L-dimensional space, there are fast methods of calculating the correlation fractal dimension, as mentioned in Section 3.3. Now, with a given time series, we can form lag plots for lags $L = 1 \ldots L_{max}$. Each lag plot is just a cloud of points in L-dimensional space, so its fractal dimension can be calculated fast. Let the fractal dimension of the lag plot for lag L be called f_L. Now we define the concept of a "Fractal Dimension vs Lag" (FDL) plot.

Symbol	Meaning
$\mathbf{x_t}$	Time series value at time t
N	Length of the time series x_t
$\mathbf{b}$	Lag vector
f	Fractal dimension of a cloud of points
L	Lag
k	Number of nearest neighbors
e	Error term
τ	Time delay
NMSE	Normalized Mean Squared Error

Table 1: Table of Symbols

Definition 5 (FDL plot of a sequence). *This is a plot of f_L versus L, for $L = 1 \ldots L_{max}$.*

Beyond a certain point, increasing the lag length adds no new information regarding the state space. This means that the intrinsic dimensions of the corresponding lag plots will remain the same. Since the FDL plot measures the intrinsic dimension for lag plots over different lags, the FDL plot should flatten out. The lag at which it settles could be used as the optimal lag length. For example, Figure 6(a) shows the FDL plot for a hypothetical dataset. We see that the plot starts flattening at lag $L = 4$. Thus, the optimal lag length in this case is $L_{opt} = 4$. Current algorithms for finding fractal dimension scale almost linearly with number of datapoints, so using this method is fast and scalable.

Generating the FDL plot requires a value of L_{max}. We generate this value adaptively. We find f_L for $L = 1, 2, \ldots$ till the variation in f_L for several successive values (w) of L lies within ϵ of their values. In our experiments, we used $w = 10$, and $\epsilon = \max(0.3, 10\%$of the moving average of the Fractal Di We found that the results are not sensitive to particular choices of these values. When the variation is within this range, it signals that the flat portion of the FDL plot has been reached, and we do not need to find f_L for higher L. The pseudo-code for this algorithm is as follows:

```
find_best_lag(X,Max){
  /* X is the time series, with individual
   * elements being x(t).
   */
  for(i = 1;; i++){
    V = The vector space with lag length of i;
```

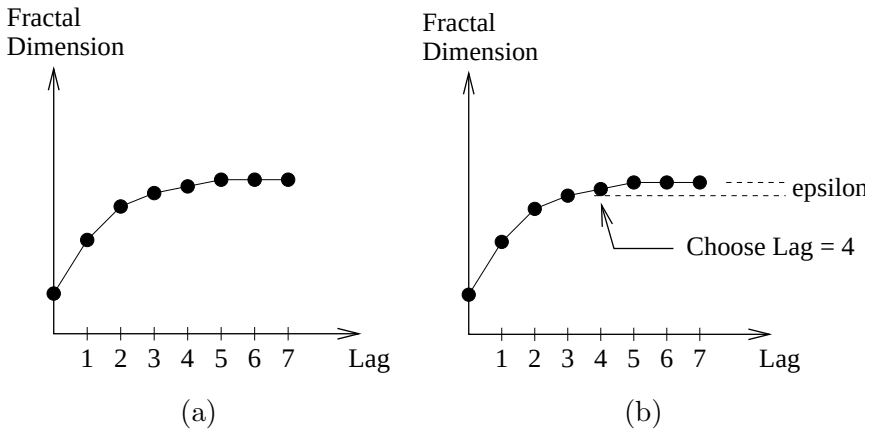


Figure 6: *FDL plots explained:* Given some time series, we can plot Fractal Dimension versus Lag to get a plot similar to (a). This is the FDL plot for the time series. We see that the plot flattens out with increasing lag, because higher lags do not add new information. The smallest lag within ϵ of the maximum fractal dimension is chosen as the optimal lag L_{opt}, as shown in Plot (b).

```
    fd[i] = fractal dimension of V;
    if we have reached the flat portion
        break;
  }
  L(opt) = minimum L such that fd[L] is within
           delta=95% of the maximum fd
  return L(opt);
}
```

We shall discuss the complexity of this algorithm later in Section 4.4, and show that it is fast. Also, this algorithm gives us information about the intrinsic dimensionality of the data, which may be useful for other forms of data mining.

4.2 Estimating the number of nearest neighbors k_{opt}

The delay-coordinate embedding approach requires finding k nearest neighbors to a given delay vector. Current implementations leave the value of k_{opt} to be fixed manually. The textbook approach is to have a dynamic assignment by using the technique of cross-validation. In this, we start by dividing the available data into a training and a holdout set. Using the optimal lag, we build an R-Tree using the training set. Then, starting with k as 1, we find the prediction

error over the holdout set. We keep incrementing k until the prediction error stops decreasing significantly.

We propose a new method for setting k_{opt} in this problem setting. Our observation is that the minimum number of points needed to bracket one point in f dimensions is $f + 1$. This leads us to the following conjecture.

Conjecture 1. *The optimal number of nearest neighbors should depend linearly on the intrinsic dimensionality; that is,*

$$k_{opt} = O(f) \tag{7}$$

To do a better interpolation, we need to bracket it in each dimension. This leads to $k = 2f$. We add a constant for handling noise, and that leads to the formula $k_{opt} = 2f + 1$. This is the technique we use. The results seem to suggest that this provides pretty good prediction accuracy (better than cross-validation, and it is also faster). It must still be mentioned that this is only a heuristic, which works well in practice.

4.3 Storage and Retrieval

Our system depends on making predictions on the basis of past regularities in the data. Thus, the data must be stored in a form making such similarity searches fast and efficient. All storage and retrieval operations are carried out only after the preprocessing step (refer Figure 5). Hence, the values for the parameters L_{opt} and k_{opt} are already known. The problem is to store the corresponding delay coordinate vectors in a manner that supports efficient k-nearest-neighbor search. Since the vectors to be stored are multi-dimensional (specifically, L_{opt}-dimensional), any Spatial Access Method can be used. We decided to use a data structure called an R-Tree. Guttman ([Gut84]) gives the details of the data structure and related algorithms.

Storage of and similarity searches over delay coordinate vectors can lead to the problem of dimensionality curse: the larger the dimension of the delay coordinate vector, the more acute the problem. This is tackled by the use of feature extraction, DFT ([FRM94]), DWT, segment means ([YF00]) and such methods.

4.4 Speed and Scalability

Here, we provide an analysis of the algorithms we have discussed. We show the order-of-magnitude computations. In Section 5, we shall show wall-clock times to confirm these.

Lemma 1 (Finding L_{opt}). *Time complexity of the step for finding L_{opt} is $O(NL_{opt}^2)$*

Proof. Given a cloud of N points in L-dimensional space, algorithms exist for calculating the fractal dimension in $O(NL*log(NL))$ time. But better algorithms can remove the $log(NL)$ factor, and what we observed in our experiments is that calculation of the fractal dimension of a cloud of points is indeed $O(NL)$ on the average. The preprocessing time of our algorithm is spent in generating the *FDL plot*, where fractal dimensions of clouds of points are calculated for dimensionality $L = 1 \ldots O(L_{opt})$. Thus, the total time taken is $O(N.1 + N.2 + \ldots + N.L_{opt}) = O(NL_{opt}^2)$. Thus the preprocessing is linear in the length of the time series and quadratic in the value of L_{opt}. □

Lemma 2 (Finding k_{opt}). *Time complexity of the step for finding k_{opt} is $O(1)$*

Proof. We have shown in Section 4.2 that our estimate of k_{opt} is:

$$k_{opt} = 2 * f_{L_{opt}} + 1$$

where $f_{L_{opt}}$ represents the fractal dimension of the L_{opt}-dimensional lag plot. The FDL plot used to compute L_{opt} already has this information. Hence, finding k_{opt} is an $O(1)$ operation. □

Lemma 3 (Preprocessing). *Time complexity of the preprocessing step is $O(NL_{opt}^2)$*

Proof. From Lemmas 1 and 2, we can see that the time complexity for preprocessing is $O(NL_{opt}^2) + O(1) = O(NL_{opt}^2)$. □

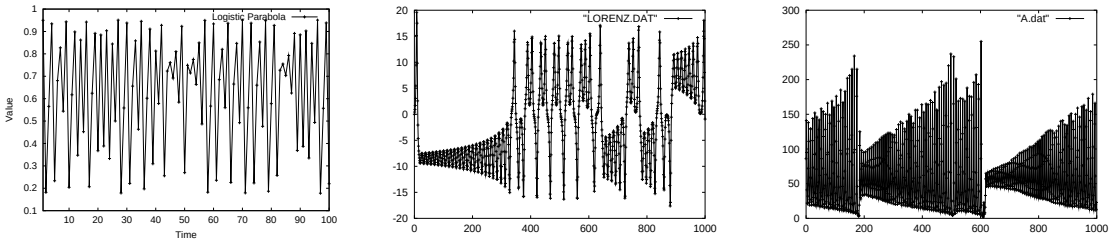
Thus, preprocessing time is shown to be linear on N, the size of the historical data. Forecasting the time series involves knn queries on R-Trees, and this subject has been treated elsewhere. The interested reader is referred to [FK94] for details.

4.5 Interpolation

A related question is determining the correct method for interpolating between the predictions given by the chosen k_{opt} nearest neighbors. We tried out several interpolation methods: one uses plain averaging over the k_{opt} predictions, another weights the predictions by the exponential of the negative of the distance squared (here, distance refers to the Euclidean distance between the current delay vector and the neighboring delay vector). We settled on an SVD-based interpolation method detailed in [Sau93].

Thus, the main aim of our work was to find formal methods for choosing values for lag length (L_{opt}) and number of nearest neighbors used for interpolation (k_{opt}). The success of the work can be judged by comparing the results produced by using these parameters against those when the parameters are chosen on an ad hoc basis. We have compared them on several datasets, the results of which will be described in the next section.

5 Results



(a) Logistic Parabola (b) LORENZ attractor (c) Laser fluctuations dataset

Figure 7: *The datasets*: These are samples of the datasets we use in our experiments.

We compared out algorithm to several parameter settings based on computation time required and prediction accuracy. We used several datasets, both real and synthetic, on which to test our algorithm. These were:

- Synthetic (but realistic):
  - *Logistic Parabola (LP)*: N = $3,000$ points, $x_t = 3.8x_{t-1}(1 - x_{t-1}) + e_t$, where e_t is Normal$(0, 0.001)$. This equation has been used to model the population of flies over time ([May76]).
  - *Lorenz equations (LORENZ)*: N = $20,000$ points. The equations are:

$$\dot{x} = \sigma(y - x) \tag{8}$$

$$\dot{y} = -xz + rx - y \tag{9}$$

$$\dot{z} = xy - bz \tag{10}$$

    where σ, b and r are constants. These equations have been used to model convection currents.

- Real-world:
  - *Laser Fluctuations (LASER)*: N = $10,000$ points, this is Time Series A from the Santa Fe competition(1992)

Figure 7 gives snapshots of these datasets. The following subsections will describe each of the datasets and the results we obtained on them.

For prediction accuracy, we used the typical measure called *Normalized Mean Squared Error* [AN93].

$$NMSE = \frac{1}{\sigma^2 N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2 \tag{11}$$

where x_i is the true value of the ith point in the series of length N, $\hat{x}_i$ is the predicted value, and σ is the standard deviation of the true time series during the prediction interval. The lower this value, the better the algorithm.

We did experiments to answer the following questions:

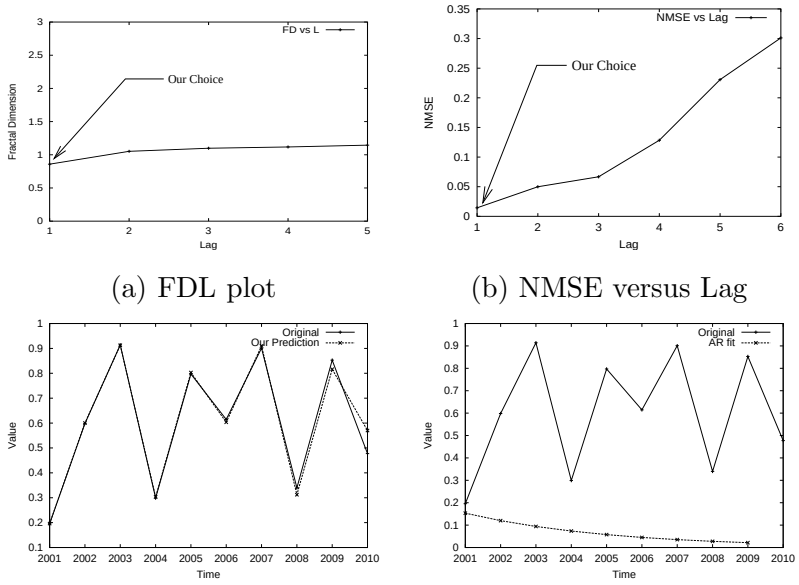
- how good (that is, accurate) are the proposed choices of L_{opt} and k_{opt}
- how fast is the preprocessing
- how fast is the forecasting

5.1 Accuracy

For each dataset, we show the FDL plot and the choice of L_{opt}. This choice is then compared to other possible values of L in the form of an NMSE versus Lag plot. We show that our choice of L_{opt} is either optimal or near-optimal. For each point on this plot, we do a 10–step prediction which is repeated 50 times. The NMSE plotted is the median NMSE from the repeated experiments. The median was chosen as opposed to the mean because it is less susceptible to outliers. We also give a snapshot of prediction using our method, as well as prediction over the same sample using the AR model. These plots confirm the assertion that our method of non-linear forecasting performs far better than the AR method.

Logistic Parabola (LP) dataset We can see from the equation of the system that the present value of the time series is sufficient to predict the next value. Thus, a lag length of 1 is enough for prediction purposes. In fact, if we increase the lag length, then no new information will be added but more noise gets introduced. So, the optimal lag length is one, and this is the value that our algorithm should return. Thus, this dataset can used for performing a "sanity check" on our algorithm.

The results for this dataset are shown in Fig 8. The plot of Fractal Dimension versus Lag Length shows that the Fractal Dimension does not significantly increase as lag length is incremented. This implies that a lag length of one is sufficient to reconstruct the state space. This is indeed what we would expect from the equation, because x_t depends only on x_{t-1}. Thus, in this case, our algorithm gives the correct intuitive result. A prediction snapshot is

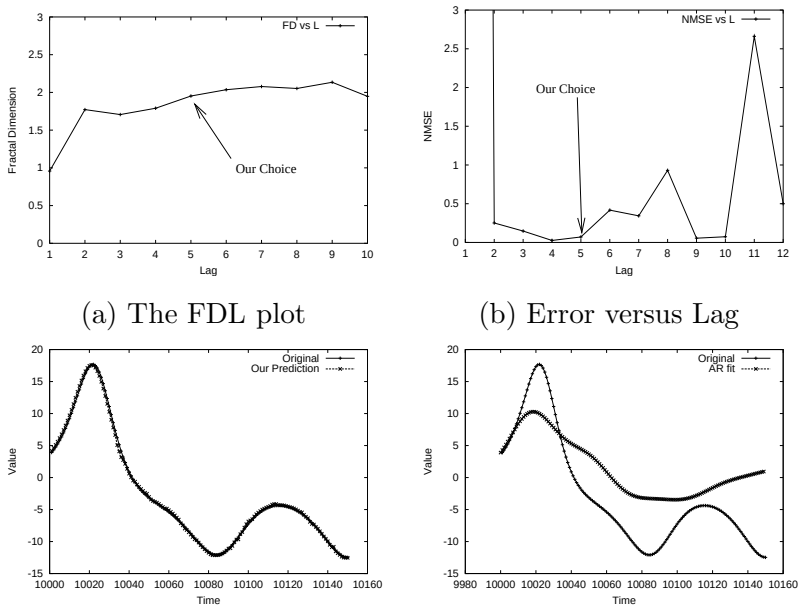


(a) FDL plot (b) NMSE versus Lag

(c) 10-step prediction (d) The AR(1) fit

Figure 8: *Logistic Parabola dataset*: Plot (a) is the FDL plot, suggesting a lag length of $L_{opt} = 1$, because further increase in lag length does not appreciably increase the fractal dimension. Plot (b) shows that our chosen $L_{opt} = 1$ actually gives optimal NMSE. Plot (c) shows a specific case of a 10-step prediction, and plot (d) shows the AR(1) prediction over the same sample.

also shown, in which the system predicts data from the range of timestep 2001 to timestep 2010, based on data seen from timestep 1 to 2000. We see that the prediction accuracy is excellent. The AR prediction is far worse, as expected.



(a) The FDL plot (b) Error versus Lag

(c) Our 150-step prediction (d) AR fit

Figure 9: *The LORENZ dataset*: (a)Fractal Dimension vs Lag plot, which suggests a lag length of 5; (b) NMSE vs Lag plot, which reaches its minimum at lag of 4; (c) sample prediction sequence, and (d) the Autoregressive prediction for the same sample. Our prediction is seen to perform far better than the AR(5) model.

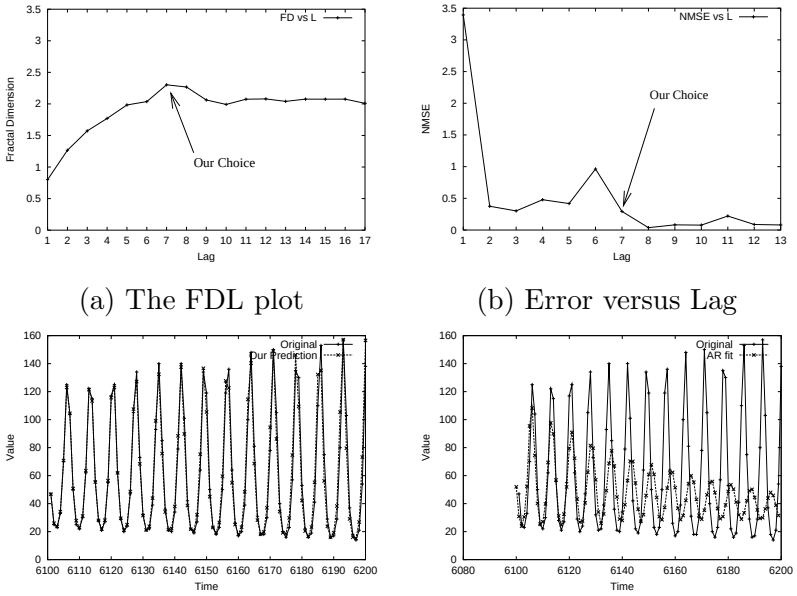
LORENZ dataset The LORENZ dataset is a synthetic dataset which models convection currents. Results from the LORENZ dataset are shown in Figure 9. The observations are:

1. The plot is seen to flatten out at a lag length of five. This verifies Takens' theorem, and the mathematical justification for our technique.

2. NMSE was also computed for each lag length, and we see that plot 9(b) has its minimum at lag length of four. This is close to the value that our approach gives. The difference in NMSEs for these two lags is also not much different.

3. We also show a sample 150-step-ahead prediction sequence. The prediction accuracy is excellent.

4. We ran AR with the same window size as out L_{opt}, that is, 5. Our method is seen to clearly outperform the AR(5) model.



(a) The FDL plot (b) Error versus Lag

(c) 100-step prediction (d) The AR fit

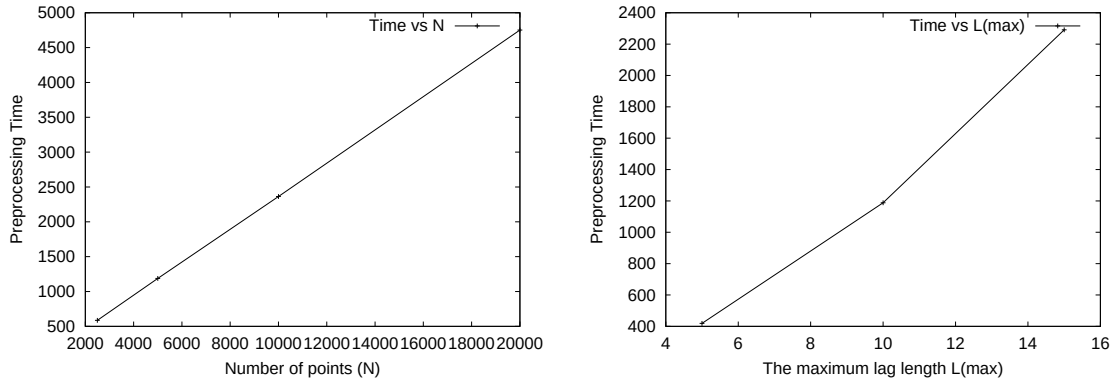
Figure 10: *The LASER dataset*: (a)Fractal Dimension vs Lag plot, which suggests a lag length of 7; (b)Error vs Lag plot, with the NMSE being low at lag 7 and continuing to decrease for higher lags; (c) Sample 100-step prediction sequence, and (d) the AR(7) fit over the same sample. Our method clearly outperforms the AR(7) model.

LASER dataset This is a dataset of fluctuations measured during the operation of a Laser. It is more widely known as "Data Set A" from the Santa Fe competition [AN93]. We used the first 6000 points as the training set. Our results on a particular stretch of the dataset are shown in Figure 10. The observations are:

1. The FDL plot flattens out at around $L = 7$, which is chosen as L_{opt}.

2. The NMSE versus Lag plot shows that this choice is one of the smallest values of L which give good accuracy.

3. We also show a sample 100-step-ahead prediction sequence, where we predict using the AR model, using a window size of $7 (= L_{opt})$. Our prediction is clearly better.



(a) (b)

Figure 11: *Preprocessing Time*: (a) This is the plot for preprocessing time given a fixed value of $L_{max} = 10$. The preprocessing time is seen to be linear in the length of the time series. (b) This is the plot where the length of the time series is kept fixed at $N = 5000$. The time taken varies with the square of the value of L_{max}.

5.2 Preprocessing Time

For the experiment, we generated the LORENZ dataset for different lengths of time. Figure 11 gives the experimental results. It can be seen that the preprocessing time varies almost linearly with the size of the training set N, and quadratically with L. This verifies the assertions in Section 4.4.

5.3 Forecasting Time

The same experimental setting as before is used in this. Figure 12 gives the results when forecasting time is compared with database size (that is, the length of the time series). It can be seen that the system easily outperforms plain Sequential Search. The observations to note are:

- F4 takes approximately constant time (about 1.5 seconds), which is interactive.
- F4 does not suffer from dimensionality curse problems, thanks to our careful design (using segments means and R-Trees).

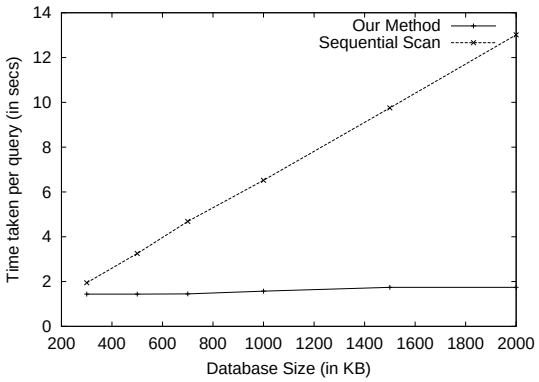


Figure 12: *Forecasting Time*: LORENZ forecasting time versus the database size. Our method is found to clearly outperform sequential scan, without suffering from dimensionality curse problems.

6 Conclusions

We focussed on building a fast and completely automated non-linear forecasting system. The major contributions of this work are the following:

- The automatic estimation of vital forecasting parameters, namely of the optimal lag L_{opt} and a good guess for the number of neighbors k_{opt}. The novelty is the use of 'fractal dimensions' for this purpose.
- Scalability: 'F4' is the first scalable, 'black-box' forecasting method, capable of handling arbitrarily large datasets, and with fast forecasting algorithms.
- Experiments on real and synthetic datasets, that show that our method is not only fast and scalable, but also accurate, achieving low prediction error.

Additional, smaller contributions include the following:

- The modularization of the forecasting problem: the 'F4' system (Figure 5) can trivially use better/newer access methods, as they come along, although the chosen R-tree and 'segment means' perform very well. It can also trivially accommodate newer/better interpolation methods, once the k nearest neighbors have been retrieved.
- The introduction of lag-plots to the data-mining field. Papers on non-linear forecasting often require a strong mathematical background. In this paper, we have kept only the most fundamental concepts (such as lag-plots), and we tried to give the intuition

behind them, as well as their relationship to database and data-mining concepts (R-trees, nearest-neighbor queries).

Future research could focus on extending 'F4' to do forecasting on multiple, co-evolving time sequences, that exhibit linear or non-linear correlations. A more ambitious direction would be to automatically recover the (non)linear equations that govern the time sequence at hand.

Appendix

A Selecting Time Delay (τ)

The embedding theorem [Tak81] does not specify any value for the time delay τ, except certain values, as long as there is infinite amount of data. But since this is never true, we need to find some workable value for the time delay. Abarbanel ([Aba95]) suggests certain principles for choosing this value:

- It must be some multiple of the sampling rate, so that the values of the observable at those time instants are available directly from our set of measurements. If this was not true, we would have to interpolate from observed values at different time instants, which may be a noisy process.
- If the time delay is too short, the system would not have explored enough of its state space to produce, in a numerical sense, new information about that state space. We would expect, from a physical standpoint, that the measurements x_t and $x_{t-\tau}$ to be essentially the same measurement. The system's dynamics would not have been explored.
- If the time delay is too large, the chaos in the system would effectively make x_t and $x_{t-\tau}$ look random. Even a small measurement error, which is inevitable, will blow up exponentially over time.

Hence, an intermediate value of τ is needed. Fraser ([FS86, Fra89]) suggested the use of Mutual Information to set the time delay. Mutual information in this setting is defined as

$$I(T) = \sum_{x_t, x_{t-T}} P(x_t, x_{t-T}) \log_2 [\frac{P(x_t, x_{t-T})}{P(x_t)P(x_{t-T})}] \tag{12}$$

Here, $P(x_t, x_{t-T})$ is the probability of observing x_t given that the previous observation was x_{t-T}. The entire range of x_t is divided into a number of buckets, and probabilities are calculated for these buckets. Thus $P(x_t)$ refers to the probability of an observation falling in the bucket corresponding to x_t, and $P(xt, x_{t-T})$ is the probability of consecutive observations

from the corresponding buckets. $I(T)$ represents the amount of information in bits which the value of x_t provides about x_{t-T}. The prescription is to take the time delay τ to be the first minimum in the plot of $I(T)$ versus T. Thus, we make sure that x_t and $x_{t-\tau}$ are not totally correlated, and yet not so far apart as to be random.

B R-Trees

R-Trees belong to a group of algorithms called *Spatial Access Methods*. These allow fast storage and retrieval of multidimensional data. R-Trees have been one of the most popular Spatial Access Methods. A lot of work has been done on them since they were developed by Guttman in 1984 ([Gut84]). Here, we shall only provide a brief overview of the concepts.

Data structures such as B-Trees have long been used for one-dimensional data. Initial attempts to store multi-dimensional data focussed on modifying B-Trees. One such approach was that of *k-d-B-Trees* ([Rob81]). The problem was that maintenance of the data structure became very hard, without any space utilization guarantees. R-Trees made a fundamental break from B-Trees in that overlap between nodes was now allowed. Each leaf in an R-Tree corresponds to some region in the multidimensional space. Each non-leaf node corresponds to a *Minimum Bounding Rectangle (MBR)*, which fits tightly around all the MBRs in its subtree. MBRs are allowed to share regions. This technique gives guaranteed 50% utilization, and the insertion and splitting algorithms also become easy.

References

[Aba95] H. D. I. Abarbanel. *Analysis of Observed Chaotic Data*. Springer, 1995.

[Aka70] H. Akaike. Statistical predictor identification. *Annals of the Institute of Statistical Mathematics*, 22:203–217, 1970.

[AN93] A.S.Weigend and N.A.Gershenfeld, editors. *Time Series Prediction: Forecasting the Future and Understanding the Past*. Addison Wesley, 1993.

[BF95] Alberto Belussi and Christos Faloutsos. Estimating the selectivity of spatial queries using the 'correlation' fractal dimension. In Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio, editors, *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland*, pages 299–310. Morgan Kaufmann, 1995.

[BGS01] Philippe Bonnet, J. E. Gehrke, and Praveen Sheshadri. Towards sensor database systems. In *Proceedings of the Second International Conference on Mobile Data Management, Hong Kong, January 2001*, 2001.

[BJ76] G.E.P. Box and F.M. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, Oakland, CA, second edition, 1976.

[BW01] S. Babu and J. Widom. Continuous queries over data streams. *Sigmod Record, September 2001*, 2001.

[Cha89] C. Chatfield. *The Analysis of Time Series*. London: Chapman and Hall, 1989.

[CM00] Kaushik Chakrabarti and Sharad Mehrotra. Local dimensionality reduction: A new approach to indexing high dimensional spaces. In *Proceedings of the 26th VLDB Conference*, Cairo, Egypt, 2000.

[DLM+98] Gautam Das, King-Ip Lin, Heikki Mannila, Gopal Renganathan, and Padhraic Smyth. Rule discovery from time series. In *Proceedings of the 3rd International Confrerence on Knowledge Discovery and Data Mining*, 1998.

[FBC95] F. Fessant, S. Bengio, and D. Collobert. On the prediction of solar activity using different neural network models. *Annales Geophysicae*, 1995.

[FD93] A. M. Fraser and A. Dimitriadis. *Forecasting Probability Densities by Using Hidden Markov Models with Mixed States*. 1993.

[FK94] Christos Faloutsos and Ibrahim Kamel. Beyond uniformity and independence: Analysis of R- trees using the concept of fractal dimension. In *Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 4–13, Minneapolis, 1994.

[Fra89] A. M. Fraser. *Information Theory and Strange Attractors*. PhD thesis, University of Texas, Austin, 1989.

[FRM94] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. *SIGMOD*, 1994.

[FS86] A. M. Fraser and H. L. Swinney. Independent coordinates for strange attractors with mutual information. *Phys. Rev. A*, 33:1134–1140, 1986.

[GD01] D. Gunopoulos and G. Das. Time series similarity search measures and time series indexing. In *Proceedings of the ACM SIGMOD*, page 624, Santa Barbara, 2001.

[GKS01] J. E. Gehrke, Flip Korn, and Divesh Srivastava. On computing correlated aggregates over continual data streams. In *SIGMOD 2001, Proceedings of the 2001 ACM Sigmod International Conference on Management of Data, Santa Barabara, California, May 2001*, 2001.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. *ACM SIGMOD*, pages 47–57, 1984.

[KCMP01] Eamonn Keogh, Kaushik Chakrabarti, Sharad Mehrotra, and Michael Pazzani. Locally adaptive dimensionality rediction for indexing large time series databases. In *ACM SIGMOD*, Santa Barbara, California, USA, May 2001.

[KJF97] Flip Korn, H. Jagadish, and Christos Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. In *Proceedings of SIGMOD*, Tucson, Arizona, USA, 1997.

[KL93] E. J. Kostelich and D. P. Lathrop. *Time Series Prediction by Using the Method of Analogues*. 1993.

[LF87] A. Lapedes and R. Farber. Nonlinear signal processing using neural networks. Technical Report LA-UR-87-2662, Los Alamos National Laboratory, Los Alamos, NM, 1987.

[LH88] K. Lang and G. Hilton. A time-delay neural network architecture for speech recognition. Technical Report CMU-CS-88-152, Carnegie Mellon University, Pittsburgh, PA, 1988.

[May76] Robert M. May. Simple mathematical models with very complicated dynamics. *Nature*, 261:459, 1976.

[Moz93] Michael C. Mozer. *Neural Network Architectures for Temporal Sequence Processing*, pages 243–264. Addison Wesley, 1993.

[PJS92] Heinz-Otto Peitgen, Hartmut Jurgens, and Dietmar Saupe. *Chaos and Fractals: New Frontiers of Science*. Springer-Verlag, 1992.

[RDGC93] D.E. Rumelhart, R. Durbin, R. Golden, and Y. Chauvin. *Backpropagation: The Basic Theory*. Lawrence Erlbaum, Hillsdale, NJ, 1993.

[Rob81] J. T. Robinson. The k-d-b-tree: A search structure for large multidimensional dynamic indexes. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 10–18, Ann Arbor, Michigan, 1981.

[Sau93] T. Sauer. *Time Series Prediction by Using Delay Coordinate Embedding(Data Set A)*, pages 175–193. Addison Wesley, 1993.

[ST99] Betty Salzberg and Vassilis J. Tsotras. Comparison of access methods for time-evolving data. *ACM Computing Surveys*, 31(2):158–221, 1999.

[Tak81] F. Takens. Detecting strange attractors in turbulence. In D. A. Rand and L.-S. Young, editors, *Dynamical Systems and Turbulence; Lecture Notes in Mathematics*, volume 898, pages 366–381. Berlin: Springer-Verlag, 1981.

[Wai89] A. Waibel. Modular construction of time-delay neural networks for speech recognition. *Neur. Comp.*, 1(1):39–46, 1989.

[Wan93] E.A. Wan. *Time Series Prediction by Using a Connectionist Network with Internal Delay Line*, pages 195–217. Addison Wesley, 1993.

[Wer74] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioural Sciences*. PhD thesis, Harvard University, Cambridge, MA, 1974.

[Wer88] P. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neur. Net.*, 1:339–356, 1988.

[WHR90] A. S. Weigend, B. A. Huberman, and D. E. Rumelhart. Predicting the future: A connectionist approach. *International Journal of Neural Systems*, 1:193–209, 1990.

[YF00] Byoung-Kee Yi and Christos Faloutsos. Fast time sequence indexing for arbitrary lp norms. In Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter, and Kyu-Young Whang, editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10–14, 2000, Cairo, Egypt*, pages 385–394. Morgan Kaufmann, 2000.