Mining Multivariate Time-Series Sensor Data to Discover Behavior Envelopes

Dennis DeCoste
Monitoring and Diagnosis Technology Group
Jet Propulsion Laboratory / California Institute of Technology
4800 Oak Grove Drive; Pasadena, CA 91109
http://www-aig.jpl.nasa.gov/home/decoste/, decoste@aig.jpl.nasa.gov

Abstract

This paper addresses large-scale regression tasks using a novel combination of greedy input selection and asymmetric cost. Our primary goal is learning envelope functions suitable for automated detection of anomalies in future sensor data. We argue that this new approach can be more effective than traditional techniques, such as static red-line limits, variance-based error bars, and general probability density estimation.

Introduction [1]

This paper explores the combination of a specific feature selection technique and an asymmetric regression cost function which appears promising for efficient, incremental data-mining of large multivariate data.

Motivating this work is our primary target application of automated detection of novel behavior, such as spacecraft anomalies, based on expectations learned by data-mining large data bases of historic performance. In common practice, anomaly detection relies heavily on two approaches: *limit-checking* (checking sensed values against typically-constant, manually-predetermined "red-line" high and low limits) and *discrepancy-checking* (comparing the difference between predicted values and sensed values). Whereas red-lines tend to be cheap but imprecise (i.e. missed alarms), prediction approaches tend to be expensive and *overly precise* (i.e. false alarms).

The framework developed in this paper provides a means to move incrementally from (loose) red-line quality to (tight) prediction-quality expectation models, given suitable volumes of representative historic training data. We independently learn two function approximations — representing the best current estimates of the high and low input-conditional bounds on the sensor's value. In the extreme initial case where the only input to these functions is a constant bias term, the learned bounds will simply reflect expected maximum and minimum values. As additional input features (e.g. sensors or transforms such as lags or means) are selected, these limit functions converge inwards.

A key underlying motivation is that a fault typically manifests in multiple ways over multiple times. Thus, especially in sensor-rich domains common to data-mining, some predictive precision can often be sacrificed to achieve low false alarm rates and efficiency (e.g. small input/weight sets).

The following section presents a simple formulation for learning envelopes. The next two selections introduce our feature selection method and asymmetric cost function, respectively. We then present performance on a real-world NASA example.

Bounds Estimation

We define the *bounds estimation problem* as follows:

Definition 1 (Bounds estimation) *Given a set of patterns $\mathcal{P}$, each specifying values for inputs $x_1, ..., x_d$ and target y generated from the true underlying function $y = f(x_1, ..., x_D) + \varepsilon$, learn high and low approximate bounds $y_L = f_L(x_1, ..., x_l)$ and $y_H = f_H(x_1, ..., x_h)$, such that $y_L \leq y \leq y_H$ generally holds for each pattern, according to given cost functions.*

We allow any $1 \leq l \leq d$, $1 \leq h \leq d$, $d \geq 1$, $D \geq 0$, making explicit both our expectation that some critical inputs of the generator may be completely missing from our patterns and our expectation that some pattern inputs may be irrelevant or useful in determining only one of the bounds. [2] We also make the standard assumption that the inputs are noiseless whereas the target has Gaussian noise defined by ε.

To simplify discussion, we will usually discuss learning only high bounds y_H; the low bounds case is essentially symmetric. An *alarm* occurs when output y_H is below the target y, and a *non-alarm* occurs when $y_H \geq y$. We will call these alarm and non-alarm patterns, denoted by sets $\mathcal{P}_a$ and $\mathcal{P}_n$ respectively, where $\mathcal{N} = |\mathcal{P}| = |\mathcal{P}_a| + |\mathcal{P}_n|$.

[1]

[2] We assume x_1 is a constant bias input of 1 which is always provided. For meaningful comparisons, other inputs with effective weight of zero are not counted in these dimensionality numbers D,d,h, and l.

This paper focusses on linear regression to perform bounds estimation, both for simplicity and because our larger work stresses heuristics for identifying promising explicit nonlinear input features, such as product terms (e.g. (SM91)). Nevertheless, the concepts discussed here should be useful for nonlinear regression as well.

Let $\mathbf{X}$ be a $\mathcal{N}$-row by d-column matrix [3] of (sensor) inputs where each column represents a particular input x_i and the p-th row is a row-vector $\mathbf{X}^{(p)}$ specifying the values of each input for pattern p. Similarly, let $\mathbf{Z}$ be a $\mathcal{N}$-row by z-column *design matrix*, where each column i represents a particular basis function [4] $g_i(x_1, ..., x_d)$ and each row is the corresponding row-vector $\mathbf{Z}^{(p)}$. For each function approximation, such as f_H, there is a corresponding design matrix $\mathbf{Z_H}$ of z_h columns and containing row-vectors $\mathbf{Z_H}^{(p)}$. Let $\mathbf{w_H}$ represent a z_h-row column-vector of weights and $\mathbf{y_H}$ represent a $\mathcal{N}$-row column-vector of outputs such that $\mathbf{y_H} = \mathbf{Z_H w_H}$, and similarly for others (i.e. for f_L and f_M).

The simplest and perhaps most popular way to estimate bounds is to use variance-based error bars. This requires estimating the input-conditional means $y_M = f_M(x_1, ..., x_m)$ $(1 \leq m \leq d)$ and the input-conditional variances σ^2. The bounds for each pattern can then be computed as $y_H = y_M + k * \sigma$ and $y_L = y_M - k * \sigma$, with k=2 yielding 95% confidence intervals under ideal statistical conditions. Standard linear regression via singular value decomposition (SVD) can find least-squared estimates $\mathbf{y_M} = \mathbf{Z_M w_M}$ and variance can be estimated as follows (Bis95): $\mathbf{A} = \alpha\mathbf{I} + \frac{\beta}{\mathcal{N}} \sum_{p \in \mathcal{P}} \mathbf{Z_M}^{(p)} (\mathbf{Z_M}^{(p)})^T$ and $(\sigma^2)^{(p)} = \frac{1}{\beta} + \mathbf{Z}_M^{(p)} \mathbf{A}^{-1} (\mathbf{Z}_M^{(p)})^T$, where β reflects intrinsic noise and α is a small factor ensuring the Hessian $\mathbf{A}$ is positive definite.

However, as the following artificial examples will illustrate, estimating y_L and y_H by estimating y_M using all d inputs and standard sum of squares cost functions is problematic for large high-dimensional data sets.

Artificial Example

For simple illustration, we will discuss our techniques in terms of the following simple example. We generated $\mathcal{N}$=100 patterns for $d = 10$ inputs: bias input $x_1 = 1$ and 9 other inputs $x_2, ..., x_{10}$ randomly from [0,1]. As in all later examples, we normalized each column of $\mathbf{Z_M}$ (except the first (bias) column) to have mean 0 and variance 1. First consider the case where $\mathbf{Z_M} = \mathbf{X_M}$ — i.e. where no feature selection is used. Figure 1 summarizes example results using SVD on all inputs. Note that this can yield significant weight to irrelevant terms (e.g. x_{11}), when attempting to fit nonlinear terms (e.g. $x_3 * x_5$) in the target. [5]

```
RUN: Pn=1,Pa=1,Rn=2,Ra=2,d=10,N=100,SVD.fit,useAllInputs
  target = 5 + x4 + 2*x7 + x9 + x3*x5
  RESULT = 5*x1+0.076*x2+0.081*x3+1*x4+0.06*x5-0.012*x6+
       2*x7+0.028*x8+1.1*x9-0.024*x10-0.11*x11
  non-alarms:  47, error: min=0.02, mean=0.85, max=2.4
      alarms:  53, error: min=-2, mean=-0.75, max=-0.048
```

Figure 1: **No feature selection.**

Feature Selection

We are concerned with regression tasks for which the potential design matrix dimensionality is typically in the hundreds or more, due to large numbers of raw sensors and basis function transforms (e.g. time-lagged values and time-windowed mins and maxs). Despite the relative cheapness of linear regression, its complexity is still quadratic in the number of input features $O(\mathcal{N} * z^2)$. Therefore, we desire a design matrix $\mathbf{Z}$ much smaller than the potential size, and even much smaller than the input matrix $\mathbf{X}$ of raw sensors. Standard dimensionality-reduction methods, such as principal component analysis, are often insufficient. Sensors are often too expensive to be redundant in all contexts.

Whereas constructive methods are often used to incrementally add hidden units to neural networks (e.g. (FL90)) similar attention to incremental selection of inputs per se is relatively rare. The statistical method of forward subset selection (Orr96) is one option. However, efficient formulations are based on orthogonal least squares, which is incompatible with the asymmetric cost function we will soon present.

Instead, we adopt the greedy methods for incremental hidden unit addition to the problem of input selection. Since input units have no incoming weights to train, this amounts to using the same score function as those methods, without the need for any optimization. The basic idea is that each iteration selects the candidate unit (or basis function, in our case) U whose outputs $\mathbf{u}$ covary the most with the current error residuals $\mathbf{e}$. Falhman (FL90) proposed using the standard covariance definition, to give a simple sum over patterns, with mean error $\bar{e}$ and mean (hidden) unit output $\bar{u}$: $S1 = |\sum_{p \in \mathcal{P}} (\mathbf{e}^{(p)} - \bar{\mathbf{e}})(\mathbf{u}^{(p)} - \bar{\mathbf{u}})|$. Kwok recently proposed a normalized refinement (KY94) $S2 = \frac{(\mathbf{e}^T \mathbf{u})^2}{\mathbf{u}^T \mathbf{u}}$. Like Kwok, we have found score S2 to work somewhat better than S1. We note that Kwok's score is very similar to that of forward subset selection based on orthogonal least squares: S3=$\frac{((\mathbf{y_M})^T \hat{\mathbf{u}})^2}{\hat{\mathbf{u}}^T \hat{\mathbf{u}}}$, where the $\hat{\mathbf{u}}$ terms represent outputs of U made orthogonal to the existing columns in the current design matrix. Although Kwok did not note this relation, it appears that S2's scoring via covariance with the error residual provides essentially the same sort of orthogonality.

[3] We use the convention of upper-case bold ($\mathbf{X}$) for matrices, lower-case bold ($\mathbf{x}$) for column-vectors, and normal (x) for scalars and other variables. We use $\mathbf{X}^{(r)}$ to refer to the r-th row-vector of $\mathbf{X}$.

[4] By convention, g_1 is the constant bias input $x_1 = 1$.

[5] For SVD fits, we report "alarm" statistics as if $y_H = y_M$, to illustrate the degree of error symmetry. Actual alarms would be based on error bars.

We start with a small design matrix $\mathbf{Z_M}$ that contains at least the bias term (g_1), plus any arbitrary basis functions suggested by knowledge engineers (none for examples in this paper). At each iteration round, we compute the column-vector of current error residuals $\mathbf{e} = \mathbf{y_M} - \mathbf{y}$ for all patterns and add to $\mathbf{Z_M}$ the input U with the highest S2 score.

Artificial Example Using Feature Selection

The result of our feature selection method on our artificial example is summarized in Figure 2. Note that while the most relevant inputs are properly selected first, the nonlinear term in the target causes x_3 to be selected — even though its resulting weight does not (and cannot, using linear estimation alone) reflect the true significance of x_3.

```
RUN: Pn=1,Pa=1,Rn=2,Ra=2,d=10,N=100,SVD.fit
  target = 5 + x4 + 2*x7 + x9 + x3*x5
— Selection cycle 1: avg train err=7.9, alarms=47, non=53:
  fit = 5*x1
  validation errors: avg err=7.35503, alarms=51, non=49:
  scores: x7:67 x4:32 x9:25 x11:8.6 x5:4.3 x3:4.0 x8:1.1 x10:0.8
— Selection cycle 2: avg train err=3.3, alarms=51, non=49:
  fit = 5*x1+2.2*x7
  validation errors: avg err=2.8254, alarms=53, non=47:
  scores: x9:53 x4:42 x3:5.1 x11:2.5 x8:2.4 x10:2.0 x5:1.9 x6:1.1
— Selection cycle 3: avg train err=2, alarms=57, non=43:
  fit = 5*x1+2.1*x7+1.2*x9
  validation errors: avg err=1.70093, alarms=50, non=50:
  scores: x4:67 x8:3.9 x2:2.8 x3:2.2 x6:2.0 x1:1.3 x11:0.9 x5:0.3
— Selection cycle 4: avg train err=1, alarms=55, non=45:
  fit = 5*x1+2*x7+1.1*x9+1*x4
  validation errors: avg err=0.975131, alarms=51, non=49:
  scores: x3:3.4 x11:2.9 x5:1.3 x2:0.8 x1:0.6 x9:0.5 x6:0.1 x4:0.06
— Selection cycle 5: avg train err=1, alarms=55, non=45:
  fit = 5*x1+2*x7+1.1*x9+1*x4+0.092*x3
  validation errors: avg err=0.956103, alarms=51, non=49:
  scores: x5:2.2 x11:1.9 x9:0.7 x1:0.7 x6:0.7 x2:0.6 x3:0.3 x7:0.3
— Selection cycle 6: avg train err=1, alarms=53, non=47:
  fit = 5*x1+2*x7+1.1*x9+1*x4+0.093*x3+0.054*x5
  validation errors: avg err=0.965555, alarms=51, non=49:
STOP: validation error worse  ... retract last cycle!
```

Figure 2: **Feature selection: nonlinear target**
Error bars for α=1.0e-20 and β=10; target in bold.

Asymmetric Cost Function

Probability density estimation (PDE) approaches (e.g. (WS95)) are more general than error bars (e.g. (NW95)). However, PDE is also more expensive, in terms of both computation and amounts of training data required to properly estimate the input-conditional probabilities across the entire output range. For example, consider learning worst and best case complexity bounds for quicksort (i.e. $O(N^2)$ and $\Omega(NlgN)$). The variances between the expected case and the worst and best cases are not symmetric, making error bars inappropriate. Whereas PDE would learn more than is required for the bounds estimation task per se.

Our basic intuition is that the cost function should discourage outputs below (above) the target for learning high (low) bounds. To do this, we split the task of bounds estimation into two independent regressions over the same set of patterns $\mathcal{P}$ — one to learn the expected high bound f_H and one to learn the expected low bound f_L. Figure 3 defines respective asymmetric cost functions for errors E_H and E_L over $\mathcal{P}$.

$$e_H = \begin{cases} P_{H_n}(y_H - y)^{R_{H_n}} & \text{if } y_H \geq y \\ P_{H_a}(y_H - y)^{R_{H_a}} & \text{if } y_H < y \end{cases} \quad e_L = \begin{cases} P_{L_n}(y_L - y)^{R_{L_n}} & \text{if } y_L \leq y \\ P_{L_a}(y_L - y)^{R_{L_a}} & \text{if } y_L > y \end{cases}$$

$$E_H = \frac{1}{N}\sum_{p\in\mathcal{P}} e_H,\ E_{|H|} = \frac{1}{N}\sum_{p\in\mathcal{P}} |e_H|, E_L = \frac{1}{N}\sum_{p\in\mathcal{P}} e_L,$$

Figure 3: **Asymmetric high/low cost functions.**
Parameters: $P_{H_a}, P_{H_n}, P_{L_a}, P_{L_n} \geq 0$; $R_{H_a}, R_{H_n}, R_{L_a}, R_{L_n} \geq 1$.

We can favor non-alarms (i.e. looseness) over alarms (incorrectness) by making $P_{H_a} > P_{H_n}$. This is analogous to the use of nonstandard loss functions to perform risk minimization in classification tasks (Bis95).

The special symmetric case of $P_{H_n} = P_{H_a} = P_{L_n} = P_{L_a} = 1$ and $R_{H_n} = R_{H_a} = R_{L_n} = R_{L_a} = 2$ gives standard least-squares regression. Thus, in the limit of sufficient inputs and training patterns, both bounds can converge to the standard means estimation (i.e. f_M).

Efficient Training

Our asymmetric cost function is incompatible with standard linear regression methods based on SVD. Instead, we batch optimization via Newton's method (Bis95): $\mathbf{w_H}(t) = \mathbf{w_H}(t-1) - \mathbf{A}^{-1}\mathbf{g}$ for each epoch t, where $\mathbf{g}$ is the z-row vector gradient of elements $\frac{\delta E_H}{\delta w_i}$ and $\mathbf{A}$ is the z x z Hessian matrix of elements $\frac{\delta^2 E_H}{\delta w_i \delta w_j}$. For each pattern $p \in \mathcal{P}$: $y_H = \sum_{i=1}^{z} w_i z_i$, $\mathbf{Z_H}^{(p)} = [z_1, ..., z_z]^T$, and $\mathbf{w_H} = [w_1, ..., w_z]^T$.

For our specific cost function E_H, the elements of the gradient at each epoch can be computed by averaging over alarm and non-alarm patterns, as follows :

$$\frac{\delta E_H}{\delta w_i} = \frac{1}{N}\Big[\ P_{H_n} R_{H_n} \sum_{p\in\mathcal{P}_n} |y_H - y|^{R_{H_n}-1} z_i \ - P_{H_a} R_{H_a} \sum_{p\in\mathcal{P}_a} |y_H - y|^{R_{H_a}-1} z_i\Big]$$

With $e_p = |y_H - y|$ for each pattern p, the elements of $\mathbf{A}$ are partial derivatives of $\mathbf{g}$: $\frac{\delta E_H}{\delta w_i \delta w_j} = \frac{1}{N}\Big[\ P_{H_n} R_{H_n} \sum_{p\in\mathcal{P}_n} [(R_{H_n}-1) e_p^{R_{H_n}-2} z_i z_j] + P_{H_a} R_{H_a} \sum_{p\in\mathcal{P}_a} [(R_{H_a}-1) e_p^{R_{H_a}-2} z_i z_j]\Big]$

For $R_{H_n} = R_{H_a} = 2$, $\mathbf{A}$ simplifies to: $\frac{\delta E_H}{\delta w_i \delta w_j} = \frac{2}{N} P_{H_n} [\sum_{p\in\mathcal{P}_n} z_i z_j] + \frac{2}{N} P_{H_a} [\sum_{p\in\mathcal{P}_a} z_i z_j]$.

We start with initial weights $\mathbf{w}(0)$ given by SVD. Those initial weights are particularly useful for learning tighter low bounds with P_{H_n}=0, where initial zero weights would immediately satisfy our asymmetric cost function. We run Newton until convergence (i.e. all elements in gradient $\mathbf{g}$ near zero) or 100 epochs reached (rare even for large multi-dimensional data).

Spot-Checking for R and P Parameters

Instead of attempting to find optimal values for R,P parameters, we currently spot-check generally useful combinations, such as: $R_{H_n} \in \{1,2\}$, $R_{H_a} \in \{2,10,20\}$, $P_{H_n} \in \{1,0.1,.01,.001,1e\text{–}5,1e\text{–}10,1e\text{–}15\}$, $P_{H_a} \in \{1,1000\}$. We train to obtain weight vectors for each combination and then select the weights giving smallest cost (using common set of reference parameters, such as $R_{H_n}=2, R_{H_a}=2, P_{H_n}=0.0001, P_{H_a}=1$) on validation data. For large data sets, training first with various R and P on small random subsets often quickly identifies good values for the whole set.

Feature Selection With Asymmetric Cost

Our earlier definition of feature selection score S2 assumed a symmetric cost function. We want to prefer features likely to reduce the most costly errors, while still measuring covariance with respect to the true error residuals $\mathbf{e} = \mathbf{y_H} - \mathbf{y}$, not the asymmetric errors. This leads to our following weighted form of S2:

$$S2' = \frac{E_{|Hn|}}{E_{|H|}} \frac{\sum_{p\in\mathcal{P}_n} (\mathbf{e}^{(p)}\mathbf{u}^{(p)})^2}{\sum_{p\in\mathcal{P}_n} (\mathbf{u}^{(p)})^2} + \frac{E_{|Ha|}}{E_{|H|}} \frac{\sum_{p\in\mathcal{P}_a} (\mathbf{e}^{(p)}\mathbf{u}^{(p)})^2}{\sum_{p\in\mathcal{P}_a} (\mathbf{u}^{(p)})^2},$$

$$E_{|Ha|} = \frac{1}{N}\sum_{p\in\mathcal{P}_a} |e_H| \quad E_{|Hn|} = \frac{1}{N}\sum_{p\in\mathcal{P}_n} |e_H|.$$

Artificial Example Using Asymmetric Cost

Figure 4 summarizes results using both selection and asymmetric cost, for the best spot-checked R,P values. The high bound here is much tighter than in Figure 2.

```
RUN: Pn=0.01,Pa=1,Rn=2,Ra=2,d=10,N=100,HI.bound
  target = 5 + x4 + 2*x7 + x9 + x3*x5
— Selection cycle 1: avg train err=0.34, alarms=5, non=95:
  fit = 9.6*x1
  validation errors: avg err=0.352449, alarms=3, non=97:
  scores: x7:134 x9:49 x4:48 x11:11 x3:3.8 x6:3.3 x8:2.1 x2:2.0
— Selection cycle 2: avg train err=0.15, alarms=5, non=95:
  fit = 7.8*x1+2.2*x7
  validation errors: avg err=0.145489, alarms=5, non=95:
  scores: x9:77 x4:63 x8:7.6 x3:5.1 x10:3.8 x6:3.0 x11:3.0 x5:1.2
— Selection cycle 3: avg train err=0.093, alarms=5, non=95:
  fit = 7.2*x1+2.5*x7+1.2*x9
  validation errors: avg err=0.138962, alarms=5, non=95:
  scores: x4:79 x7:16 x2:5.4 x8:4.6 x3:2.1 x6:0.7 x11:0.6 x1:0.3
— Selection cycle 4: avg train err=0.038, alarms=9, non=91:
  fit = 6.6*x1+1.9*x7+1.1*x9+1*x4
  validation errors: avg err=0.0541614, alarms=5, non=95:
  scores: x1:4.8 x11:2.2 x7:1.7 x9:1.7 x6:0.9 x4:0.8 x2:0.5 x10:0.4
— Selection cycle 5: avg train err=0.037, alarms=10, non=90:
  fit = 6.5*x1+1.9*x7+1*x9+1*x4-0.14*x11
  validation errors: avg err=0.0549278, alarms=7, non=93:
STOP: validation error worse ... retract last cycle!
```

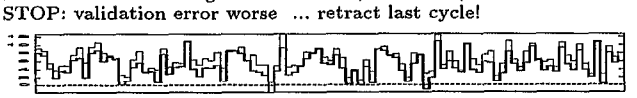

Figure 4: **Feature selection and asymmetric cost.**

Real-World Example: TOPEX

Figure 5 summarizes learning a high bound for high-dimensional time-series data. This data set consists of 1000 successive patterns of 56 sensors of the NASA TOPEX spacecraft. This result was obtained for the predictive target being the value of x_{19} in the next pattern and with cost parameters P_{H_n}=1e-15, P_{H_a}=1, R_{H_n}=2, R_{H_a}=10. The first selected non-bias input was, quite reasonably, the target sensor itself. Note that the weight of the bias (x_1) tends to drop as additional features are selected to take over its early role in minimizing the alarm error.

```
— Selection cycle 1: avg train err=8.4e-15, alarms=7, non=992:
  fit = 2.6*x1
  scores: x19:2929 x51:560 x43:541 x56:468 x47:435 x41:384 x42:293
— Selection cycle 2: avg train err=6.5e-15, alarms=1, non=998:
  fit = 2.3*x1+0.16*x19
  scores: x19:2995 x43:585 x51:578 x56:500 x47:449 x41:409 x42:322
— Selection cycle 3: avg train err=5.9e-15, alarms=1, non=998:
  fit = 2.2*x1+0.2*x19+0.028*x43
  scores: x19:2906 x51:511 x43:491 x56:476 x47:375 x41:337 x17:281
— Selection cycle 4: avg train err=3.9e-15, alarms=1, non=998:
  fit = 1.8*x1+0.31*x19+0.075*x43-0.22*x51
  scores: x19:2139 x56:236 x17:198 x52:104 x55:77 x23:72 x45:61
— Selection cycle 5: avg train err=3.9e-15, alarms=1, non=998:
  fit = 1.8*x1+0.31*x19+0.075*x43-0.22*x51+0.00029*x56
  scores: x19:2133 x56:234 x17:198 x52:106 x55:77 x23:71 x45:61
— Selection cycle 6: avg train err=5.8e-08, alarms=12, non=987:
  fit = 1.4*x1+0.33*x19+0.053*x43+0.076*x51+0.12*x56-0.011*x17
STOP: err getting worse ... retract last cycle!
```

Figure 5: **TOPEX example.**

Conclusion

This framework supports an anytime approach to large-scale incremental regression tasks. Highly-asymmetric cost can allow useful bounds even when only a small subset of the relevant features have yet been identified. Incorporating feature-construction (e.g. (SM91)) is one key direction for future work.

Acknowledgements

This work was performed by Jet Propulsion Laboratory, California Institute of Technology, under contract with National Aeronautics and Space Administration.

References

Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

Dennis DeCoste. Automated learning and monitoring of limit functions. In *Proceedings of the Fourth International Symposium on Artificial Intelligence, Robotics, and Automation for Space*, Japan, July 1997.

Scott Falhman and C. Lebiere. The cascade-correlation learning architecture. *NIPS-2*, 1990.

Tin-Yau Kwok and Dit-Yan Yeung. Constructive neural networks: Some practical considerations. In *Proceedings of International Conference on Neural Networks*, 1994.

David A. Nix and Andreas S. Weigend. Learning local error bars for nonlinear regression. *NIPS-7*, 1995.

Mark Orr. Introduction to radial basis function networks. Technical Report 4/96, Center for Cognitive Science, University of Edinburgh, 1996. (http://www.cns.ed.ac.uk/people/mark/intro/intro.html).

Richard S. Sutton and Christopher J. Matheus. Learning polynomial functions by feature construction. In *Proceedings of Eighth International Workshop on Machine Learning*, 1991.

Andreas S. Weigend and Ashok N. Srivastava. Predicting conditional probability distributions: A connectionist approach. *International Journal of Neural Systems*, 6, 1995.