F4: Large-Scale Automated Forecasting Using Fractals

Deepayan Chakrabarti
Center for Automated Learning and Discovery
School of Computer Science
Carnegie Mellon University
deepay@cs.cmu.edu

Christos Faloutsos
Computer Science Department
School of Computer Science
Carnegie Mellon University
christos@cs.cmu.edu

ABSTRACT

Forecasting has attracted a lot of research interest, with very successful methods for periodic time series. Here, we propose a fast, automated method to do non-linear forecasting, for both periodic as well as *chaotic* time series. 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.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data Mining—*time series forecasting*

General Terms

Algorithms

Keywords

Time series, Automated forecasting, Fractals

1. INTRODUCTION

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. This has applicability in a wide range of fields, such as financial systems (exchange rates), physics (laser fluctuations) and biology (physiological data over time). Many techniques have been suggested, but are either too simplistic for many real-world problems (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.

Our method, being completely automated, can operate in a sensor network setting, which is an area of increasing interest ([12], [5], [3]). Each sensor will be collecting time series 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.

Problem Definition (Predict-1):

- **Given** a time series $x_1, x_2, \ldots, x_t$,
- **predict** the value of x_{t+1}.

Later in the paper, we shall generalize the problem to allow multiple time series and n-step-ahead predictions. An example time series is shown in Figure 1(a).

The rest of the paper is organized as follows: Section 2 gives a background tutorial on certain techniques used by our algorithms. Section 3 details our contributions, and gives the algorithms we use. Section 4 gives our experiments and results. Section 5 describes related work in the field of time series prediction, followed by the conclusions in Section 6.

2. BACKGROUND

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:

$$\mathbf{b} = [\mathbf{x_t}, \mathbf{x_{t-\tau}}, \mathbf{x_{t-2\tau}}, \ldots, \mathbf{x_{t-L\tau}}] \qquad (1)$$

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.*

Definition 2. *Lag Plot: A plot of this $(L+1)$ dimensional vector space is called the* **Lag Plot** *for lag L. This vector space is also called the Phase-Space.*

We explain the technique we use with an example. Figure 1(a) shows a plot of 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 1(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, and interpolate between them. The result is our predicted value of x_{t+1}. This idea can be generalized to lag plots with $L > 1$, with KNN being done on the L dimensions representing $x_{t-\tau}, \ldots, x_{t-L\tau}$, followed by interpolation on the dimension representing x_t. As against this, we show the fit obtained by using an autoregressive (AR) model in figure 1(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.

This soundness of this method is based on a certain result (generally known as Takens' Theorem), which is outside the scope of this paper. It says that there is an optimal value of L, and increasing L beyond that value will not yield any better predictions. Interested readers are referred to [27, 26] for details.

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

The value of the time delay (τ) is usually $\tau = 1$. For setting the correct lag length L_{opt}, most current implementations either involve manual parameter setting or techniques requiring thresholding/bucketization. This is one of the issues for which we propose solutions. Again, the determination of k_{opt} has till now been ad hoc. Here we propose a method to automatically set the value of k_{opt}.

FalseNearestNeighbors

The problem is finding the optimal value for lag. Abarbanel([1]) suggests a method called *False Nearest Neighbors* to estimate this. 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 upcoming methods solve this problem of choosing L.

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 1(b)) also has an intrinsic dimensionality of 1.

Symbol	Meaning
$\mathbf{x_t}$	Time series value at time 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

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([4]). Figure 2 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 [21] for more details.

3. PROPOSED METHOD

We recap the problem: **Given** a time series $x_1, x_2, \ldots, x_t$, **predict** the value of x_{t+1}. We can generalize this problem as follows:

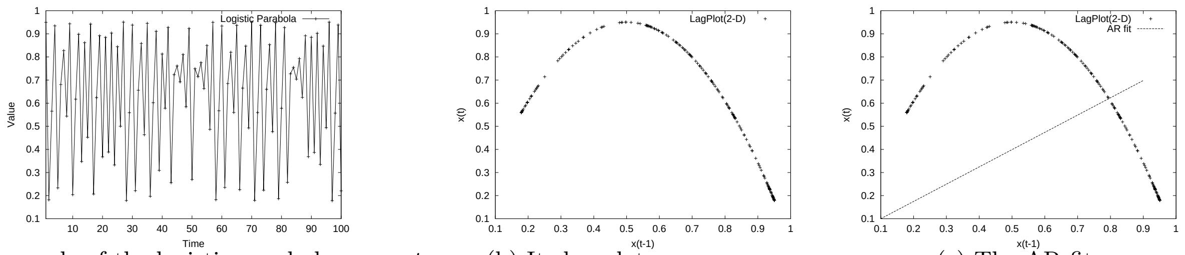
Problem Definition (Predict-n): The training set (also called the historical data, for example, stocks) is a set of time series generated by the *same* physical system over separate periods of time.

$$TS = X_1, \ldots, X_N$$

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.

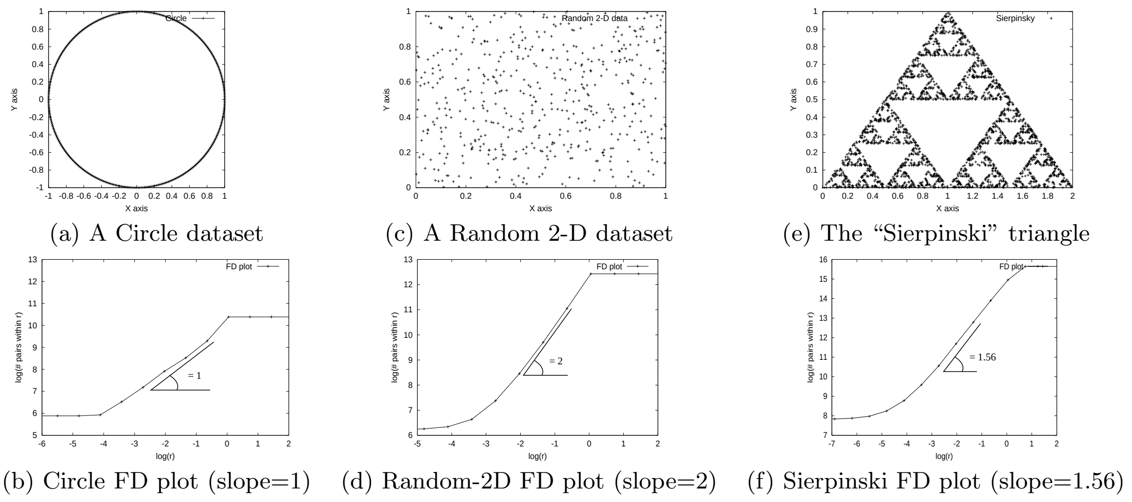
If we had only one training sequence, which was also the query sequence, then this would reduce to the *Predict-1* problem.

We automate the delay-coordinate-embedding approach in our forecasting system. An advantage of this approach is its firm mathematical basis [27, 26]. To automate the forecasting process, the system needs to be able to automatically



(a) A sample of the logistic parabola: x_t vs t (b) Its lag-plot: x_{t+1} vs x_t (c) The AR fit

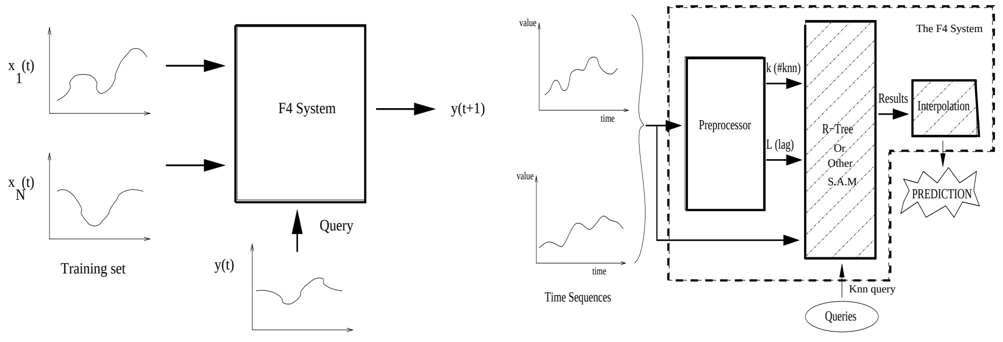
Figure 1: *The logistic parabola*: The equation generating this dataset is: $x_t = 3.8x_{t-1}(1 - x_{t-1}) + e_t$, where e_t is a $\mathbf{N}(0, 0.001)$ random variable. This is a chaotic dataset, a sample of which is shown in (a). This appears hard to predict, but on generating the two-dimensional lag-plot, 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.



(a) A Circle dataset (c) A Random 2-D dataset (e) The "Sierpinski" triangle

(b) Circle FD plot (slope=1) (d) Random-2D FD plot (slope=2) (f) Sierpinski FD plot (slope=1.56)

Figure 2: *Examples of Fractal Dimension (FD) plots*: The circle of plot (a) is a smooth curve; its fractal dimension is found to be 1 from the slope of plot (b). Plot (c) shows a dataset with points spread randomly in 2 dimensions. The slope from plot (d) is close to 2 as expected. Plot (e) shows a self-similar cloud of points; its fractal dimension is approximately 1.56 from plot (f).



(a) (b)

Figure 3: *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 series. 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.

set the values of the parameters, such as the optimal lag length L_{opt} and optimal number of nearest neighbors k_{opt}. The thrust of our research has been on formalizing methods for setting these parameters, without the use of thresholding/bucketization techniques, which introduce arbitrariness into the system. Thus, we shall have a black box, which, given a time series, will generate a similarity search system based on this approach, with all parameters set automatically.

3.1 Estimating the optimal lag length L_{opt}

Given a cloud of points in an L-dimensional space, let us call its fractal dimension f_L. Now, with a given time series, we can form several clouds of points by considering $L = 1 \dots L_{max}$. Now we define the concept of a "Fractal Dimension vs Lag" (FDL) plot.

DEFINITION 3. *FDL plot of a sequence: This is a plot of f_L versus L, for $L = 1 \dots L_{max}$.*

Beyond a certain point, increasing the lag length adds no new information regarding the state space, so 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 the LASER dataset. We see that the plot starts flattening at lag $L = 7$. Thus, the optimal lag length in this case is $L_{opt} = 7$. 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 $L = 1, 2, \dots$ 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 Dimension). 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;
    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);
}
```

One possible method for estimating L_{opt} would be by using Cross-Validation. In this case, the historical data is divided into a training set and a holdout set, and for each $L = 1, 2, \dots, L_{max}$, the error is estimated over the holdout set. The value of L at which this error is minimized is chosen to be L_{opt}. But this approach requires us to do k-nearest-neighbors for each value of L, which means that an R-Tree has to be built for each value. This is an extremely time-consuming operation, and cannot be done for large datasets. Hence, this technique cannot be used to find L_{opt}. Our approach is more scalable and can be extended to very large datasets.

3.2 Estimating the Number of Nearest Neighbors k_{opt}

Our approach requires finding k nearest neighbors to a given delay vector. Current implementations leave the value of k 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,*

$$k_{opt} = O(f) \quad (2)$$

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.

3.3 Speed and Scalability

Finally, this 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. Storage of and similarity searches over delay coordinate vectors leads to several problems. One is the problem of dimensionality curse: the larger the dimension of the delay coordinate vector, the more acute the problem. Another problem is that of fast retrieval from the database. We have already implemented a time series similarity search prototype, which uses the R-Tree spatial data access method ([14]). The dimensionality curse is tackled by the use of feature extraction, DFT ([9]), DWT, segmented means ([32, 16]) and such methods.

The other point of concern is the speed and scalability of the preprocessing step. We show in Section 4.2 that this step is indeed fast. We give order-of-magnitude computations, as well as experimental wall-clock times to prove this.

LEMMA 1. *Time complexity of the preprocessing step is $O(NL_{opt}^2)$*

PROOF 1. *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}.*

3.4 Interpolation

A related question is determining the correct method for interpolating between the predictions given by the chosen k nearest neighbors. We tried out several interpolation methods: one uses plain averaging over the k 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 [25], where the least-squares fitting line through the nearest neighbor predictions is used as an approximation of the true predictor function. The projection of the point-of-prediction on to this line gives the value of the prediction.

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.

4. RESULTS

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):
  - *Lorenz equations (LORENZ)*: N = 20,000 points. The equations are:

$$\dot{x} = \sigma(y - x) \quad (3)$$
$$\dot{y} = -xz + rx - y \quad (4)$$
$$\dot{z} = xy - bz \quad (5)$$

    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 4 gives snapshots of these datasets. Several other datasets were also tested, but are not detailed due to space considerations; results in all cases were similar. 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* [2].

$$NMSE = \frac{1}{\sigma^2 N} \sum_{i=1}^{N} (x_i - \hat{x}_i)^2 \quad (6)$$

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

4.1 Accuracy

LORENZ dataset

The LORENZ dataset is a synthetic dataset which models convection currents. Results from the LORENZ dataset are shown in Figure 5. 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 5(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.

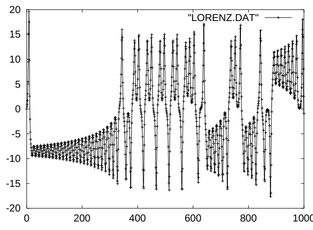
LASER dataset

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 [2]. We used the first 6000 points as the training set. Our results on a particular stretch of the dataset are shown in Figure 6. The observations are:

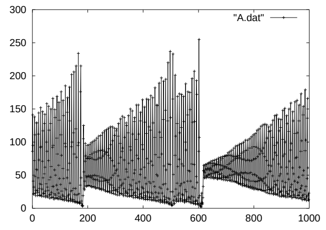
1. The FDL plot flattens out at around $L = 7$, which is chosen as L_{opt}.
2. The NMSE versus Lag plot shows that this choice is one of the smallest values of L which give good accuracy.
3. We also show a sample 100-step-ahead prediction sequence, where we predict using the AR model, using a window size of $7 (= L_{opt})$. Our prediction is clearly better.

4.2 Preprocessing Time

For the experiment, we generated the LORENZ dataset for different lengths of time. Figure 7 (plots a and b) give 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 3.3.

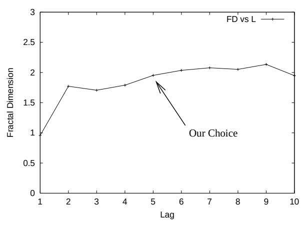


(a) LORENZ attractor

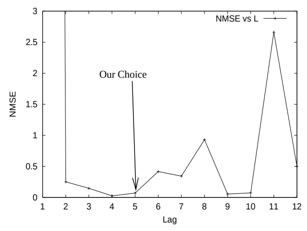


(b) Laser fluctuations dataset

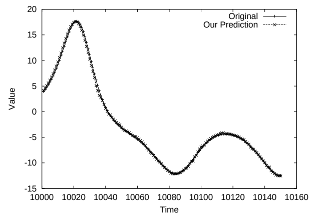
Figure 4: *The datasets*: These are samples of the datasets we use in our experiments.



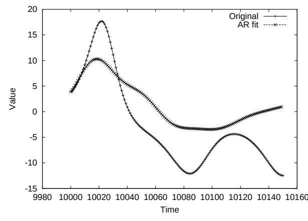
(a) The FDL plot



(b) Error versus Lag

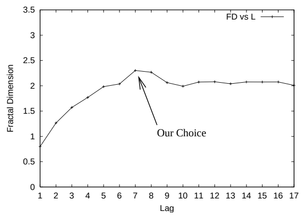


(c) Our 150-step prediction

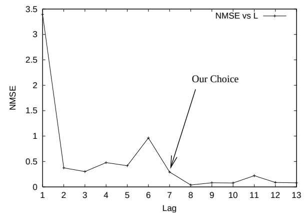


(d) AR fit

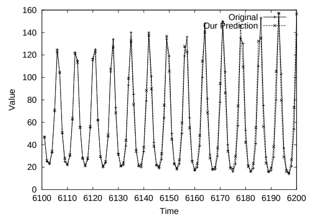
Figure 5: *The LORENZ dataset*: (a)Fractal Dimension vs Lag plot, which suggests a lag length of 5; (b) NMSE vs Lag plot, which reaches its minimum at lag of 4; (c) sample prediction sequence, and (d) the Autoregressive prediction for the same sample. Our prediction is seen to perform far better than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



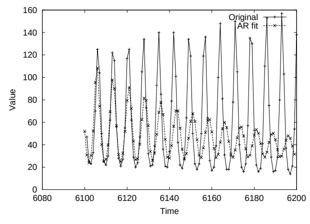
(a) The FDL plot



(b) Error versus Lag

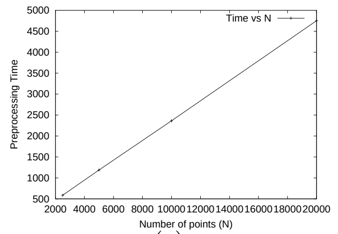


(c) 100-step prediction

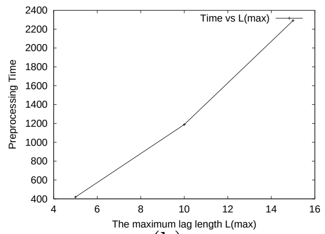


(d) The AR fit

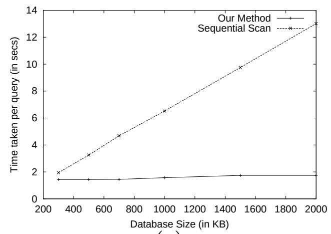
Figure 6: *The LASER dataset*: (a)Fractal Dimension vs Lag plot, which suggests a lag length of 7; (b)Error vs Lag plot, with the NMSE being low at lag 7 and continuing to decrease for higher lags; (c) Sample 100-step prediction sequence, and (d) the AR(7) fit over the same sample. Our method clearly outperforms the AR(7) model.



(a)



(b)



(c)

Figure 7: *Preprocessing and Forecasting Time*: (a) This is the plot for preprocessing time versus N 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 preprocessing time taken varies with the square of the value of L_{max}. (c) Here we plot forecasting time versus database size. Our method is found to clearly outperform sequential scan, without suffering from dimensionality curse problems.

4.3 Forecasting Time

The same experimental setting as before is used in this. Figure 7 (plot c) 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 segmented means and R-Trees).

5. RELATED WORK

Work on time series can be broadly classified into three categories: Linear Prediction, Non-linear Prediction and Time-Series-Indexing related work.

Linear Prediction

There exist a large number of linear models, such as AR, MA, ARIMA, ARFIMA, and so on. The basic ideas are illustrated here in the case of the ARMA(M,N) model, where the equation for x_t is given by:

$$x_t = \sum_{m=1}^{M} a_m x_{t-m} + \sum_{n=0}^{N} b_n e_{t-n} \qquad (7)$$

where the a_ms lead to internal dynamics, and the b_ns are weighing factors for external inputs e_t. It has been widely used in all areas of time series analysis and discrete-time signal processing. Reference texts for this are [7, 22]. The downside of linear models is that they fail in the presence of non-linearities in the underlying process. We have provided an intuitive reason for this with regard to the Logistic Parabola dataset (figure 1(c)).

Non-linear Prediction

One nonlinear prediction method is the *Delay Coordinate Embedding* method, which has been described in Section 2. Another approach has been through the use of Artificial Neural Networks [30, 31, 19, 29, 28]. Neural networks use the same idea: find a function f which gives the best fit for $x_t = f(x_{t-1}, \ldots, f_{t-w})$. The only difference is in the method for estimating the function f. A taxonomy of neural network structures for time-series processing is provided in [20]. But all these methods suffer from the traditional problems associated with neural networks: long training times and large number of parameters.

Hidden Markov Models(HMMs) ([23]) have also been used ([10, 11]). There is a similarity between these models and our technique, but the forward-backward algorithm for finding the parameters in such models is $O(N^2)$ and hence not scalable to large datasets. Also our approach seems to lead to a more natural problem description.

Another method is called the Method of Analogues ([18]). 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.

Other Time Series Work

Time sequences have been studied from several points of view, other than prediction. [13] and [24] are some recent surveys on time series similarity and indexing. One problem of particular relevance to the data-mining and database communities 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 n-dimensional data, spatial databases such as R-Trees([14]) and variants have been used. But such structures cannot efficiently handle very high-dimensional data. Hence, techniques have been sought to reduce dimensionality: [17] used Singular Value Decomposition (SVD) to do this. In [32], the authors use an approach called "Segmented Means" in which the time series 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. A similar technique is detailed in [16]. In [6], the authors 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, [15] 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 by [8] 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.

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 3) can trivially use better/newer access methods, as they come along, although the chosen R-tree and 'segmented 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 series, that exhibit linear or non-linear correlations. A more ambitious direction would be to automatically recover the (non)linear equations that govern the time series at hand.

7. REFERENCES

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

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

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

[4] A. Belussi and C. Faloutsos. Estimating the selectivity of spatial queries using the 'correlation' fractal dimension. In U. Dayal, P. M. D. Gray, and S. 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.

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

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

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

[8] G. Das, K.-I. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule discovery from time series. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, 1998.

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

[10] A. M. Fraser and A. Dimitriadis. Forecasting probability densities by using hidden markov models with mixed states. *Time Series Prediction: Forecasting the Future and Understanding the Past*, 1993.

[11] X. Ge and P. Smyth. Deformable markov model templates for time-series pattern matching. In *Knowledge Discovery and Data Mining*, pages 81–90, 2000.

[12] J. E. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continual data streams. In *Proceedings of the 2001 ACM Sigmod International Conference on Management of Data, Santa Barbara, California*, May 2001.

[13] 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.

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

[15] E. Keogh, K. Chakrabarti, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *ACM SIGMOD*, Santa Barbara, California, USA, May 2001.

[16] E. J. Keogh and M. J. Pazzani. A simple dimensionality reduction technique for fast similarity search in large time series databases. In T. Terano, H. Liu, and A. Chen, editors, *Knowledge Discovery and Data Mining, Current Issues and New Applications, 4th Pacific-Asia Conference, PAKDD*, volume 1805, pages 122–133, Kyoto, Japan, 2000. Springer.

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

[18] E. J. Kostelich and D. P. Lathrop. Time series prediction by using the method of analogues. *Time Series Prediction: Forecasting the Future and Understanding the Past*, 1993.

[19] 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.

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

[21] H.-O. Peitgen, H. Jurgens, and D. Saupe. *Chaos and Fractals: New Frontiers of Science*. Springer-Verlag, 1992.

[22] M. Priestley. *Spectral Analysis and Time Series*. London: Academic Press, 1981.

[23] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *IEEE Proc.*, 77(2):257–284, 1989.

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

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

[26] T. Sauer, J. A. Yorke, and M. Casdagli. Embedology. *J. Stat. Phys.*, 65(3/4):579–616, 1991.

[27] 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.

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

[29] 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.

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

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

[32] B.-K. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary lp norms. In *26th VLDB, Cairo, Egypt*, pages 385–394. Morgan Kaufmann, 2000.