Time Series Forecasting from High-Dimensional Data with Multiple Adaptive Layers

R. Bharat Rao, Scott Rickard, and Frans Coetzee
Adaptive Information & Signal Processing Department
Siemens Corporate Research, Inc.
755 College Road East
Princeton, NJ 08540, USA
E-mail contact: bharat@scr.siemens.com

Abstract

This paper describes our work in learning on-line models that forecast real-valued variables in a high-dimensional space. A 3GB database was collected by sampling 421 real-valued sensors in a cement manufacturing plant, once every minute, for several months. The goal is to learn models that, every minute, forecast the values of all 421 sensors for the next hour. The underlying process is highly non-stationary: there are abrupt changes in sensor behavior (time-frame: minutes), semi-periodic behavior (time-frame: hours–days), and slow long-term drift in plant dynamics (time-frame: weeks–months). Therefore, the models need to adapt on-line as new data is received; all learning and prediction must occur in real-time (i.e., one minute). The learning methods must also deal with two forms of data corruption: large amounts of data are missing, and what is recorded is very noisy. We have developed a framework with multiple levels of adaptation in which several thousand incremental learning algorithms that adapt on-line are automatically evaluated (also on-line) to arrive at the "best" predictions. We present experimental results to show that by combining multiple learning methods, we can automatically learn good models for time-series prediction without being provided with any physical models of the underlying dynamics.

Introduction

Manufacturing plants typically have systems that sample, display, and store data from several sensors. Buried within this sensor data is valuable information about patterns and trends in the plant operation; for instance, forecasting sensor behavior may provide plant operators with advance warning of crisis situations. Unfortunately, extracting these patterns can be difficult(Fayyad 1996), particularly when the size and dimensionality of the data is high (Locher 1996).

CEMAT, the Siemens plant automation system for cement manufacture, continuously samples hundreds of sensors in a cement plant, displays the information to plant operators, and records it in a massive historical database. A 3GB section of the database (421 sensors, sampled every minute for 6 months) was provided to us; the task is to develop a system which can be automatically deployed in a cement plant, and, after an initial training phase (which consists of "watching" and learning from sensor behavior), the system begins forecasting sensor values in real time; i.e., it provides 60-minute long forecasts for all 421 sensors every minute.

This paper describes MAL (Multiple Adaptive Layers), a *general data mining framework* for time-series forecasting, in which hundreds/thousands of algorithms (instantiated from a tool box of algorithms) are continuously evaluated on-line to determine the most appropriate algorithm for each particular situation.

The primary factors which necessitated the development of the automated MAL framework for time-series forecasting, are the high-dimensionality and volume of the data, the real-time constraints, and the rapidly changing behavior of the data. This high-dimensionality was of particular concern because we are provided with no physical models/knowledge about the sensors. Withholding all sensor information may appear to make the forecasting task unnecessarily difficult – clearly cement plant operators and engineers have this sensor knowledge, which can help bias the search in the 421-dimensional space. However, this lack of information was by design. MAL must be installed in a cement plant with no plant-specific customization. Cement plants have widely-varying behaviors and configurations. Even within a single plant, new sensors could be added, and old sensors removed or replaced over time (e.g., small differences in sensor placement may cause different measurements); furthermore, drifting plant dynamics could radically alter plant behavior over a long period of time. Thus relationships may not hold across plants. While some high-level information is common to all cement plants, the initial focus of this research was to determine if it is possible to forecast highly complex plant dynamics by mining large amounts of data.

In this paper, the CEMAT data is discussed in greater detail. We describe the format we used to test multi-

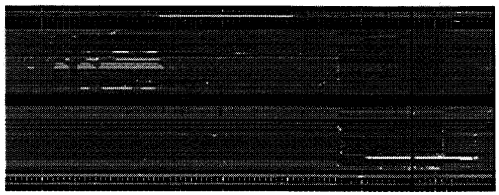

Figure 1: "Missing features" on Day 124. The X-axis shows time in minutes. The Y-axis shows the 421 features (sensors) Plot: "?" (black), "0" (white), real data(gray values).

ple algorithms on the CEMAT data, the results of our initial experiments with "batch" learning algorithms, and the implementation of the *recursive least squares* (RLS) algorithm; an incremental learning algorithm which is an efficient way to learn on-line models. We then present the MAL framework and report experimental results comparing the performance of different learning algorithms on the data. We conclude by discussing some of the lessons we have learned as a result of this project, and potential extensions to MAL.

The CEMAT Data

CEMAT is the Siemens CEMent AuTomation system for controlling cement plants. CEMAT samples hundreds of real-valued sensors, and stores the data in an SQL database. The most critical part of a cement plant is the kiln, a 50-foot long oven which rotates slowly (2-5 rpm) on the factory floor. Cement manufacture is a *continuous* (as opposed to batch) process; the kiln runs 24 hours a day, 365 days a year. Raw material takes 7-8 hours to pass through the kiln and is output as cement clinker (ground to produce cement). The CEMAT data used in this paper was collected from a cement manufacturing plant located outside Hamburg.

The data consists of sensor readings and control signals associated with the kiln. Often, the data acquisition equipment fails to obtain values for some of the sensors. In these cases, these sensors are assigned the value "?". Additionally, some sensors often have long periods where their value is zero. While a zero can be a significant value, it can also mean that a sensor is defective or turned off. Figure 1 illustrates the amount of missing feature sensor information for a typical day.

Testing Prediction Algorithms

We conducted tests on a period of 1 week (Days 131-137) of the 180 days of data. In order to test several different learning algorithms (decision trees, neural networks, statistical algorithms, heuristically-based predictors, rule induction programs, any arbitrary combination of the above), we simulated plant operation by reading values of all 421 sensors from the stored database. The prediction algorithms were provided with all data up to the simulated current time and then the algorithms made forecasts for all 421 sensors. Rather than requiring each algorithm to predict all 60 values for the following hour, only predictions for 1, 2, 5, 10, 15, 30, and 60 minutes in the future were used. The principal evaluation metric was the normalized root-mean-square error (the error is normalized by the standard deviation of each sensor, as measured over Days 1-130). Henceforth, all references to "error" in this paper should be read as the "normalized root-mean-square error."

Each algorithm can be viewed as producing 2947 ($= 421 \times 7$) forecasters, each forecaster making nearly 10,000 predictions (7days x 1440minutes/day) over the test set. We calculated error statistics for each of the 2947 forecasters i.e., mean error over all predictions (with suitable allowances for not calculating error when the data was '?'), aggregated errors per sensor (421 values), per forecast horizon (7 values), plus an overall error.

No limitations were put on the learning algorithm. In general, any past data could be used for making predictions. For example, the forecast for any sensor x_i, θ minutes in the future, at any time t, may use all past data from all sensors in arriving at its prediction:

$$\begin{aligned} \hat{x}_i(t+\theta) &= f(x_1(t), x_1(t-1), \ldots, x_1(0), \\ &\quad \ldots \\ &\quad x_{421}(t), x_{421}(t-1), \ldots, x_{421}(0)) \quad (1) \\ &\quad \theta \in \{1, 2, 5, 10, 15, 30, 60\} \quad (2) \end{aligned}$$

In a cement plant, it will be sufficient to run in real time; i.e., algorithms can take up to 60 seconds to learn and make all predictions. However, this is untenable for testing purposes. If a run over 3-months of data takes 3 months of real time, feedback from evaluation will be too slow. Our algorithms generally ran 10-20 times faster than real-time, completing the one week test in less than one day when running on a Pentium Pro 200 or a Sparc-20.

Experiments

Overview

We evaluated many learning methods on the data:

- CVF (current value forecaster): this simple algorithm provides a baseline.
- Various batch learning algorithms: These violate the time constraints, but illustrate the need for incremental algorithms that adapt on-line to the data.
- RLS: Recursive Least Squares: An efficient incremental algorithm for learning on-line models using least squares methods.
- MOSS: The Mixture of Simple Schemes. Given a set of simple sensor-averaging prediction algorithms, MOSS, for each of the 2947 forecasters, selects an

Scheme	Rating
RLS (incremental)	0.424479
MAL with simple schemes	0.428431
Mixture of Simple Schemes (MOSS)	0.432147
Best Batch Learning Scheme	0.457717
Current Value Forecaster (CVF)	0.477056
Mean Value Forecaster	1.145009

Table 1: Overall error results for all predictions made on Days 131-137 on the CEMAT Data Set. Given the vast number of predictions made by the learning systems (even for a week of test data, over 10,000 predictions are made), differences are significant to well over the 99% confidence level.

algorithm based on a static criterion (here, the algorithm which performed best on a prior test set).

- MAL: The Multiple Adaptive Layers framework extends the idea of MOSS to select (for each forecaster) an algorithm from a set of algorithms based upon current on-line performance.

CVF

We began by implementing a very simple forecaster, the current value forecaster (CVF), whose forecast for $x_i(t+\theta)\ \forall\theta\ \forall x_i$ is the current sensor value $x_i(t)$. Performance of CVF is shown in Table 1.

Batch Learning Methods

To our surprise, CVF proved very hard to beat. All batch learning methods, i.e., those that learned on the training data off-line, or even those that learned new models periodically (say every 8-24 hours), performed significantly worse than CVF. The traditional rule-induction and decision-tree methods (Quinlan 1986) had the worst performance. A possible reason may be the severe non-stationary nature of the data.

To verify this we tested several thousand modifications of numeric computation algorithms on the data. Using the SINELib package (Flake 1998), we were very rapidly able to implement permutations of architecture (linear, nonlinear (tanh, Gaussian, and others), nonlinear with linear short-circuits, radial basis functions, Volterra filters, nonlinear Volterra filters, and time-delay networks), permutations of search (exact solutions for linear models and models with linear portions, exact solutions with respect to recent history, off-line search for recent window, and on-line search), and various preprocessing and heuristic permutations for handling '?', '0', and when to stop training. By trying many hundreds of algorithms, most of which did not meet the real-time constraints, we were able to learn models that significantly outperformed CVF. The performance of the best SINELib routine, called "Best Batch Learning Scheme," is shown in Table 1.

Incremental Learning with Recursive Least Squares

We then investigated learning algorithms that were able to modify themselves in response to the test data, i.e., incremental learning algorithms. In particular, we considered linear prediction using the last few samples to predict the future value. So for a given x_i and θ, we adapt a linear predictor,

$$\hat{x}_i(t+\theta) = \sum_{j=0}^{n-1} a_j x_i(t-j). \tag{3}$$

The criterion that was used to select the parameters a_j was least-squares minimization (LS). In Recursive Least Squares (RLS) estimation, the objective is to obtain coefficients at each time step minimizing the error $\epsilon^2(n)$ over all data received up to the present,

$$\epsilon^2(n) \;=\; \sum_{t=0}^{n} \lambda^{n-t}(\hat{x}_i(t) - x_i(t))^2, \tag{4}$$

where λ is a weighting factor. The solution to this problem requires inversion of a correlation matrix every time a new data sample is obtained, which is computationally expensive. The *Recursive Least Squares* algorithm developed in the signal processing literature(Haykin 1996) addresses this problem. The RLS calculates parameters minimizing $\epsilon^2(n)$ using parameters minimizing $\epsilon^2(n-1)$ and low-rank matrix updates incorporating the new data. The algorithm allows for efficient, exact, on-line solution of the parameters.

Extending the RLS

Instead of using a contiguous window of past values as inputs, the linear regression above can be performed by selecting inputs on a finite grid relative to the current time step over all the observed values of all the sensors. The question is how to find the grid, i.e. the most valuable sensors and delays to use for prediction. These taps can be thought of as "hot"-spots containing maximal information about the value to be predicted.

For a first order predictor, the first hot-spot value is the observation having the highest correlation with the value to be predicted. For higher dimensional prediction, the exact procedure for detecting hotspots requires Gram-Schmidt orthogonalization of the candidate hotspots with all the earlier identified hotspots. This procedure effectively prevents designating as hotspots two sensors which due to their high correlation with each other are redundant. The orthogonalization approach can be implemented efficiently using a Levinson-Durbin scheme(Kay 1987), but in practice the calculation is still remarkably expensive. In our work we therefore also used approximate solution for finding new hotspots by maximizing correlation of the proposed hotspot with the error residual of prediction when the previously selected hotspots are used in prediction.

In addition to including simple sensor values, features generated by simple processing of sensor data streams

were also included. The use of "delta-hotspots" where the forward differences of sensor values are included, is especially useful in reducing the number of parameters in the regressor. We defined a language to allow us to specify arbitrary RLS "taps" (not just the last n samples, but for instance, going back exponentially) and arbitrary other sensors as the hotspots and delta-hotspots. Correlation analysis of the first 130 days of data provided us with an ordered list of candidates Table 1 shows that an RLS with 7 taps / forecaster outperformed all other algorithms that we tried on this test set.

We also developed methods to extend RLS processing by adding strategies that allowed us to specify how to deal with '?' and '0.0' values, filters to deal with noise, heuristics when to stop training, and when to over-ride the RLS prediction. For instance, using these methods we were able to combine RLS's with the C4.5 rule induction algorithms (Quinlan 1992). The idea was to learn if a value of '0.0' in some variable was significant, as a control variable, and use that within the RLS. Although this idea appears interesting, initial results were not consistent, and the area remains open.

Leveraging the Dimensionality: MOSS

An orthogonal direction that we considered was how to deal with the high-dimensionality of the problem. Due to the high-dimensionality, we found that that we focused almost exclusively on the overall error in evaluating schemes, ignoring the nearly 3000 individual error statistics that were calculated (for each of the 2947 forecasters, aggregated for each of the 421 sensors and for each of the 7 prediction horizons). It seemed reasonable *that different algorithms would be better suited for different prediction situations*; i.e., different algorithms would perform differently for different sensors and for different forecast horizons.

As a simple experiment we developed MOSS (Mixture of Simple Schemes), that provided an automated way to select the "best" algorithm for each of the 2947 forecasters, based upon prior performance. A number of very simple sensor-averaging schemes (CVF plus some others, such as the mean-value-forecaster, that always predicts the sensor mean, others that predict some fixed weighted average of the last few samples) were run on the week preceding the test data (i.e., Days 124-130). Then a new scheme (MOSS) was built from these simple schemes, which simply used the error over the preceding week to pick the best scheme for each of the 2947 forecasters.

Table 1 shows that the best SINELib routine ("Best Batch Learning Scheme") was significantly outperformed by the "MOSS" meta-scheme. Note that while the batch learning schemes took several days to run over the test set, MOSS executed within 2 minutes.

Multiple Adaptive Layers Framework

The MAL framework extends the notion of developing a meta-schema from several simple schema to on-line evaluation of schema, by determining after every sample, which scheme to use for the next prediction from many competing schemes. MAL has 3 steps. The first step is an initialization and training phase (say 1 week). This is followed by an evaluation phase of 1-2 weeks where error statistics are generated for use in "switching" during the test phase. Here, "switching" refers to the selection of a algorithm for the current forecast for a given sensor and time offset. During testing, for each of the 2947 forecasters, MAL's output is the prediction made by the scheme which has the best performance based on a switching criterion. This criterion could be the cumulative error, the error during some past window, decaying error measurements, etc. Note that all the schemes can learn during all three phases.

Table 1 shows that, as expected, MAL (using an on-line evaluation criterion) with the simple schemes greatly outperformed MOSS with the same schemes. Furthermore, although MAL ran within 2 minutes, its performance was very close to that of the best incremental algorithm (RLS).

Summary of Experimental Results

The results of Table 1 illustrate the improvements that can be made along two orthogonal directions:

Incremental Learning Algorithms: The "Best Batch Learning Schemes" that took several days to run were outperformed by the RLS incremental learning algorithm that ran well within the alloted time of 24 hours.

Leveraging the Dimensionality: The "Best Batch Learning Schemes" were outperformed by MOSS, that used a static criterion (performance on data the week before the test period) to choose between simple sensor-averaging algorithms. The MOSS notion was further extended in MAL, where an on-line evaluation of the performance of the algorithms was used to select the best algorithm. MAL with simple algorithms ran within minutes, and had performance very similar to that of the RLS.

MAL with many "smart" schemes

The next step was to combine these two notions, and extend MAL to work with multiple "sophisticated" schemes: for instance by using RLS's with different strategies for handling '?' and '0' values, and for stopping training and over-riding predictions. Table 2 shows results using MAL with multiple RLS's on a new section of the data with a reduced number of sensors.

Extensions to MAL

Obviously there is no reason to restrict the adaptation in MAL to only two levels as is currently the case. Note that the output of MAL is identical to that of any learning algorithm, a set of predictions for all 2947 forecasters. Therefore, we could use MAL hierarchically, for instance, using multiple "switching" criteria among

Scheme	Rating
MAL (6 RLS's)	0.363859
MAL (3 RLS's)	0.373590
MAL (SS's)	0.389200
MOSS	0.410660
CVF	0.453434

Table 2: Error results with MAL on new CEMAT data. These are aggregated over 167 of the 421 sensors that were identified as important by cement plant experts.

the same set of algorithms, and then using yet another switcher.

MAL can be extended in many ways. For instance, MAL could "kill" forecasters that have bad performance, and use resources (memory and CPU time) to start up new learning programs. MAL could try to predict when to switch from one forecaster to another.

Comparison: MAL vs. CBR

We also compared MAL to another scheme performing case-based reasoning (CBR). The CBR system tries to find the best previous match to any given situation, and use that to make predictions. The CBR system performs significantly slower than real time. MAL had far superior forecasting performance to the CBR system. Both systems made a total of 2.5 million predictions. MAL's overall error of 0.585 was far better than CBR's error of 0.716 (this was a much noisier section of the data). For the 421 sensors, MAL's aggregate sensor error was lower than CBR's error on 356 sensors, with 55 ties. MAL outperformed CBR by a factor of 2 to 1 on a secondary measure of "closeness".

Conclusions

The MAL framework has been motivated primarily by the problems of non-stationarity, high-dimensionality, real-time computation, high data volume, and automated learning. The two other data problems (noise and missing features) do have a large impact. However, we deal with these by eliminating from consideration algorithms that could not handle such data, by extending algorithms in relatively simple ways to deal with these problems, and finally by letting the MAL framework automatically choose the strategies which worked best on the data. However, integrating techniques for noise-filtering (Rousseeuw & van Zomeren 1990) into the algorithms in MAL can only improve performance.

In this research we have focused on algorithms that can use all available data to learn. Other methods (such as (Apte & Hong 1996)) have achieved success in complex domains by using batch learning methods that produce static models. However, a crucial problem in KDD is to determine what data is "useful" and what can be safely ignored or forgotten(Uthurusamy 1996). This is one of the long-term goals of this research. Currently CEMAT logs all available data from all sensors for several years, leading to unnecessary storage expenses. However, analyzing the learned models, may provide insights into what historical information is important and should be preserved.

In addition to developing the MAL framework, this project has given us some insights that may possibly apply to other projects. This study indicates the need to benchmark results. (Even the straw-man CVF algorithm proved very useful, if only as a powerful motivator to develop better results.) By developing a unified framework, we are quickly able to test a number of diverse algorithms on the data. Finally, even though we did achieve some success using off-the-shelf algorithm, to get maximum benefits from the data, it was necessary to build domain-specific techniques (such as MAL) to tackle the problems found in the CEMAT data.

Acknowledgments

We gratefully acknowledge assistance received from Lueder Heidemann, Stefan Siegel, Michael Brown, and Karsten Schneider, of Siemens AG, Erlangen, Germany, and Gary William Flake, Siemens Corporate Research, Princeton, NJ.

References

Apte, C., and Hong, S. 1996. Predicting equity returns from security data with minimum rule generation. In Fayyad, U. M.; Piatetsky-Shapiro, G.; Smyth, P.; and Uthurusamy, R., eds., *Advances in Knowledge Discovery and Data Mining.* MIT Press. 517–539.

Fayyad, U. M. 1996. From data mining to knowledge discovery: An overview. In Fayyad, U. M.; Piatetsky-Shapiro, G.; Smyth, P.; and Uthurusamy, R., eds., *Advances in Knowledge Discovery and Data Mining.* MIT Press. 1–35.

Flake, G. W. 1998. Sinelib, the Siemens industrial neural engine: Introduction, rationale, and user's guide. Technical Report SCR-98-TR-620, Siemens Corporate Research, Princeton, NJ.

Haykin, S. 1996. *Adaptive Filter Theory.* Prentice Hall, 3 edition.

Kay, S. M. 1987. *Spectral Estimation: Theory and Practice.* Prentice Hall, 1 edition.

Locher, G. 1996. Automatic knowledge acquisition in industrial processes using the example of roller grinding mill. *ZKG International* 49(8).

Quinlan, J. R. 1986. Induction of decision trees. *Machine Learning Journal* 1(1):81–106.

Quinlan, J. R. 1992. *C4.5.* Santa Cruz, CA: Morgan Kaufmann.

Rousseeuw, P., and van Zomeren, B. 1990. Unmasking multivariate outliers and leverage points. *Journal of the American Statistical Association* 85(411):633–651.

Uthurusamy, R. 1996. From data mining to knowledge discovery: Current challenges and future directions. In Fayyad, U. M.; Piatetsky-Shapiro, G.; Smyth, P.; and Uthurusamy, R., eds., *Advances in Knowledge Discovery and Data Mining.* MIT Press. 1–35.