5 Linear learning systems

In this chapter, we will discuss different linear learning algorithms. We start with regression systems: Batch learning linear regression and adaptive linear filters. We end with linear classification/discrimination algorithms like the Adaline, the simple Perceptron, logistic regression, and the Gaussian classifier.

5.1 Linear regression

The linear regression problem is:

$$\begin{aligned} \text{We assume a linear process } y &= \mathbf{w}_*^T\mathbf{x} + \varepsilon \quad (5.1) \\ \text{Our model is linear } \hat{y} &= \mathbf{w}^T\mathbf{x} \quad (5.2) \end{aligned}$$

where we assume that the noise process ε fulfills the Gauss-Markov conditions, and a possible intercept term (w_0) is included in the parameter vector $\mathbf{w}$. We solve the problem by minimizing the summed square error

$$E = \text{SSE} = \sum_{n=1}^{N} (y(n) - \hat{y}(n))^2 \quad (5.3)$$

The solution is most easily derived by working with a vector/matrix formulation of SSE. We introduce the following matrix

$$\mathbf{X} = \begin{pmatrix} 1 & x_1(1) & x_2(1) & \cdots & x_D(1) \\ 1 & x_1(2) & x_2(2) & \cdots & x_D(2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1(N) & x_2(N) & \cdots & x_D(N) \end{pmatrix} \quad (5.4)$$

where every row corresponds to an observation, and every column to a variable (the first column is the dummy variable multiplying with the intercept term w_0). We also introduce the column vector

$$\mathbf{y} = \begin{pmatrix} y(1) \\ y(2) \\ \vdots \\ y(N) \end{pmatrix} \quad (5.5)$$

and the corresponding column vector $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$.

The SSE can then be written as

$$E = \text{SSE} = [\mathbf{y} - \hat{\mathbf{y}}]^T[\mathbf{y} - \hat{\mathbf{y}}] = \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{y} + \mathbf{y}^T\mathbf{y}. \qquad (5.6)$$

where it is important to note that this is a quadratic form. That is, it has only one unique extremum (if the matrix $\mathbf{X}^T\mathbf{X}$ is non-singular). There are no suboptimal minima in linear regression, there is only one minima and it is the solution.

Requiring $\nabla_w E = 0$ in (5.6) gives us

$$\nabla_w E = 2\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\mathbf{y} = 0 \qquad (5.7)$$

with the solution

$$\mathbf{w} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}. \qquad (5.8)$$

The expression $\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T$ is called the *pseudoinverse* of the matrix $\mathbf{X}$. (The normal inverse does not apply, since $\mathbf{X}$ is non-quadratic.)

In signal processing, one often speaks of the *Wiener equations*, which define the solution for the linear Wiener filter. The Wiener equations are

$$\mathbf{R}_{xx}\mathbf{w} - \mathbf{r}_{xy} = 0 \qquad (5.9)$$

where $\mathbf{R}_{xx}$ is the *correlation matrix*, and $\mathbf{r}_{xy}$ the *cross-correlation vector*, defined as

$$[\mathbf{R}_{xx}]_{kl} = \frac{1}{N}\sum_{n=1}^{N} x_k(n)x_l(n) \qquad (5.10)$$

$$[\mathbf{r}_{xy}]_{k} = \frac{1}{N}\sum_{n=1}^{N} x_k(n)y(n) \qquad (5.11)$$

which means that $\mathbf{R}_{xx} = \mathbf{X}^T\mathbf{X}/N$, and that $\mathbf{r}_{xy} = \mathbf{X}^T\mathbf{y}/N$. The Wiener equations are thus equivalent to (5.7).

5.2 Ridge regression

What happens if $\mathbf{X}^T\mathbf{X}$ is singular, or almost singular (*ill-conditioned*)? Then we cannot simply invert it and find the solution. One fix is to *regularize* the matrix $\mathbf{X}^T\mathbf{X}$ by adding a small number on the diagonal

$$\mathbf{X}^T\mathbf{X} \rightarrow \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right) \qquad (5.12)$$

This makes the matrix inverible, since we effectively filter away all eigenvalues smaller than λ. This is called *ridge regression* and the solution is

$$\mathbf{w} = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{y}. \tag{5.13}$$

One very interesting observation about ridge regression is the fact that there always exists a positive λ such that the generalization error is smaller than the generalization error for the linear regression estimate (Wetherill 1986). This is because λ is a "knob" that allows us to tune the bias vs. variance trade-off, and there exists a positive λ value such that the decrease in model variance is larger than the increase in model bias (squared).

Ridge regression has an interesting Bayesian interpretation. Ridge regression is equivalent to using the error function

$$E = \sum_{n=1}^{N}\left(y(n) - \hat{y}(n)\right)^2 + \lambda\|\mathbf{w}\|^2 \tag{5.14}$$

The Bayesian error measure is

$$E = -\ln\mathcal{L}(\mathcal{X}|\mathbf{w}) - \ln p(\mathbf{w}) \tag{5.15}$$

where $p(\mathbf{w})$ is our prior on the parameters $\mathbf{w}$. Assuming a Gaussian noise distribution, we have that the negative log likelihood is (ignoring irrelevant terms)

$$-\ln\mathcal{L}(\mathcal{X}|\mathbf{w}) = \frac{1}{2\sigma_\varepsilon^2}\sum_{n=1}^{N}\left(y(n) - \hat{y}(n)\right)^2 \tag{5.16}$$

so that

$$E = \frac{1}{2\sigma_\varepsilon^2}\sum_{n=1}^{N}\left(y(n) - \hat{y}(n)\right)^2 - \ln p(\mathbf{w}) \tag{5.17}$$

Comparing (5.17) with (5.14), we see that we can make the identification (ignoring irrelevant terms)

$$-\ln p(\mathbf{w}) = \frac{\lambda}{2\sigma_\varepsilon^2}\|\mathbf{w}\|^2 \tag{5.18}$$

i.e. ridge regression is equivalent to imposing a Gaussian prior on the weights $\mathbf{w}$, $p(\mathbf{w}) \propto \exp\left(\frac{-\lambda}{2\sigma_\varepsilon^2}\|\mathbf{w}\|^2\right)$, and the factor λ tunes the width of this distribution.

5.3 Gradient descent

Another fix to the singularity problem is to avoid inverting the matrix $\mathbf{X}^T\mathbf{X}$ by using a gradient descent scheme, which iteratively searches for the solution to (5.3). This strategy is slow but robust. The algorithm is based on

adapting the parameters in the direction of the negative gradient of E

$$\Delta \mathbf{w} \propto -\nabla_w E = 2 \sum_{n=1}^{N} \left(y(n) - \hat{y}(n) \right) \mathbf{x}(n) = 2 \sum_{n=1}^{N} e(n) \mathbf{x}(n) \qquad (5.19)$$

The gradient descent algorithms goes as follows (the factor 2 above is usually absorbed into the step size, or learning rate, η):

1. Initialize $\mathbf{w} = \mathbf{w}_{init}$
2. Repeat until convergence (either when the error E is below some preset value or until the gradient $\nabla_w E$ is smaller than a preset value), t is the iteration number
   - 2.1 Compute the update $\Delta \mathbf{w}(t) = -\eta \nabla_w E(t) = \eta \sum_{n=1}^{N} e(n,t) \mathbf{x}(n)$ where $e(n,t) = \Big(y(n) - \mathbf{w}^T(t) \mathbf{x}(n) \Big)$
   - 2.2 Update the weights $\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta \mathbf{w}(t)$
   - 2.3 Compute error $E(t+1)$

5.4 Linear adaptive filters: The LMS algorithm

The most well-known form of the gradient descent algorithm, at least in the field of adaptive control and adaptive signal processing, is the "on-line" version, which goes under the name of the *least mean square* (LMS) algorithm. On-line means that an update is made after each observation $\mathbf{x}(n)$, and not after looping through the whole training data set. The latter form is usually termed "batch" learning. The LMS algoritm is e.g. suitable in a scenario where data is coming in in an endless stream, and goes like

1. Initialize $\mathbf{w} = \mathbf{w}_{init}$
2. Repeat (essentially forever)
   - 2.1 Compute the update $\Delta \mathbf{w}(n) = \eta e(n) \mathbf{x}(n)$ where $e(n) = \Big(y(n) - \mathbf{w}^T(n) \mathbf{x}(n) \Big)$
   - 2.2 Update the weights $\mathbf{w}(n+1) = \mathbf{w}(n) + \Delta \mathbf{w}(n)$

The crucial parameter is η. A too large η means that the algorithm fails to converge (in fact, it diverges), a too small η means that it takes "forever" to converge. It can be shown that the learning rate should lie in the interval

$$0 < \eta < \frac{2}{\text{Trace}\left[\mathbf{R}_{xx}\right]} = \frac{2}{\text{Trace}\left[\mathbf{X}^T \mathbf{X}\right]} \qquad (5.20)$$

That $\mathbf{R}_{xx}$ enters this condition is not surprising, since it is the curvature matrix (Hessian), i.e. the second order derivative, in the quadratic error expression (5.6).

Of course, a constant learning rate is not in general the best. A large learning rate is usually beneficial in the initial phase of the learning, and then the learning rate should decrease towards the later stages of learning, when it is important to take small steps to avoid oscillation effects etc..

A good learning algorithm estimates the learning rate automatically.

5.5 The Adaline

The Adaline (*Adaptive LINear Element*) is an adaptive linear two-class classifier that builds on the LMS algorithm. The Adaline uses discrete target values

$$y = \begin{cases} +1 & \text{if } \mathbf{x} \text{ belongs to class 1} \\ -1 & \text{if } \mathbf{x} \text{ belongs to class 2} \end{cases} \tag{5.21}$$

The output of the Adaline is

$$\hat{y} = \text{sign}\left(\mathbf{w}^T\mathbf{x}(n)\right) = \begin{cases} +1 & \text{if } \mathbf{w}^T\mathbf{x} > 0 \\ -1 & \text{if } \mathbf{w}^T\mathbf{x} < 0 \end{cases} \tag{5.22}$$

A straightforward application of gradient descent on SSE here requires that we take the derivative of the sign function, which is $\text{sign}'(z) = 2\delta(z)$, with $\delta(z)$ denoting the Dirac delta function. This derivative is of course impossible to use, and it was therefore suggested that the LMS algorithm should be implemented without it (i.e. simply skipping over it). This turned out to work quite well, although the lack of principle is disturbing. However, we will return to this and try to explain why this learning works later on in this lecture.

5.6 The simple Perceptron

The simple Perceptron is another adaptive linear two-class classifier. It takes binary targets

$$y = \begin{cases} 1 & \text{if } \mathbf{x} \text{ belongs to class 1} \\ 0 & \text{if } \mathbf{x} \text{ belongs to class 2} \end{cases} \tag{5.23}$$

The output of the simple Perceptron is

$$\hat{y} = \Theta\left(\mathbf{w}^T\mathbf{x}(n)\right) = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x} > 0 \\ 0 & \text{if } \mathbf{w}^T\mathbf{x} < 0 \end{cases} \tag{5.24}$$

The function $\Theta(z)$ is usually called the "Heaviside" function, after the person who first introduced it.

The simple Perceptron is trained using "Perceptron learning", see e.g. (Haykin 1999) or (Bishop 1995). This learning algorithm is not very useful, and not generalizable to more interesting nonlinear systems, so we will not waste time with it here.

5.7 Gradient descent and LMS for a sigmoidal Perceptron

The only reason why the Adaline cannot be trained with a correct gradient descent algorithm is that the transfer function $\text{sign}(z)$ does not have a proper derivative. If we replace the sign function with a function that has a proper derivative, then we should be able to train the Adaline in a proper way (and the simple Perceptron too, for that matter).

We can do this by replacing the sign function (or the Heaviside function) with a smooth version:

$$\text{sign}(z) \quad \rightarrow \quad \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{5.25}$$

$$\Theta(z) \quad \rightarrow \quad \frac{1}{1 + e^{-z}} \tag{5.26}$$

Both these functions have proper derivatives, which are

$$\phi(z) = \tanh(z) \quad \rightarrow \quad \phi'(z) = 1 - \phi^2(z) \tag{5.27}$$

$$\phi(z) = \frac{1}{1 + e^{-z}} \quad \rightarrow \quad \phi'(z) = \phi(z)\left(1 - \phi(z)\right) \tag{5.28}$$

If we now use a Perceptron (or Adaline) with the functional form

$$\hat{y} = \phi(\mathbf{w}^T\mathbf{x}) \tag{5.29}$$

where $\phi(z)$ is any of the sigmoidal functions above, then it is traightforward to derive the gradient descent equations for training this Perceptron. The gradient descent weight update is (if we use SSE)

$$\Delta\mathbf{w} = -\eta\nabla_w E = \eta 2 \sum_{n=1}^{N} e(n)\phi'\left(\mathbf{w}^T\mathbf{x}(n)\right)\mathbf{x}(n) \tag{5.30}$$

which should be compared to expression (5.19).

The gradient descent batch learning algorithm for this Perceptron thus reads (again, the factor 2 is absorbed into η):

1. Initialize $\mathbf{w} = \mathbf{w}_{init}$
2. Repeat until convergence (either when the error E is below some preset value or until the gradient $\nabla_w E$ is smaller than a preset value), t is the iteration number
   - 2.1 Compute the update $\Delta\mathbf{w}(t) = \eta \sum_{n=1}^{N} e(n,t)\phi'\left(\mathbf{w}^T(t)\mathbf{x}(n)\right)\mathbf{x}(n)$ where $e(n,t) = \left(y(n) - \phi(\mathbf{w}^T(t)\mathbf{x}(n))\right)$
   - 2.2 Update the weights $\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta\mathbf{w}(t)$
   - 2.3 Compute error $E(t+1)$

Similarly, the on-line form is

1. Initialize $\mathbf{w} = \mathbf{w}_{init}$
2. Repeat (essentially forever)
   - 2.1 Compute the update $\Delta\mathbf{w}(n) = \eta e(n)\phi'\left(\mathbf{w}^T(n)\mathbf{x}(n)\right)\mathbf{x}(n)$ where $e(n) = \left(y(n) - \phi(\mathbf{w}^T(n)\mathbf{x}(n))\right)$
   - 2.2 Update the weights $\mathbf{w}(n+1) = \mathbf{w}(n) + \Delta\mathbf{w}(n)$

If we compare this to the standard Adaline learning (i.e. LMS) then we see that the difference is not that great. The additional factor $\phi'(z)$ is always a positive number and if the learning rate η is small then it should work fine with LMS instead of this correct form (which is also the case). Thus, it is not surprising that the Adaline learning works. However, using the correct form above is more efficient.

Gradient descent learning using sigmoidal transfer functions $\phi(z)$ is easily generalized to more complicated and nonlinear network structures. This will be the subject of the next lecture.

Note: The only difference between the Adaline and the simple Perceptron is the learning algorithm. Adaline is usually used to denote a classifier where the LMS algorithm is used.

5.8 The linear Gaussian classifier

A common and simple linear classifier in statistics is the *parametric* Gaussian classifier. The key assumption here is that the data in each category follows a Gaussian distribution, i.e.

$$p(\mathbf{x}|c_k) = \frac{1}{(2\pi)^{D/2}\sqrt{|\mathbf{\Sigma}|}} \exp\left[\frac{-(\mathbf{x}-\mu_k)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu_k)}{2}\right]. \tag{5.31}$$

The likelihood for the whole data set is

$$\begin{aligned}\mathcal{L} &= \prod_{k=1}^{K}\prod_{\mathbf{x}(n)\in c_k} \hat{p}(\mathbf{x}(n)|c_k) \\ &= \prod_{k=1}^{K}\prod_{\mathbf{x}(n)\in c_k} \frac{1}{(2\pi)^{D/2}\sqrt{|\hat{\mathbf{\Sigma}}|}} \exp\left[\frac{-(\mathbf{x}(n)-\hat{\mu}_k)^T\hat{\mathbf{\Sigma}}^{-1}(\mathbf{x}(n)-\hat{\mu}_k)}{2}\right]\end{aligned} \tag{5.32}$$

where $\hat{\mathbf{\Sigma}}$ and $\hat{\mu}_k$ are our estimated values for the covariance matrix and the mean. Maximizing $\mathcal{L}$ w.r.t. $\hat{\mathbf{\Sigma}}$ and $\hat{\mu}_k$ gives

$$\hat{\mu}_k = \frac{1}{N_k}\sum_{\mathbf{x}(n)\in c_k} \mathbf{x}(n) \tag{5.33}$$

$$\hat{\mathbf{\Sigma}} = \sum_k \frac{N_k}{N}\hat{\mathbf{\Sigma}}_k = \frac{1}{N}\sum_k \sum_{\mathbf{x}(n)\in c_k} (\mathbf{x}(n)-\hat{\mu}_k)(\mathbf{x}(n)-\hat{\mu}_k)^T. \tag{5.34}$$

A new observation $\mathbf{x}$ is classified by computing the probability

$$p(c_k|\mathbf{x}) = \frac{p(c_k)p(\mathbf{x}|c_k)}{p(\mathbf{x})}, \tag{5.35}$$

where

$$p(c_k) = \frac{N_k}{N} \tag{5.36}$$

is the sample estimate for the a priori probabilities.

The decision boundary between two categories c_k och c_j is given by

$$\begin{aligned} p(c_k|\mathbf{x}) &= p(c_j|\mathbf{x}) \Rightarrow \\ \ln\left[\frac{p(c_k)}{p(c_j)}\right] &= \frac{(\mathbf{x}-\hat{\mu}_k)^T\hat{\mathbf{\Sigma}}^{-1}(\mathbf{x}-\hat{\mu}_k)}{2} \\ &\quad -\frac{(\mathbf{x}-\hat{\mu}_j)^T\hat{\mathbf{\Sigma}}^{-1}(\mathbf{x}-\hat{\mu}_j)}{2} \end{aligned} \tag{5.37}$$

which defines a hyperplane, i.e. the classifier is a linear classifier.

If the a priori probabilities $p(c_k)$ are equal, then the decision boundaries become

$$(\mathbf{x} - \hat{\mu}_k)^T \hat{\mathbf{\Sigma}}^{-1} (\mathbf{x} - \hat{\mu}_k) = (\mathbf{x} - \hat{\mu}_j)^T \hat{\mathbf{\Sigma}}^{-1} (\mathbf{x} - \hat{\mu}_j). \tag{5.38}$$

Note: $(\mathbf{x} - \mu_k)^T \mathbf{\Sigma}^{-1} (\mathbf{x} - \mu_k) = \|\mathbf{x} - \mu_k\|_\Sigma^2$ is usually called the *Mahalanobis distance* between $\mathbf{x}$ and μ_k. A point $\mathbf{x}$ is thus classified as belonging to category c_k if it lies closer to the mean μ_k for category c_k, than to any other mean when the distance is measured with the Mahalanobis distance.

5.9 The logistic regression

The logistic regression is another linear parametric classifier. The logistic regression classifier builds on the insight that the a posteriori probabilities for Gaussian distributed categories is described by the logistic sigmoid function.

5.9.1 Two categories

We assume we have two categories with Gaussian probability densities $p(\mathbf{x}|c_k)$ (5.31). We then have

$$\begin{aligned} \ln\left[\frac{p(c_1|\mathbf{x})}{p(c_2|\mathbf{x})}\right] &= \ln\left[\frac{p(c_1)}{p(c_2)}\right] \\ &+ \frac{(\mathbf{x} - \mu_2)^T \mathbf{\Sigma}^{-1} (\mathbf{x} - \mu_2)}{2} \\ &- \frac{(\mathbf{x} - \mu_1)^T \mathbf{\Sigma}^{-1} (\mathbf{x} - \mu_1)}{2} \\ \Rightarrow \ln\left[\frac{p(c_1|\mathbf{x})}{p(c_2|\mathbf{x})}\right] &= \gamma + \boldsymbol{\beta}^T \mathbf{x}. \end{aligned} \tag{5.39}$$

where

$$\gamma = \ln\left[\frac{p(c_1)}{p(c_2)}\right] + \frac{\mu_2^T \mathbf{\Sigma}^{-1} \mu_2}{2} - \frac{\mu_1^T \mathbf{\Sigma}^{-1} \mu_1}{2}; \tag{5.40}$$

$$\beta = \mathbf{\Sigma}^{-1}(\mu_1 - \mu_2). \tag{5.41}$$

In the special case with only two classes we have $p(c_1|\mathbf{x}) + p(c_2|\mathbf{x}) = 1$ which is usually expressed as

$$\text{logit}[p(c_1|\mathbf{x})] = \gamma + \beta^T \mathbf{x} \tag{5.42}$$

with the definition

$$\text{logit}(z) \equiv \ln\left[\frac{z}{1-z}\right]. \tag{5.43}$$

This corresponds to

$$p(c_1|\mathbf{x}) = \frac{1}{1+\exp[-\gamma - \beta^T\mathbf{x}]} \tag{5.44}$$

which is the common logistic function.

The parameters γ and β are set by maximizing the likelihood for the data. We are here dealing with a posteriori probabilities, and the likelihood looks a little different from the regression case.

We have here an sequence of observed pairs $\mathcal{X} = \{\mathbf{x}(n), \mathbf{c}(n)\}_{n=1...N}$ and we ask what is the probability for observing this particular set of data. We assume that the observations are independent Bernoulli events. If we define our desired values as

$$y(n) = \begin{cases} 1 & \text{if } \mathbf{x}(n) \in c_1 \\ 0 & \text{if } \mathbf{x}(n) \in c_2 \end{cases} \tag{5.45}$$

then the likelihood $\mathcal{L}$ for $\mathcal{X}$ is

$$\mathcal{L} = \prod_{n=1}^{N} p(c_1|\mathbf{x}(n))^{y(n)} p(c_2|\mathbf{x}(n))^{1-y(n)}. \tag{5.46}$$

Maximizing $\mathcal{L}$ corresponds to minimizing $-\ln\mathcal{L}$, i.e. we can define our training error as

$$\begin{aligned} -\ln\mathcal{L} &= -\sum_{n=1}^{N} \{d(n)\ln p(c_1|\mathbf{x}(n)) + [1-d(n)]\ln p(c_2|\mathbf{x}(n))\} \\ &= -\sum_{n=1}^{N} \{d(n)\ln p(c_1|\mathbf{x}(n)) + [1-d(n)]\ln[1-p(c_1|\mathbf{x}(n))]\} \end{aligned} \tag{5.47}$$

In summary, logistic regression corresponds to assuming a model (the notation $\hat{p}(c_1|\mathbf{x})$ emphasizes that we are talking about an estimate of $p(c_1|\mathbf{x})$)

$$\hat{p}(c_1|\mathbf{x}) = \hat{y}(n) = \frac{1}{1+\exp[-\hat{\gamma} - \hat{\beta}^T, \mathbf{x}]} \tag{5.48}$$

and the parameters $\hat{\gamma}$ and $\hat{\beta}$ are adjusted so that

$$E = -\ln\mathcal{L} = -\sum_{n=1}^{N} \{y(n)\ln\hat{y}(n) + [1-y(n)]\ln[1-\hat{y}(n)]\} \tag{5.49}$$

is minimized. This can be achieved with, for instance, an iterative learning algorithm.

It is worth noting that even though we assume the same thing (Gaussian probability densities) for both the linear Gaussian classifier and the logistic regression, the results from using the two methods will be different (unless you have a large amount of data).

In the logistic regression, we have to find $D + 1$ parameters from N observations. In the Gaussian classifier case, we have to estimate $\frac{D(D+1)}{2} + 2D$ from N observations. The logistic regression thus has fewer free parameters than the Gaussian classifier. The logistic regresion is also more stable than the Gaussian classifier.

References

Bishop, C. M. (1995), *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford.

Haykin, S. (1999), *Neural Networks – A Comprehensive Foundation*, second edn, Prentice Hall Inc., Upper Saddle River, New Jersey.

Wetherill, G. B. (1986), *Regression Analysis with Applications*, Monographs on Statistics and Applied Probability, Chapman and Hall, London.