Chapter 3 Linear Models

This chapter aims at developing linear models for learning from data. Specifically we study linear models for classification, regression, and probability estimation. The materials presented in this chapter are primarily based on [8].

3.1 Linear Models for Classification

A. *The Model and Why It should Work*

The hypothesis set $\mathcal{H}$ for classifying data into two classes can be characterized by

$$h(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T \boldsymbol{x}) \tag{3.1}$$

where weight vector $\boldsymbol{w} \in R^{d+1}$ parameterizes the hypothesis set, and feature vector assumes the form $\boldsymbol{x} = [1 \;\; x_1 \;\; \cdots \;\; x_d]^T$. The classification problem is that given a training set $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, ..., N\}$ where $y_n \in \{1, -1\}$, find a weight vector $\boldsymbol{w}^*$ such that hypothesis $g(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^{*T} \boldsymbol{x})$ generalizes well over the input space $\mathcal{X}$.

A natural question arising here is whether the linear model in (3.1) can offer a satisfactory solution in terms of small generalization errors? To this end we recall the VC bound from Chapter 2, namely,

$$E_{\text{out}}(g) \le E_{\text{in}}(g) + \sqrt{\frac{8}{N} \ln \frac{4(2N)^{d_{vc}} + 4}{\delta}}$$

where VC dimension d_{vc} for linear model (3.1) is known to be $d + 1$, hence we obtain

$$E_{\text{out}}(g) \le E_{\text{in}}(g) + O\left(\sqrt{\frac{d}{N} \ln N} \right) \tag{3.2}$$

Although the VC bound as shown in (3.2) is not tight, it remains instructive that good generalization performance can be assured if the number of samples in the training set is sufficiently large and the in-sample error E_{in} is small.

B. *Finding w^* by the Pocket Algorithm*

Suppose we are given a training data $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, ..., N\}$ with N sufficiently large. From the analysis made above, our focus is to find a $\boldsymbol{w}^*$ that makes E_{in} small, where the error measure is given by

$$E_{\text{in}}(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} \left[\!\left[\text{sign}(\boldsymbol{w}^T \boldsymbol{x}_n) \ne y_n \right]\!\right] \tag{3.3}$$

which is essentially the percentage of misclassified samples in data set $\mathcal{D}$. To this end, the perceptron learning algorithm (PLA) and Adaline from Chapter 2 are two candidate algorithms as they produce zero E_{in} as long as the data are linearly separable. The problems is, unfortunately, PLA as well as Adaline do not converge for non-separable data sets, but the data sets in many practical machine learning problems are non-separable.

A variant of the PLA, called *pocket algorithm* (PA), runs PLA iterations but keeps only the best weight vector achieved so far in its "pocket", and produces the best weight vector obtained after a given number of iterations as the final hypothesis. In this way, the pocket algorithm is able to deal with non-separable data sets with improved performance.

The Pocket Algorithm (PCA)

Input: A training data set $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, ..., N\}$ and an initial $\boldsymbol{w}(0)$. Compute $E_{\text{in}}(\boldsymbol{w}(0))$. Set $k = 0$.

For $k = 0, 1, \ldots, K - 1$, do

Step 1: Do one PLA iteration, namely, pick one of the misclassified pairs *at random*, denote it by $(\boldsymbol{x}(k), y(k))$, and calculate

$$\tilde{\boldsymbol{w}} = \boldsymbol{w}(k) + y(k)\boldsymbol{x}(k) \tag{3.4}$$

Step 2: Evaluate $E_{in}(\tilde{\boldsymbol{w}})$.

If $E_{in}(\tilde{\boldsymbol{w}}) < E_{in}(\boldsymbol{w}(k))$, update $\boldsymbol{w}(t)$ to $\boldsymbol{w}(k+1) = \tilde{\boldsymbol{w}}$, else set $\boldsymbol{w}(k+1) = \boldsymbol{w}(k)$.

■

Example 3.1

Apply PLA and PCA to a non-separable training data set of 2000 samples known as "double semi-circle". As shown in Fig. 3.1, each semi-circle contains 1000 random samples. Each semi-circle is characterized by radius *r* and thickness *thk*. Each sample in the upper semi-circle is labeled by *y* = –1 while each sample in the lower semi-circle is labeled by *y* = 1. The center of the upper semi-circle is aligned with the middle of the edge of the lower semi-circle. If the gap between the edges of the two semi-circles is denoted by *sep*, then evidently a positive *sep* means the samples in the upper semi-circle are linearly separable from the samples in the lower semi-circle, and a negative *sep* means they are not. In this example, we are deal with non-separable training data with *r* = 10, *thk* = 5, and *sep* = –1.

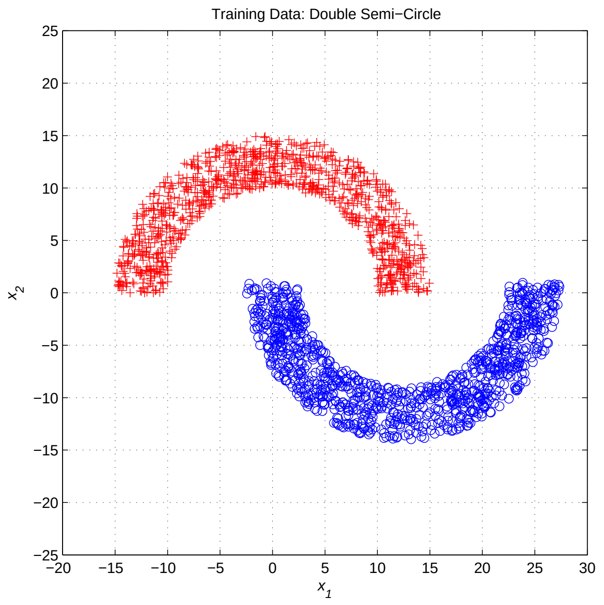


Figure 3.1 A data set of 2000 random samples in double semi-circle with *r* = 10, *thk* = 5, and *sep* = –1.

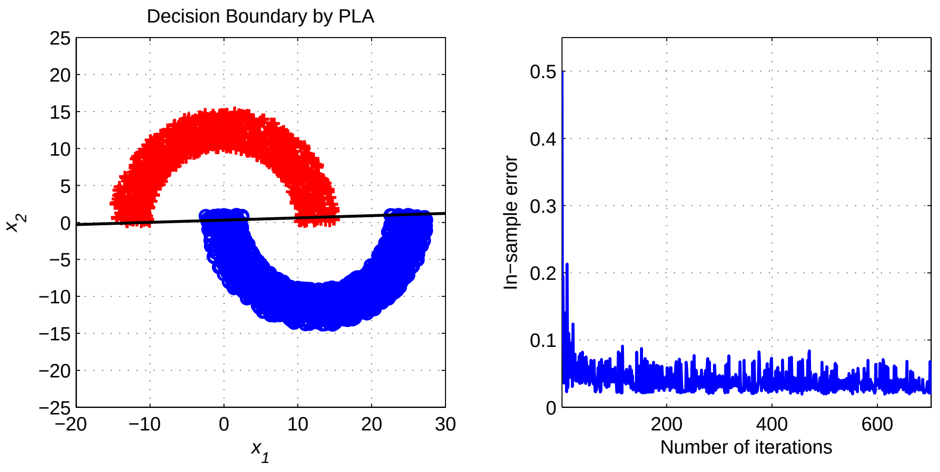
The MATLAB code producing the training data and Fig. 3.1 is listed below.

```
% Example: [x,y,xp,xn] = data_semi_circle(10,5,-1,1000,9,7);
% Written by W.-S. Lu, University of Victoria.
% Last modified: Dec. 30, 2014.
function [x,y,xp,xn] = data_semi_circle(r,thk,sep,N,st1,st2)
r1 = r;
r2 = r + thk;
ku = 0;
xn = [];
rand('state',st1)
while ku < N,
   xw = 12*randn(2,1);
   cr = 0;
   xm = norm(xw);
   if xm < r2 & xm > r1,
      cr = cr + 1;
   end
   if xw(2) > 0,
      cr = cr + 1;
   end
   if cr == 2,
      xn = [xn xw];
      ku = ku + 1;
   end
end
kd = 0;
xp = [];
rand('state',st2)
while kd < N,
   xw = 12*randn(2,1);
   cr = 0;
   xm = norm(xw);
   if xm < r2 & xm > r1,
      cr = cr + 1;
   end
   if xw(2) < 0,
      cr = cr + 1;
   end
   if cr == 2,
      xp = [xp xw];
      kd = kd + 1;
   end
end
ra = 0.5*(r1+r2);
xp(1,:) = xp(1,:) + ra;
xp(2,:) = xp(2,:) - sep;
x = [xp xn];
y = [ones(N,1); -ones(N,1)];
figure(1)
clf
plot(xp(1,:),xp(2,:),'bo')
hold on
```

```
plot(xn(1,:),xn(2,:),'r+')
grid
xlabel('\itx_1')
ylabel('\itx_2')
axis([-20 30 -25 25])
axis square
title('Training Data: Double Semi-Circle')
hold off
```

Because the training data is non-separable, the PLA does not converge. Furthermore, because the PLA is not a descent algorithm, in-sample error typically goes down and up in a randomly oscillatory fashion. With initial weight w0 = zeros(3,1), the PLA was applied to the above data set. The in-sample error after 701 iterations was found to be $E_{\text{in}} = 0.0190$. Out of $N = 2000$ sample points, 1962 points were classified correctly. The in-sample error obtained versus iteration number is shown in Fig. 3.2b. Over the first 701 iterations, the smallest in-sample error occurs at the 701th iteration which produces the weight

$$\boldsymbol{w}(701) = \begin{bmatrix} 47 & 4.4703 & -148.2111 \end{bmatrix}^T$$



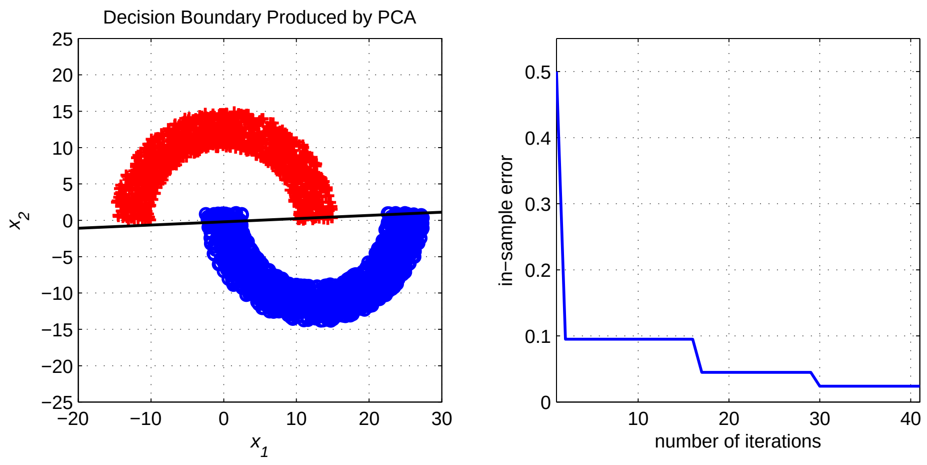
(a) (b)

Figure 3.2 PLA applied to the double semi-circle data set (a) decision boundary obtained after 701 iterations, (b) in-sample error pattern during the first 701 PLA iterations.

Next, with the same initial weight w0 = zeros(3,1), the PCA was applied to the same data set with 40 iterations. The in-sample error obtained versus iteration number is shown in Fig. 3.3b. Notice the non-decreasing in-sample error patternoffered only by PCA. The in-sample error after 40 iterations was found to be $E_{\text{in}} = 0.0240$. Out of $N = 2000$ sample points, 1952 points were classified correctly. The final weight vector was obtained as

$$\boldsymbol{w}(40) = \begin{bmatrix} -0.8 & 0.1705 & -3.8645 \end{bmatrix}^T$$

The decision boundary is shown in Fig. 3.3a.
The MATLAB code that implements the PCA for Example 3.1 is given below.



(a) (b)

Figure 3.3 PCA applied to the double semi-circle data set (a) decision boundary obtained after 40 iterations, (b) in-sample error pattern during the first 40 PCA iterations.

```
% Pocket algorithm (PCA) as applied to training data known as semi-circle.
% Example:
% [x,y,xp,xn] = data_semi_circle(10,5,-1,1000,9,7);
% [wt,t,ein_pocket] = pocket_semi_circle(x,y,xp,xn,0.1,[0 0 0]',15,26,40);
% Written by W.-S. Lu, University of Victoria. % Last modified: Jan. 25, 2015.
function [wt,t,ein_pocket] = pocket_semi_circle(x,y,xp,xn,lr,w0,st1,st2,K)
N = length(y);
rand('state',st1)
N1 = randperm(N);
x0 = x;
y0 = y;
x = x0(:,N1);
y = y0(N1);
t = 0;
w = w0(:);
w_pocket = w;
D = [ones(N,1) x'];
y = y(:);
dwt = (D*w >= 0);
acc = sum(y == 2*dwt - 1);
acc_pocket(1) = acc;
rand('state',st2)
N1 = randperm(N);
while t < K & acc_pocket(t+1) < N,
    ii = mod(t,N) + 1;
    i = N1(ii);
    fi = 2*dwt(i) - 1;
    w = w + lr*(y(i)-fi)*D(i,:)';
    dwt = (D*w >= 0);
    acc = sum(y == 2*dwt - 1);
    if acc > acc_pocket(t+1),
       w_pocket = w;
```

```
        acc_pocket(t+2) = acc;
    else
        acc_pocket(t+2) = acc_pocket(t+1);
    end
    t = t + 1;
end
disp(sprintf('Solution reached in %d iterations.', t));
disp('Final weights:')
wt = w_pocket
ent = N - acc_pocket(end);
Ein = ent/N;
disp(sprintf('In-sample error was found to be %d.', Ein));
disp(sprintf('Out of N = %d sample points,',N));
disp(sprintf('%d points were classified correctly',N - ent));
figure(1)
subplot(121)
plot(xp(1,:),xp(2,:),'bo','linewidth',1.5)
hold on
plot(xn(1,:),xn(2,:),'r+','linewidth',1.5)
grid
xlabel('\itx_1')
ylabel('\itx_2')
p1 = -20;
p2 = (-wt(2)*p1-wt(1))/wt(3);
q1 = 30;
q2 = (-wt(2)*q1-wt(1))/wt(3);
plot([p1 q1],[p2 q2],'k-','linewidth',1.5)
axis([-20 30 -25 25])
axis square
title('Decision Boundary Produced by PCA')
hold off
subplot(122)
ein_pocket = (N - acc_pocket)/N;
plot(1:1:t+1,ein_pocket,'b-','linewidth',1.5)
axis square
axis([1 t+1 0 1.1*ein_pocket(1)])
xlabel('number of iterations')
ylabel('in-sample error')
grid
```
■

3.2 Linear Models for Regression

A. ***Linear Regression***

According to investopedia, "regression is a statistical measure that attempts to determine the strength of the relationship between one dependent variable (usually denoted by *Y*) and a series of other changing variables (known as independent variables)". In the context of machine learning, a regression model relates to a family hypotheses where each hypothesis is a function that depends on several input features $\{x_i, i = 1, \ldots, d\}$ and assumes *real-value* outputs. As such, a hypothesis in a regression model is a function that maps input $\boldsymbol{x}$ to real-valued y that are not necessarily integers. It follows that a *linear* regression simply refers to a *linear* function of the form

$$h(\boldsymbol{x}) = \sum_{i=0}^{d} w_i x_i = \boldsymbol{w}^T \boldsymbol{x} \tag{3.5}$$

where $\boldsymbol{x} = \begin{bmatrix} 1 & x_1 & \cdots & x_d \end{bmatrix}^T \in R^{d+1}$. In a linear regression problem, one is given training data $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, ..., N\}$ where y_n are real numbers and seeks to find an appropriate weight $\boldsymbol{w}^*$ for $h(\boldsymbol{x})$ in (3.5) so that $h(\boldsymbol{x})$ well predicts data outside $\mathcal{D}$.
For regression models a natural choice of the error measure, also known as *loss function*, is the so-called L_2-*loss* given by

$$E_{\text{in}}(h) = \frac{1}{N} \sum_{n=1}^{N} \left(h(\boldsymbol{x}_n) - y_n \right)^2 \tag{3.6}$$

An error measure for out-of-sample error that is consisting with (3.6) is given by

$$E_{\text{out}} = E\left[\left(h(\boldsymbol{x}) - y \right)^2 \right] \tag{3.7}$$

where the expectation is taken with respect to the joint probability distribution $P(\boldsymbol{x}, y)$ which is obviously unknown in a practical learning problem. A realistic way to evaluate E_{out} is to use an approximate measure such as

$$\hat{E}_{\text{out}}(h) = \frac{1}{M} \sum_{m=1}^{M} \left(h(\boldsymbol{x}_m) - y_m \right)^2 \tag{3.8}$$

that is evaluated over *testing* data set $\mathcal{T} = \{(\boldsymbol{x}_m, y_m), m = 1, 2, ..., M\}$ which is drawn independently randomly outside $\mathcal{D}$.

B. *A Closed-Form Solution*

Since the hypothesis $h(\boldsymbol{x})$ (see (3.5)) depends on weight $\boldsymbol{w}$, the in-sample error in (3.6) is a function of $\boldsymbol{w}$. Combining (3.6) with (3.5), we can write

$$E_{\text{in}}(h) = \frac{1}{N} \sum_{n=1}^{N} \left(h(\boldsymbol{x}_n) - y_n \right)^2 = \frac{1}{N} \sum_{n=1}^{N} \left(\boldsymbol{w}^T \boldsymbol{x}_n - y_n \right)^2 = \frac{1}{N} \left\| \boldsymbol{X}\boldsymbol{w} - \boldsymbol{y} \right\|^2 \tag{3.9}$$

where

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_N^T \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

The expression of E_{in} in (3.9) shows that it is a convex quadratic function of $\boldsymbol{w}$. Consequently, the $\boldsymbol{w}^*$ that minimizes E_{in} is the one that nullifies the gradient of E_{in}. Computing the gradient and set it to zero, we obtain

$$\nabla_w \left(E_{\text{in}}(h) \right) = \frac{2}{N} \left(\boldsymbol{X}^T \boldsymbol{X} \boldsymbol{w} - \boldsymbol{X}^T \boldsymbol{y} \right) = \boldsymbol{0}$$

which yields a solution

$$\boldsymbol{w}^* = \underbrace{\left(\boldsymbol{X}^T \boldsymbol{X} \right)^{-1} \boldsymbol{X}^T}_{\boldsymbol{X}^\dagger} \boldsymbol{y} \triangleq \boldsymbol{X}^\dagger \boldsymbol{y} \tag{3.10}$$

where $X^{\dagger} = \left(X^T X\right)^{-1} X^T$ is the pseudo-inverse of X. With the optimal weight given by (3.10), the linear regression function maps the training data X to

$$\hat{y} = Xw^* = \underbrace{X\left(X^T X\right)^{-1} X^T}_{H} y \triangleq Hy$$

where $H = X\left(X^T X\right)^{-1} X^T$ is called "hat matrix" [8]. It follows that the above prediction differs from the training data y by $y - \hat{y} = (I - H)y$, hence the minimized in-sample error becomes

$$E_{\text{in}} = \frac{1}{N}\left\|(I - H)y\right\|^2 \tag{3.11}$$

We remark that linear regression also applies to binary classification problems. In such cases the training data assumes the form $\mathcal{D} = \{(x_n, y_n), n = 1, 2, ..., N\}$ where $y_n \in \{1, -1\}$.

Example 3.2

Apply linear regression to the classification problem in Example 3.1.

Solution

For the problem in Example 3.1, we have $d = 2$ and $N = 2000$. In this case X is a matrix of size 2000 by 3, hence the matrix $X^T X$ that needs to be inverted is a 3 by 3 positive definite matrix. Applying the closed formula in (3.10) yields

$$w^* = \begin{bmatrix} -0.0598 & 0.0162 & -0.0938 \end{bmatrix}^T$$

As is shown in Fig. 3.4, the decision boundary determined by w^* is a reasonable one, but not as good as those obtained by PLA and PCA. As a matter of fact, the in-sample error measured by the percentage of misclassified pairs was found to be 0.0415. Out of $N = 2000$ sample points, 1917 points were classified correctly. For comparison we recall that the PLA yields a considerably lower in-sample error as 0.0190.

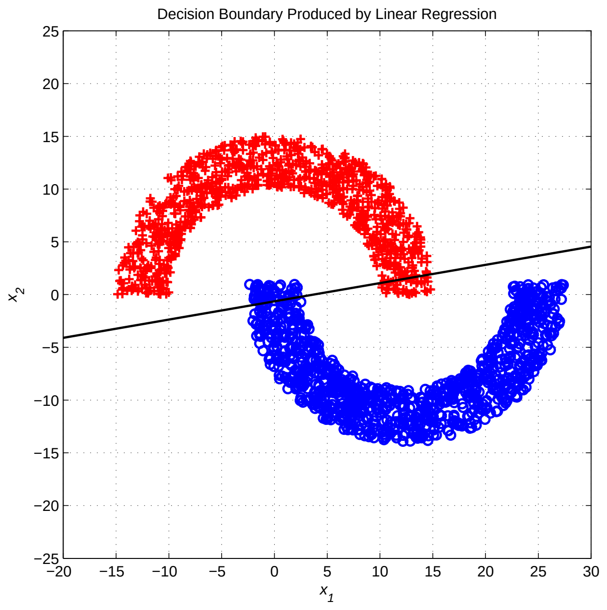


Figure 3.4 Decision boundary obtained by linear regression.

The MATLAB code that implements the linear regression algorithm for Example 3.2 is given below.

```
% Linear regression as applied to training data known as semi-circle.
% Example:
% [x,y,xp,xn] = data_semi_circle(10,5,-1,1000,9,7);
% [wt,Ein1,Ein2] = regression_semi_circle(x,y,xp,xn,5);
% Written by W.-S. Lu, University of Victoria. Last modified: Jan. 25, 2015.
function [wt,Ein1,Ein2] = regression_semi_circle(x,y,xp,xn,st)
N = length(y);
rand('state',st)
N1 = randperm(N);
x0 = x;
y0 = y;
x = x0(:,N1);
y = y0(N1);
X = [ones(N,1) x'];
y = y(:);
wt = (inv(X'*X))*(X'*y);
disp('Final weights:')
wt
Ein1 = norm(X*wt-y)^2/N;
disp(sprintf('In-sample error was found to be %d.', Ein1));
dwt = (X*wt >= 0);
z = dwt + dwt - y - 1;
L = sum(abs(z))/2;
Ein2 = L/N;
disp(sprintf('Out of a total of %d samples,',N));
disp(sprintf('%d points were classified correctly.',N-L));
figure(1)
plot(xp(1,:),xp(2,:),'bo','linewidth',1.5)
hold on
plot(xn(1,:),xn(2,:),'r+','linewidth',1.5)
grid
xlabel('\itx_1')
ylabel('\itx_2')
p1 = -20;
p2 = (-wt(2)*p1-wt(1))/wt(3);
q1 = 30;
q2 = (-wt(2)*q1-wt(1))/wt(3);
plot([p1 q1],[p2 q2],'k-','linewidth',1.5)
axis([-20 30 -25 25])
axis square
title('Decision Boundary Produced by Linear Regression')
hold off      ■
```

C. ***Computing w^* by Numerical Optimization***

Formula (3.10) is compact and convenient to use, and there are stable numerical methods to compute $\boldsymbol{X}^\dagger$ [12]. Nevertheless, the complexity of (3.10) grows rapidly as the dimension d of the input space and/or the size N of the training data increases.

An alternative approach to computing $\boldsymbol{w}^*$ (or an approximate of it) is to use numerical optimization methods that run inexpensive iterations to reach the solution. Candidate methods for this purpose include the steepest descent (also called gradient descent) method (SDM) where the input features are adequately scaled, and quasi-Newton algorithms such as DFP and BFGS algorithms. The key quantity required in these methods is the gradient of the loss function, which is given by

$$\nabla_w \left(E_{\text{in}}(h)\right) = \frac{2}{N} \boldsymbol{X}^T \left(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\right) \tag{3.12}$$

The algorithmic details of the SDM and BFGS algorithms can be found in Sec. 1.2. It is worth noting that because the loss function is convex these gradient-based algorithms are guaranteed to converge to the global minimizer regardless of where the algorithm is initiated.

3.3 Logistic Regression

A. ***Logistic Regression for Predicting a Probability***

Mathematically, a logistic regression function falls in between the binary-decision hypothesis $h(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T\boldsymbol{x})$ and continuous and unconstrained linear regression hypothesis $h(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}$: like the linear classification and linear regression functions, a logistic regression function combines the components of the feature vector $\boldsymbol{x}$ linearly via a weight vector $\boldsymbol{w}$ as $\boldsymbol{w}^T\boldsymbol{x}$, on the other hand, unlike these functions, the logistic regression function produces an output between 0 and 1. Physically and more importantly, because its output is constrained in [0, 1], logistic regression fits well into problems where one seeks to predict a *probability*. Specifically, in logistic regression one is given an i.i.d. training data $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, N\}$ where $y_n \in \{1, -1\}$ representing an event of interest (heart attack for example) occurs ($y = 1$) or does not occur ($y = -1$) and the goal is to identify a logistic regression function $h(\boldsymbol{x})$ that can be used to predict how probable a heart attack will occur to a patient whose features are given in terms of input vector $\boldsymbol{x}$.

The reason behind the name "logistic regression" is the use of the logistic function $\theta(z)$ to map the linear combination $z = \boldsymbol{w}^T\boldsymbol{x}$ to a value between 0 and 1:

$$h(\boldsymbol{x}) = \theta(\boldsymbol{w}^T\boldsymbol{x}) \tag{3.13a}$$

where

$$\theta(z) = \frac{e^z}{1 + e^z} \tag{3.13b}$$

is called *logistic function* in literature, see Fig. 3.4 for an illustration $\theta(z)$ which smoothly and monotonically increases from 0 to 1 with $\theta(0) = 0.5$.

By combining (3.13a) with (3.13b), we obtain the logistic regression hypothesis

$$h(\boldsymbol{x}) = \frac{e^{\boldsymbol{w}^T\boldsymbol{x}}}{1 + e^{\boldsymbol{w}^T\boldsymbol{x}}} \tag{3.14}$$

For problems where the objective is to predict a probability rather than a deterministic answer, the target is not a function but a conditional probability distribution $P(y \mid \boldsymbol{x})$. In this situation, logistic regression can be employed to define a likelihood-function based error measure (see Sec. 2.5) as follows. First, one may relate $P(y \mid \boldsymbol{x})$ to the logistic regression function in (3.14) as

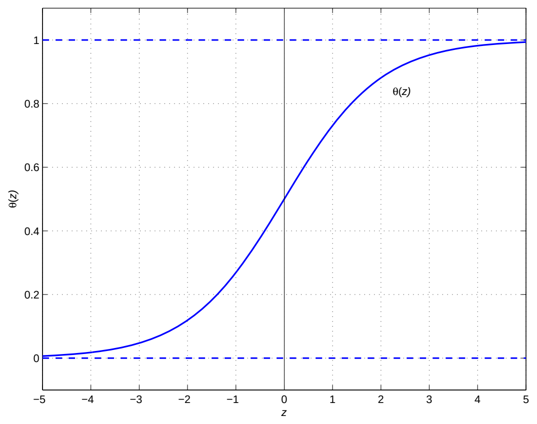


Figure 3.4 the logistic function $\theta(z) = e^z/(1+e^z)$.

$$P(y \mid \boldsymbol{x}) = \begin{cases} h(\boldsymbol{x}) & \text{for } y = 1 \\ 1 - h(\boldsymbol{x}) & \text{for } y = -1 \end{cases} \tag{3.15}$$

Using (3.13b), (3.14) and (3.15), it can readily be verified that (3.15) can be written compactly as

$$P(y \mid \boldsymbol{x}) = \theta(y \cdot \boldsymbol{w}^T \boldsymbol{x}) \tag{3.16}$$

Next, to find a weight $\boldsymbol{w}^*$ for $h(\boldsymbol{x}, \boldsymbol{w}^*)$ to best fit an i.i.d. data set $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, ..., N\}$, we try to maximize the conditional probability of getting all the y_n's in the data set given the corresponding x_n's where the conditional probability is modeled by the logistic model in (3.16). Because the data are i.i.d., we have

$$P(y_1, \cdots, y_N \mid \boldsymbol{x}_1, \cdots, \boldsymbol{x}_N) = \prod_{n=1}^{N} P(y_n \mid \boldsymbol{x}_n) = \prod_{n=1}^{N} \theta(y_n \boldsymbol{w}^T \boldsymbol{x}_n)$$

whose maximization is equivalent to minimizing its scaled negative logarithm, namely

$$-\frac{1}{N} \ln\left(\prod_{n=1}^{N} \theta(y_n \boldsymbol{w}^T \boldsymbol{x}_n) \right) = \frac{1}{N} \sum_{n=1}^{N} \ln\left(\frac{1}{\theta(y_n \boldsymbol{w}^T \boldsymbol{x}_n)} \right) = \frac{1}{N} \sum_{n=1}^{N} \ln\left(1 + e^{-y_n \boldsymbol{w}^T \boldsymbol{x}_n}\right)$$

The analysis above suggests the logistic loss function as the in-sample error measure:

$$E_{\text{in}}(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} \ln\left(1 + e^{-y_n \boldsymbol{w}^T \boldsymbol{x}_n}\right) \tag{3.17}$$

B. ***<u>Finding w* by Numerical Optimization</u>***

The gradient and Hessian of $E_{\text{in}}(\boldsymbol{w})$ are given by

$$\nabla\left(E_{\text{in}}(\boldsymbol{w})\right) = -\frac{1}{N} \sum_{n=1}^{N} \frac{y_n \boldsymbol{x}_n}{1 + e^{y_n \boldsymbol{w}^T \boldsymbol{x}_n}} \tag{3.18}$$

and

$$\nabla^2\left(E_{\text{in}}(\boldsymbol{w})\right) = \frac{1}{N} \sum_{n=1}^{N} \frac{\boldsymbol{x}_n \boldsymbol{x}_n^T}{(1 + e^{y_n \boldsymbol{w}^T \boldsymbol{x}_n})^2} \tag{3.19}$$

respectively. Because the Hessian is unconditionally positive semidefinite, $E_{\text{in}}(\boldsymbol{w})$ is global convex. Therefore, gradient-based optimization methods equipped with an adequate line search step (e.g. the back-tracking line search technique, see Sec. 1.2) shall converge to the global solution regardless of the initial point used. The SDM with adequately scaled input features and BFGS algorithm are good candidate optimization methods for minimizing $E_{\text{in}}(\boldsymbol{w})$ in (3.19). The Newton method might also be a good choice for problems where both d and N are moderate.

Like the linear regression, we remark that logistic regression also applies to binary classification problems. In such cases, the probability produced by logistic regression can be used to label an input feature vector with $y = 1$ if the probability exceeds 0.5, which from (3.14) corresponds to $\boldsymbol{w}^T\boldsymbol{x} > 0$, and with $y = -1$ which corresponds to $\boldsymbol{w}^T\boldsymbol{x} < 0$ otherwise. It follows that when applying to binary classification problems logistic regression offers a linear decision boundary, namely $\boldsymbol{w}^T\boldsymbol{x} = 0$, see Example 3.3 below.

It is also interesting to note that logistic regression can also be applied to multi-class classification problem. As described in [13], a k-class classification problem can be solved as follows. Given an $\boldsymbol{x}$ in the input space and we want to classify it into one of a total of k classes. First, we solve k *binary* classification problems, each singles out one (say the ith) of the k classes as "the class" and the union of the rest classes as "the other class". Denote the optimal weight obtained by $\boldsymbol{w}_{(i)}^*$. Next, with the given input $\boldsymbol{x}$ we compare the values $\{\boldsymbol{w}_{(i)}^{*T}\boldsymbol{x}, i = 1,\ldots,k\}$ and identify the index i^* corresponding the maximum value $\boldsymbol{w}_{(i^*)}^{*T}\boldsymbol{x}$ as the class to which $\boldsymbol{x}$ belongs. Note that in principle we are supposed to compute and compare the values of the logistic regression functions obtained as probability estimates. Since the logistic regression is monotonic, It suffices to evaluate and compare the values of $\{\boldsymbol{w}_{(i)}^{*T}\boldsymbol{x}, i = 1,\ldots,k\}$.

<u>Example 3.3</u>

Apply logistic regression to the classification problem in Example 3.1.

Solution

The BFGS with backtracking (BT) line search was chosen to minimize the loss function in (3.17). With initial point $\boldsymbol{w}(0) = \boldsymbol{0}$ and it took 20 BFGS iterations to converge to the optimal weight

$$\boldsymbol{w}^* = \begin{bmatrix} 0.8078 & 0.0813 & -2.8802 \end{bmatrix}^T$$

which can be used to classify data $\boldsymbol{x}$ to one of two classes as

$$\hat{y} = \begin{cases} 1 & \text{if } h(\boldsymbol{x}) = \dfrac{e^{\boldsymbol{w}^{*T}\boldsymbol{x}}}{1 + e^{\boldsymbol{w}^{*T}\boldsymbol{x}}} > \dfrac{1}{2} \\ -1 & \text{if } h(\boldsymbol{x}) = \dfrac{e^{\boldsymbol{w}^{*T}\boldsymbol{x}}}{1 + e^{\boldsymbol{w}^{*T}\boldsymbol{x}}} < \dfrac{1}{2} \end{cases}$$

where function $h(\boldsymbol{x})$ is from (3.14) and is interpreted as predicting probability with 0.5 as the threshold. Out of $N = 2000$ sample points, 1961 points were classified correctly.The decision boundary is therefore determined by the equation

$$h(\boldsymbol{x}) = \frac{e^{\boldsymbol{w}^{*T}\boldsymbol{x}}}{1 + e^{\boldsymbol{w}^{*T}\boldsymbol{x}}} = \frac{1}{2} \tag{3.20}$$

Evidently, an $\boldsymbol{x}$ satisfies (3.20) if and only if

$$\boldsymbol{w}^{*T}\boldsymbol{x} = 0 \tag{3.21}$$

Hence, like the PLA, PCA, and linear regression, the decision boundary of logistic-regression based classification is also a straight line. With the optimal weight $\boldsymbol{w}^*$, the decision boundary is depicted in Fig. 3.5 as a black line. For comparison purposes, the final in-sample error in terms of the percentage of misclassified data was evaluated and found to be 0.0195 (i.e. 1.95%), a performance very close to that of the PLA and considerably better than that of linear regression.

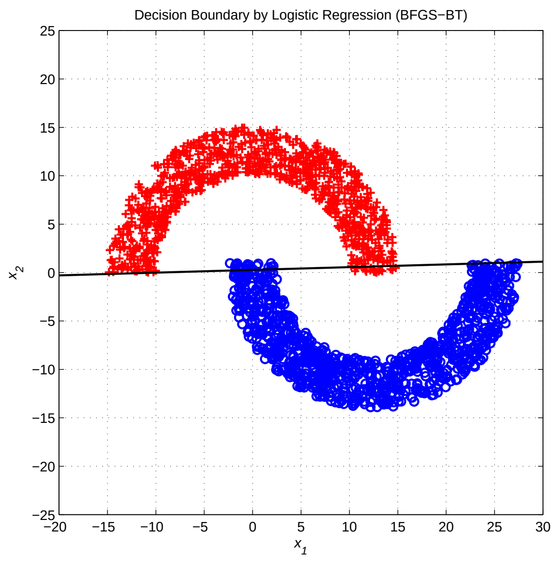


Figure 3.5 Decision boundary by logistic regression (with BFGS-BT).

Below is the MATLAB code that implements the logistic regression with BFGS-BT as applied to the double semi-circle data and produces Fig. 3.5.

```
% To apply logistic regression to classify double nonseparable demi-circle
% data. BFGS algorithm is chosen to minimize the logistic loss function.
% Written by W.-S. Lu, University of Victoria. Last modified: Jan. 25, 2015.
% Example:
% [x,y,xp,xn] = data_semi_circle(10,5,-1,1000,9,7);
% [wt,fs] = logistic_BFGS_K(x,y,xp,xn,'logistic_f','logistic_g',zeros(3,1),17,20);
function [wt,fs] = logistic_BFGS_K(x,y,xp,xn,fname,gname,w0,st,K)
N = length(y);
rand('state',st)
N1 = randperm(N);
x0 = x;
y0 = y;
x = x0(:,N1);
y = y0(N1);
d1 = length(w0);
p = [N; d1; y; x(:)];
I = eye(d1);
k = 1;
wk = w0;
Sk = I;
gk = feval(gname,wk,p);
dk = -Sk*gk;
```

```
ak = bt_lsearch(wk,dk,fname,gname,p);
dtk = ak*dk;
wk_new = wk + dtk;
while K > k,
    gk_new = feval(gname,wk_new,p);
    gmk = gk_new - gk;
    D = dtk'*gmk;
    if D <= 0,
      Sk = I;
    else
      sg = Sk*gmk;
      sw0 = (1+(gmk'*sg)/D)/D;
      sw1 = dtk*dtk';
      sw2 = sg*dtk';
      Sk = Sk + sw0*sw1 - (sw2'+sw2)/D;
    end
    gk = gk_new;
    wk = wk_new;
    dk = -Sk*gk;
    ak = bt_lsearch(wk,dk,fname,gname,p);
    dtk = ak*dk;
    wk_new = wk + dtk;
    k = k + 1;
end
disp('Final weights:')
wt = wk_new
fs = feval(fname,wt,p);
disp(sprintf('Objective function at solution point: %d.', fs));
Dt = [ones(N,1) x'];
y = y(:);
dwt = (Dt*wt >= 0);
z = dwt + dwt - y - 1;
L = sum(abs(z))/2;
disp(sprintf('Out of N = %d sample points,',N));
disp(sprintf('%d points were classified correctly.',N-L));
figure(1)
plot(xp(1,:),xp(2,:),'bo','linewidth',1.5)
hold on
plot(xn(1,:),xn(2,:),'r+','linewidth',1.5)
grid
xlabel('\itx_1')
ylabel('\itx_2')
p1 = -20;
p2 = (-wt(2)*p1-wt(1))/wt(3);
q1 = 30;
q2 = (-wt(2)*q1-wt(1))/wt(3);
plot([p1 q1],[p2 q2],'k-','linewidth',1.5)
axis([-20 30 -25 25])
axis square
title('Decision Boundary by Logistic Regression (BFGS-BT)')
hold off
```

```
=========
function z = logistic_f(w,p)
N = p(1);
d = p(2) - 1;
y = p(3:N+2);
v = p((N+3):end);
x = vec2mat(v,d)';
D = [ones(1,N); x];
z = 0;
for i = 1:N,
   z = z + log(1 + exp(-y(i)*(w'*D(:,i))));
end
z = z/N;
=========
function z = logistic_g(w,p)
N = p(1);
d = p(2) - 1;
y = p(3:N+2);
v = p((N+3):end);
x = vec2mat(v,d)';
D = [ones(1,N); x];
z = 0;
for i = 1:N,
   ti = y(i)*D(:,i);
   z = z + ti/(1 + exp(w'*ti));
end
z = -z/N;
```

For comparisons the SDM with BT line search was also used for minimizing the error measure in (3.17). As expected, SDM was slow as it took 200 iterations to converge (although the SDM iterations are slightly less complex than BFGS iterations). The algorithm yields an optimal weight

$$\boldsymbol{w}^* = \begin{bmatrix} 0.2997 & 0.0860 & -2.0634 \end{bmatrix}^T$$

which defines a decision boundary $\boldsymbol{w}^{*T}\boldsymbol{x} = 0$ shown in Fig. 3.6. The final in-sample error in terms of the percentage of misclassified data was found to be 0.0195 (i.e. 1.95%), the same performance as that of the logistic regression with BFGS-BT.

Below is the MATLAB code that implements the logistic regression with SDM-BT as applied to the double semi-circle data and produces Fig. 3.6:

```
% To apply logistic regression to classify double nonseparable demi-circle
% data. The SDM algorithm with BT is chosen to minimize the logistic loss function.
% Written by W.-S. Lu, University of Victoria.
% Last modified: Jan. 25, 2015.
% Example:
% [x,y,xp,xn] = data_semi_circle(10,5,-1,1000,9,7);
% [wt,fk] = logistic_SDM_K(x,y,xp,xn,'logistic_f','logistic_g',zeros(3,1),17,200);
function [wt,fk] = logistic_SDM_K(x,y,xp,xn,fname,gname,w0,st,K)
N = length(y);
```

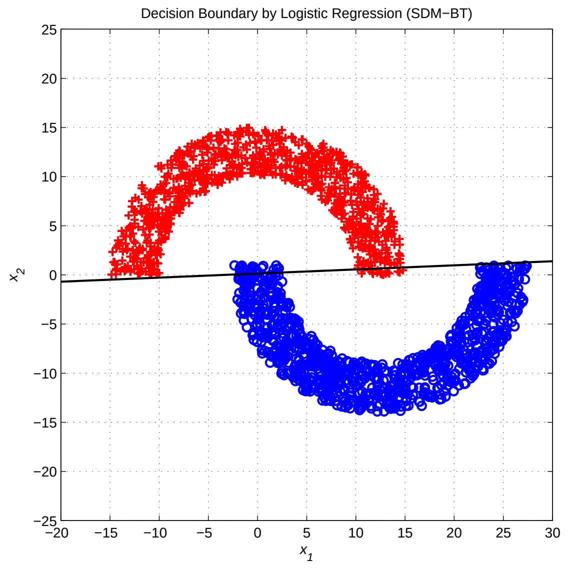


Figure 3.6 Decision boundary by logistic regression (with SDM-BT).

```
rand('state',st)
N1 = randperm(N);
x0 = x;
y0 = y;
x = x0(:,N1);
y = y0(N1);
d1 = length(w0);
p = [N; d1; y; x(:)];
k = 0;
wk = w0;
fk(1) = feval(fname,wk,p);
while k < K,
     gk = feval(gname,wk,p);
     dk = -gk;
     ak = bt_lsearch(wk,dk,fname,gname,p);
     dtk = ak*dk;
     wk = wk + dtk;
     k = k + 1;
     fk(k+1) = feval(fname,wk,p);
end
disp('Final weights:')
wt = wk
fs = feval(fname,wt,p);
disp(sprintf('Objective function at solution point: %d.', fs));
Dt = [ones(N,1) x'];
y = y(:);
dwt = (Dt*wt >= 0);
z = dwt + dwt - y - 1;
L = sum(abs(z))/2;
disp(sprintf('Out of N = %d sample points,',N));
disp(sprintf('%d points were classified correctly.',N-L));
```

```
figure(1)
plot(xp(1,:),xp(2,:),'bo','linewidth',1.5)
hold on
plot(xn(1,:),xn(2,:),'r+','linewidth',1.5)
grid
xlabel('\itx_1')
ylabel('\itx_2')
p1 = -20;
p2 = (-wt(2)*p1-wt(1))/wt(3);
q1 = 30;
q2 = (-wt(2)*q1-wt(1))/wt(3);
plot([p1 q1],[p2 q2],'k-','linewidth',1.5)
axis([-20 30 -25 25])
axis square
title('Decision Boundary by Logistic Regression (SDM-BT)')
hold off
```

■

C. *Minimizing $E_{\text{in}}(\boldsymbol{w})$ Using Stochastic Gradient Descent (SGD)*

When the size of training data N is large, computing the gradient $\nabla\left(E_{\text{in}}(\boldsymbol{w})\right)$ as seen in (3.18) is expensive. If we write the gradient as

$$\nabla\left(E_{\text{in}}(\boldsymbol{w})\right) = \frac{1}{N}\sum_{n=1}^{N}\nabla\left(\ln\left(1+e^{-y_n\boldsymbol{w}^T\boldsymbol{x}_n}\right)\right) = \frac{1}{N}\sum_{n=1}^{N}\frac{-y_n\boldsymbol{x}_n}{1+e^{y_n\boldsymbol{w}^T\boldsymbol{x}_n}} \tag{3.22}$$

we see that it is the arithmetic average of the gradients of individual error terms (at individual data points). The method of *stochastic gradient decent* (SGD) picks a training data point ($\boldsymbol{x}_n$, y_n) uniformly *at random* and computes the gradient of the error term at ($\boldsymbol{x}_n$, y_n), and use it to replace $\nabla\left(E_{\text{in}}(\boldsymbol{w})\right)$ in the SDM. From (3.22), the gradient of the nth error term, denoted by $e_n(\boldsymbol{w})$, is given by

$$\nabla e_n(\boldsymbol{w}) = \frac{-y_n\boldsymbol{x}_n}{1+e^{y_n\boldsymbol{w}^T\boldsymbol{x}_n}}$$

Obviously, the complexity of computing $\nabla e_n(\boldsymbol{w})$ is N times less than that of computing $\nabla\left(E_{\text{in}}(\boldsymbol{w})\right)$, a big savings when data size N is large. Further complexity reduction is achieved by using a constant step size $\alpha > 0$ when the SGD updates weight $\boldsymbol{w}(k)$ to $\boldsymbol{w}(k+1)$:

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) + \frac{\alpha y_n\boldsymbol{x}_n}{1+e^{y_n\boldsymbol{w}(k)^T\boldsymbol{x}_n}} \tag{3.23}$$

The success of SGD has to do with independently randomly picking an individual data point ($\boldsymbol{x}_n$, y_n) in each iteration. There are a great deal of research for convergence analysis, performance evaluation, and development of improved SGD algorithms, see for example [14], [15] and the references therein.

Example 3.4

The SGD was applied to minimize the loss function in (3.17). With initial point $\boldsymbol{w}(0) = \boldsymbol{0}$ and an fixed $\alpha = 0.005$, it took 12,000 iterations to converge to an optimal weight

$$\boldsymbol{w}^* = \begin{bmatrix}0.0916 & 0.0877 & -1.7096\end{bmatrix}^T$$

which defines a decision boundary $\boldsymbol{w}^{*T}\boldsymbol{x} = 0$ shown in Fig. 3.7a. The final in-sample error in terms of the percentage of misclassified data was found to be 0.0210 (i.e. 2.1%), a decent performance on comparing with other methods tested.

The performance of SGD may improve by using a more adequate initial weight. For instance, with a randomly generated

$$\boldsymbol{w}(0) \;=\; [1.030112096647569 \quad 0.703751395701793 \quad -1.646151244456372]^T$$

and a fixed $\alpha = 0.005$, after 12,000 iterations the SGD yields a weight vector

$$\boldsymbol{w}^* = \begin{bmatrix}0.4966 & 0.0964 & -2.3410\end{bmatrix}^T$$

The associated decision boundary and a profile of in-sample error versus SGD iterations are shown in Fig, 3.8a and 3.8b, respectively. The final in-sample error in terms of the percentage of misclassified data was found to be 0.0190 (i.e. 1.9%), one of the best performance among the methods tested.

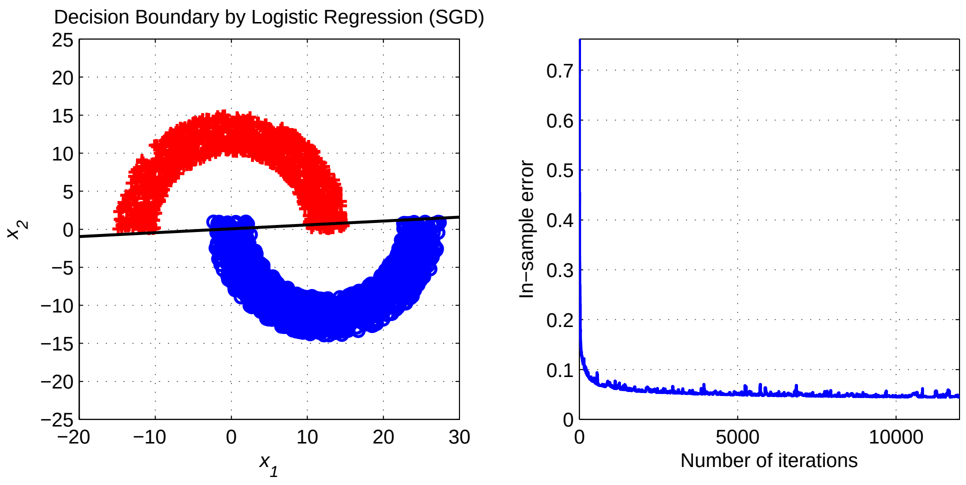


Figure 3.7 (a) Decision boundary by logistic regression (with SGD and $\boldsymbol{w}(0) = \boldsymbol{0}$) and (b) profile of in-sample error versus iterations.

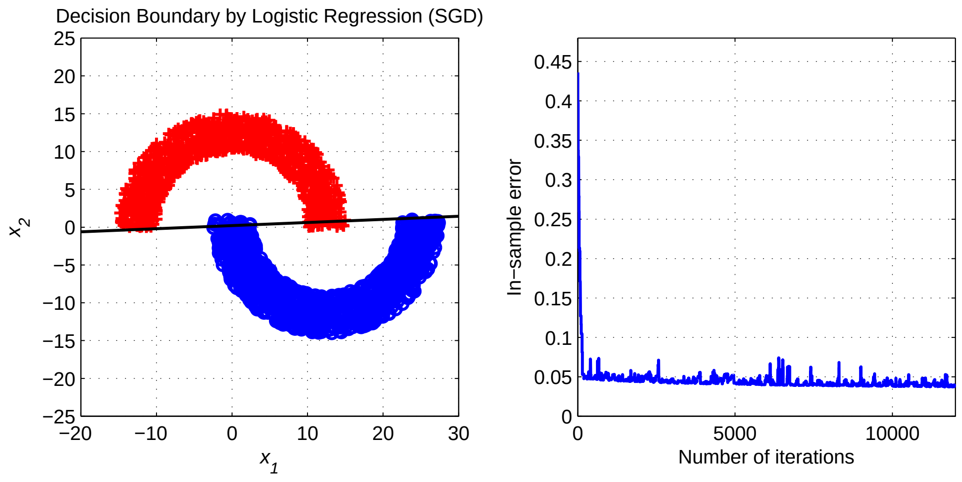


Figure 3.8 (a) Decision boundary by logistic regression (with SGD and $\boldsymbol{w}(0)$ = randn(3,1)) and (b) profile of in-sample error versus iterations.

Given below is the MATLAB code that implements the logistic regression with SGD as applied to the double semi-circle data and produces Figs. 3.7 and 3.8:

```
% To apply logistic regression to classify double nonseparable demi-circle
% data. The SGD algorithm is chosen to minimize the logistic loss function.
% Written by W.-S. Lu, University of Victoria. Last modified: Jan. 25, 2015.
% Examples:
% [x,y,xp,xn] = data_semi_circle(10,5,-1,1000,9,7);
% [wt,fk] = logistic_SGD_K(x,y,xp,xn,'logistic_f',zeros(3,1),10,5,16,12000);
% A good initial point: w0 = [1.030112096647569   0.703751395701793 -1.646151244456372]';
% [wt,fk] = logistic_SGD_K(x,y,xp,xn,'logistic_f',w0,0.005,5,16,12000);
function [wt,fk] = logistic_SGD_K(x,y,xp,xn,fname,w0,a,st1,st2,K)
N = length(y);
rand('state',st1)
N1 = randperm(N);
x0 = x;
y0 = y;
x = x0(:,N1);
y = y0(N1);
d1 = length(w0);
p = [N; d1; y; x(:)];
D = [ones(1,N); x];
k = 0;
wk = w0;
fk(1) = feval(fname,wk,p);
rand('state',st2)
while k < K,
    Nw = randperm(N);
    i = Nw(1);
    ti = y(i)*D(:,i);
    dk = ti/(1+exp(wk'*ti));
    wk = wk + a*dk;
    k = k + 1;
    fk(k+1) = feval(fname,wk,p);
end
disp('Final weights:')
wt = wk
fs = feval(fname,wt,p);
disp(sprintf('Objective function at solution point: %d.', fs));
Dt = [ones(N,1) x'];
y = y(:);
dwt = (Dt*wt >= 0);
z = dwt + dwt - y - 1;
L = sum(abs(z))/2;
disp(sprintf('Out of N = %d sample points,',N));
disp(sprintf('%d points were classified correctly.',N-L));
figure(1)
subplot(121)
plot(xp(1,:),xp(2,:),'bo','linewidth',1.5)
hold on
plot(xn(1,:),xn(2,:),'r+','linewidth',1.5)
```

```
grid
xlabel('\itx_1')
ylabel('\itx_2')
p1 = -20;
p2 = (-wt(2)*p1-wt(1))/wt(3);
q1 = 30;
q2 = (-wt(2)*q1-wt(1))/wt(3);
plot([p1 q1],[p2 q2],'k-','linewidth',1.5)
axis([-20 30 -25 25])
axis square
title('Decision Boundary by Logistic Regression (SGD)')
hold off
subplot(122)
plot(1:1:K+1,fk,'b-','linewidth',1.5)
xlabel('Number of iterations')
ylabel('In-sample error')
grid
axis square
axis([0 K+1 0 1.1*fk(1)])
```
■

3.4 Nonlinear Transforms

A common thread among the linear models studied so far in this chapter is the use of the inner product $\boldsymbol{w}^T\boldsymbol{x}$ where both the hypothesis parameter $\boldsymbol{w}$ and feature vector $\boldsymbol{x}$ appear linearly. However the roles the vectors $\boldsymbol{w}$ and $\boldsymbol{x}$ play in a learning algorithm are different: a learning model is considered linear if the weight $\boldsymbol{w}$ *linearly* combines available features. On the other hand, the features from training data are known and treated as *constants*. Consequently, as long as the inner product assumes the form $\boldsymbol{w}^T(\text{combination of features})$, a learning model remains linear regardless of how the input features are combined and manipulated because they are merely numerical constants determined by the training data. In this section, we examine the effect of using some *nonlinear* combinations of the individual features x_i's on the performance of several linear models studied earlier, especially the ability in dealing with non-separable data sets. For clarity, we follow [8] to denote a transform that maps a feature vector $\boldsymbol{x} \in R^d$ to a nonlinearly dilated feature vector $\boldsymbol{z} \in R^{\tilde{d}}$ (this Euclidean space is called $\mathcal{Z}$ space) by

$$\boldsymbol{z} = \Phi(\boldsymbol{x}) \tag{3.24}$$

and the weight associated with $\tilde{d}$ - dimensional $\mathcal{Z}$ space by $\tilde{\boldsymbol{w}} \in R^{\tilde{d}}$. Natural choices of $\mathcal{Z}$ space include polynomial type of dilations. For a 2-dimesinal input space R^2 for example, the general 2nd-order and 3rd-order dilations assume the form

$$\Phi_2(\boldsymbol{x}) = \begin{bmatrix} 1 & x_1 & x_2 & x_1^2 & x_1x_2 & x_2^2 \end{bmatrix}^T \tag{3.25}$$

and

$$\Phi_3(\boldsymbol{x}) = \begin{bmatrix} 1 & x_1 & x_2 & x_1^2 & x_1x_2 & x_2^2 & x_1^3 & x_1^2x_2 & x_1x_2^2 & x_2^3 \end{bmatrix}^T \tag{3.26}$$

respectively. Note that as the order of nonlinearity increases, the dimension of the dilated feature space grows very quickly, this implies rapid increase of the VC dimension.

In principle, the methods addressed in this chapter so far can all be extended to include nonlinear transforms to work in dilated input space $\mathcal{Z}$, and the inner product $\boldsymbol{w}^T\boldsymbol{x}$ involved in the models as well as in the equations that define decision boundaries in the case of classification problems is replaced by $\tilde{\boldsymbol{w}}^T\boldsymbol{z}$ where $\boldsymbol{z}$ is related to $\boldsymbol{x}$ via $\boldsymbol{z} = \Phi(\boldsymbol{x})$. In this way, the linear decision boundary $\boldsymbol{w}^T\boldsymbol{x} = 0$ is replaced by $\tilde{\boldsymbol{w}}^T\boldsymbol{z} = 0$ which via (3.24) becomes

$$\tilde{\boldsymbol{w}}^T\Phi(\boldsymbol{x}) = 0 \tag{3.27}$$

Using polynomial type of nonlinear transforms such as those in (3.25) and (3.26), the decision boundary in (3.27) is nonlinear and has better chances to separate the data that cannot be done with linear decision boundaries.

Example 3.5

Develop linear regression that works with a dilated input space $\mathcal{Z}$ with $\Phi(\boldsymbol{x}) = \Phi_3(\boldsymbol{x})$ and apply it to the classification problem in Example 3.1.

Solution

The regression function is this case is given by

$$h(\boldsymbol{z}) = h\left(\Phi_3(\boldsymbol{x})\right) = \sum_{i=0}^{9} \tilde{w}_i z_i = \tilde{\boldsymbol{w}}^T\boldsymbol{z} \tag{3.28}$$

where $\boldsymbol{z}$ is a 10-dimensional vector with $z_0 = 1, z_1 = x_1, z_2 = x_2,\ z_3 = x_1^2, z_4 = x_1x_2, z_5 = x_2^2, z_6 = x_1^3,$ $z_7 = x_1^2x_2, z_8 = x_1x_2^2, z_9 = x_2^3$. The loss function is given by

$$E_{\text{in}}(h) = \frac{1}{N}\sum_{n=1}^{N}\left(h\left(\Phi_3(\boldsymbol{x}_n)\right) - y_n\right)^2$$

which in conjunction with (3.28) yields

$$E_{\text{in}}(h) = \frac{1}{N}\sum_{n=1}^{N}\left(h\left(\Phi_3(\boldsymbol{x}_n)\right) - y_n\right)^2 = \frac{1}{N}\sum_{n=1}^{N}\left(\tilde{\boldsymbol{w}}^T\boldsymbol{z}_n - y_n\right)^2 = \frac{1}{N}\left\|\boldsymbol{Z}\tilde{\boldsymbol{w}} - \boldsymbol{y}\right\|^2 \tag{3.29}$$

$$\boldsymbol{Z} = \begin{bmatrix} \boldsymbol{z}_1^T \\ \boldsymbol{z}_2^T \\ \vdots \\ \boldsymbol{z}_N^T \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

We see that $E_{\text{in}}(h)$ remains to be a convex quadratic function of $\tilde{\boldsymbol{w}}$ whose global minimum is achieved at

$$\tilde{\boldsymbol{w}}^* = \left(\boldsymbol{Z}^T\boldsymbol{Z}\right)^{-1}\boldsymbol{Z}^T\boldsymbol{y} \triangleq \boldsymbol{Z}^\dagger\boldsymbol{y} \tag{3.30}$$

Applying (3.30) to the data from Example 3.1, we obtain the numerical values of 10 components of $\tilde{\boldsymbol{w}}^*$ as

```
 0.412665257054635
-0.030808874414424
-0.167834080860251
-0.006375722902075
-0.007360582606473
```

```
-0.005499749941547
 0.000334340108927
 0.000542429175521
 0.000755834680100
 0.000782595233551
```

As can be seen from Fig. 3.9, the decision boundary defined by $\tilde{\boldsymbol{w}}^{*T}\Phi_3(\boldsymbol{x}) = 0$ separates to two subsets of the data successfully, which simply means zero in-sample error in terms of the percentage of misclassified data.

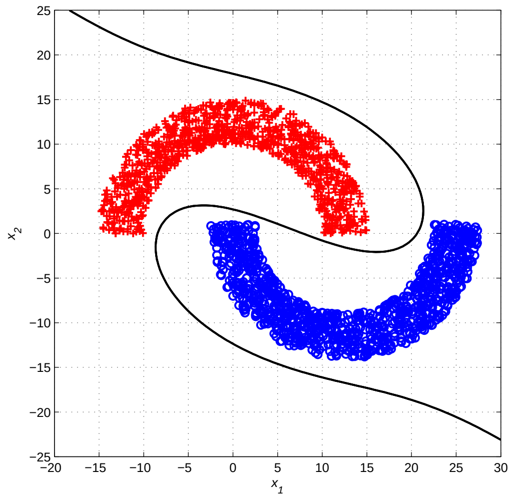


Figure 3.9 Decision boundary by linear regression with 3rd-order nonlinear transform.

The MATLAB code that implements linear regression with 3rd nonlinear transform as applied to the double semi-circle data and produces Fig. 3.9 is given below.

```
% To apply linear regression to classify nonseparable double demi-circle
% data using a dilated input space where vetors are dilated to include
% up to 3rd-order features.
% Written by W.-S. Lu, University of Victoria. Last modified: Jan. 25, 2015.
% Example:
% [x,y,xp,xn] = data_semi_circle(10,5,-1,1000,9,7);
% [wt,Ein1,Ein2] = regression_NL(x,y,xp,xn,11);
function [wt,Ein1,Ein2] = regression_NL(x,y,xp,xn,st1)
N = length(y);
rand('state',st1)
N1 = randperm(N);
x0 = x;
y0 = y;
x = x0(:,N1);
y = y0(N1);
z1 = zeros(7,N);
for i = 1:N,
   z1(:,i) = [x(1,i)^2; x(1,i)*x(2,i); x(2,i)^2; x(1,i)^3; x(1,i)^2*x(2,i); x(1,i)*x(2,i)^2; x(2,i)^3];
end
```

```
z = [x; z1];
Z = [ones(N,1) z'];
y = y(:);
wt = (inv(Z'*Z))*(Z'*y);
disp('Final weights:')
wt
Ein1 = norm(Z*wt-y)^2/N;
disp(sprintf('In-sample error was found to be %d.', Ein1));
dwt = (Z*wt >= 0);
z = dwt + dwt - y - 1;
L = sum(abs(z))/2;
Ein2 = L/N;
disp(sprintf('Out of N = %d sample points,',N));
disp(sprintf('%d points were classified correctly.',N-L));
[x1,x2] = meshgrid(-20:50/100:30,-25:50/100:25);
w0 = wt(1); w1 = wt(2); w2 = wt(3); w3 = wt(4); w4 = wt(5); w5 = wt(6);
w6 = wt(7); w7 = wt(8); w8 = wt(9); w9 = wt(10);
h1 = w0 + w1*x1 + w2*x2 + w3*(x1.^2) + w4*(x1.*x2) + w5*(x2.^2);
h = h1 + w6*(x1.^3) + w7*((x1.^2).*x2) + w8*(x1.*(x2.^2)) + w9*(x2.^3);
figure(1)
plot(xp(1,:),xp(2,:),'bo','linewidth',1.5)
hold on
plot(xn(1,:),xn(2,:),'r+','linewidth',1.5)
v = -1e-6:1e-6:1e-6;
contour(x1,x2,h,v,'k-','linewidth',1.5);
grid
xlabel('\itx_1')
ylabel('\itx_2')
axis square
axis([-20 30 -25 25])
hold off
```
■

The next example presents simulation results concerning the logistic regression with nonlinear transform for classification.

Example 3.6

Apply logistic regression with 3rd-order nonlinear transform to the classification problem in Example 3.1.

Solution

The problem was addressed using two optimization techniques, namely SDM and BFGS.

(a) ***Using SDM-BT***

With a10-dimensional feature space $\mathcal{Z}$ the objective function for logistic regression assumes the form

$$E_{\text{in}}(\tilde{\boldsymbol{w}}) = \frac{1}{N}\sum_{n=1}^{N} \ln\left(1 + e^{-y_n \tilde{\boldsymbol{w}}^T \boldsymbol{z}_n}\right) \tag{3.31}$$

whose gradient is given by

$$\nabla\left(E_{\text{in}}(\tilde{\boldsymbol{w}})\right) = -\frac{1}{N}\sum_{n=1}^{N}\frac{y_n\boldsymbol{z}_n}{1+e^{y_n\tilde{\boldsymbol{w}}^T\boldsymbol{z}_n}} \tag{3.32}$$

The above expressions are almost the same as those in (3.18) and (3.19) merely replacing $\boldsymbol{w}$ and $\boldsymbol{x}_n$ with $\tilde{\boldsymbol{w}}$ and $\boldsymbol{z}_n$, respectively. Thus the changes in MATLAB code needed for implementing the algorithm are fairly minor. A significant difference between the two sets of formulas is the increase of the variables from 3 to 10. Associated with that, the data set we deal with here, namely $\{(\boldsymbol{z}_n, y_n)$, for $n = 1, \ldots, 2000\}$, moves to the same high dimensional space and exhibits increasing numerical irregularity (see (3.28)). All these contribute to further slowness of the SDM. With $\boldsymbol{w}(0)$ = zeros(10,1), it took 10^5 iterations for SDM to converge to a weight vector $\boldsymbol{w}^*$ whose 10 components are shown below.

```
 0.287065071143320
-0.044616276316051
-0.087377965695571
-0.059737864637012
-0.053026567234247
 0.236067386235712
 0.003612581213920
-0.016650016709875
 0.071229795734063
-0.278067034535488
```

The associated decision boundary is shown in Fig, 3.10. The final in-sample error in terms of the percentage of misclassified data was found to be 0.003 (i.e. 0.3%), that outperforms all linear methods without nonlinear transform, but not as good as that of the linear regression with 3rd-order nonlinear transform as demonstrated in Example 3.5.

Evidently, improved performance can be obtained by executing more SDM iterations. As a mater of fact, running 2×10^5 SDM iterations yields a $\boldsymbol{w}^*$ that reduces the in-sample error (in terms of the percentage of misclassified data) to zero. The components of the optimal weight are given by

```
 0.748567483227266
-0.105175128517390
-0.131573469558948
-0.063641168135711
-0.097885507589359
 0.472612616904333
 0.003865924849607
-0.015072214074362
 0.091181072818658
-0.368710258973415
```

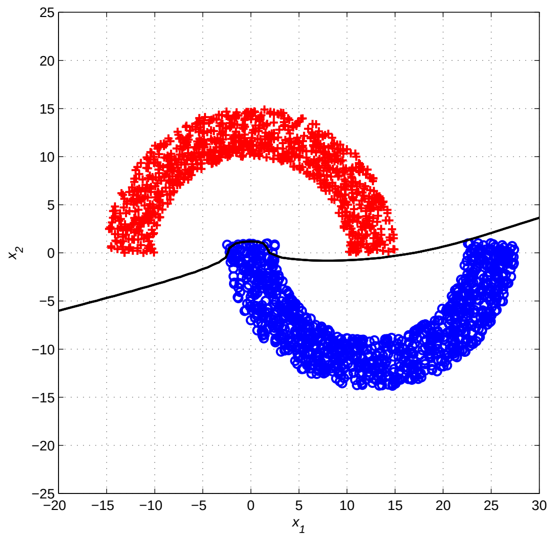


Figure 3.10 Decision boundary by logistic regression with 3rd-order nonlinear transform after 10^5 SDM iterations.

The associated decision boundary which successfully separates the two data subsets is shown in Fig, 3.11.

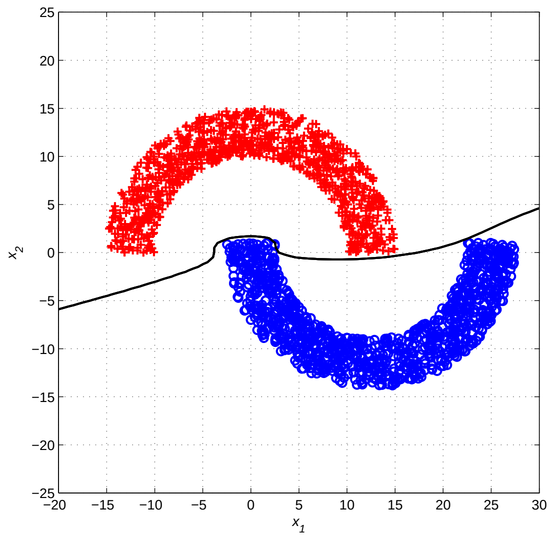


Figure 3.11 Decision boundary by logistic regression with 3rd-order nonlinear transform after 2×10^5 SDM iterations.

The MATLAB code that implements the logistic regression with 3rd-order nonlinear transform and SDM-BT as applied to the double semi-circle data is included below.

```
% To apply logistic regression with 3rd-order nonlinear transform
% to classify double nonseparable demi-circle data. SDM algorithm
% is chosen to minimize the logistic loss function.
% Written by W.-S. Lu, University of Victoria. Last modified: Jan. 25, 2015.
% Examples:
```

```
% [x,y,xp,xn] = data_semi_circle(10,5,-1,1000,9,7);
% [wt,fs] = logistic_SDM_NL(x,y,xp,xn,'logistic_f','logistic_g',zeros(10,1),11,100000);
% [wt,fs] = logistic_SDM_NL(x,y,xp,xn,'logistic_f','logistic_g',zeros(10,1),11,200000);
function [wt,fs] = logistic_SDM_NL(x,y,xp,xn,fname,gname,w0,st,K)
N = length(y);
rand('state',st)
N1 = randperm(N);
x0 = x;
y0 = y;
x = x0(:,N1);
y = y0(N1);
z1 = zeros(7,N);
for i = 1:N,
   z1(:,i) = [x(1,i)^2; x(1,i)*x(2,i); x(2,i)^2; x(1,i)^3; x(1,i)^2*x(2,i); x(1,i)*x(2,i)^2; x(2,i)^3];
end
z = [x; z1];
d1 = length(w0);
p = [N; d1; y; z(:)];
k = 0;
wk = w0;
while k < K,
      gk = feval(gname,wk,p);
      dk = -gk;
      ak = bt_lsearch(wk,dk,fname,gname,p);
      dtk = ak*dk;
      wk = wk + dtk;
      k = k + 1;
      % fk(k+1) = feval(fname,wk,p);
end
disp('Final weights:')
wt = wk
fs = feval(fname,wt,p);
disp(sprintf('Objective function at solution point: %d.', fs));
Z = [ones(N,1) z'];
y = y(:);
dwt = (Z*wt >= 0);
zt = dwt + dwt - y - 1;
L = sum(abs(zt))/2;
disp(sprintf('Out of N = %d sample points,',N));
disp(sprintf('%d points were classified correctly.',N-L));
[x1,x2] = meshgrid(-20:50/100:30,-25:50/100:25);
w0 = wt(1); w1 = wt(2); w2 = wt(3); w3 = wt(4); w4 = wt(5); w5 = wt(6);
w6 = wt(7); w7 = wt(8); w8 = wt(9); w9 = wt(10);
h1 = w0 + w1*x1 + w2*x2 + w3*(x1.^2) + w4*(x1.*x2) + w5*(x2.^2);
h = h1 + w6*(x1.^3) + w7*((x1.^2).*x2) + w8*(x1.*(x2.^2)) + w9*(x2.^3);
figure(1)
plot(xp(1,:),xp(2,:),'bo','linewidth',1.5)
hold on
plot(xn(1,:),xn(2,:),'r+','linewidth',1.5)
v = -1e-6:1e-6:1e-6;
contour(x1,x2,h,v,'k-','linewidth',1.5);
```

```
grid
xlabel('\itx_1')
ylabel('\itx_2')
axis square
axis([-20 30 -25 25])
hold off
```

■

(b) *Using BFGS-BT*

Logistic regression with 3rd-order nonlinear transform was also applied to the classification problem in Example 3.1, where the optimization was carried out by BFGS-BT. With ***w***(0) = zeros(10,1), it took 34 iterations for SDM to converge to a weight vector $\boldsymbol{w}^*$ whose 10 components are shown below.

```
 5.208531372296072
-0.603211687574635
-2.614896708089146
-0.164307452838640
 0.255000978291617
 0.732072511422932
 0.008795857137318
-0.018771408101358
-0.064320065099328
-0.073969868145694
```

The in-sample error in terms of percentage of misclassified data was zero and the associated decision boundary which successfully separates the two data subsets is shown in Fig, 3.12.

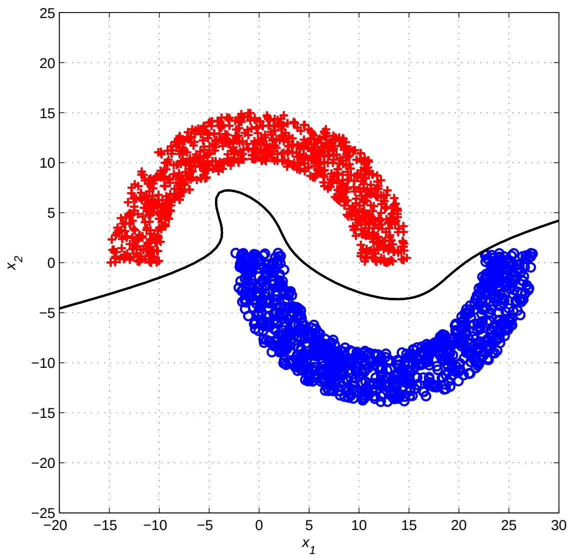


Figure 3.12 Decision boundary by logistic regression with 3rd-order nonlinear transform after 34 BFGS iterations.

Problems

3.1

(i) Download from the class website two random data sets named **xp** and xn, with size of xp being 2 by 91 and size of xn being 2 by 109. These data sets are linearly separable. You are asked to follow the steps described below to construct a training data set of the form $\mathcal{D} = \{(\boldsymbol{x}_n, y_n),\ n = 1, 2, ..., 200\}$

Step 1: Generate label 1 for each point in xp: yp = ones(91,1), and label –1 for each point in xn: yn = -ones(109,1);

Step 2: Combine the two data sets into one: xw = [xp xn]; yw = [yp; yn];

Step 3: Generate a training set with 200 examples by randomly shuffling the rows of xw and accordingly the components of yw:

```
rand('state',5)
N1 = randperm(200);
x = xw(:,N1);
y = yw(N1);
```

Data set $\mathcal{D}$ generated above will be used as the training data for classification data set outside $\mathcal{D}$.

(ii) Download from the class website two random sets named as **xp**_ns and xn_ns, with size of xp_ns being 2 by 92 and size of xn_ns being 2 by 108. These data sets are non-separable by linear boundary lines. You are asked to follow the steps described below to construct a training data set of the form $\mathcal{D}_{\text{ns}} = \{(\boldsymbol{x}_n, y_n),\ n = 1, 2, ..., 200\}$

Step 1: Generate label 1 for each point in xp_ns: yp_ns = ones(92,1), and label –1 for each point in xn_ns: yn_ns = -ones(108,1);

Step 2: Combine the two data sets into one: xt = [xp_ns xn_ns]; yt = [yp_ns; yn_ns];

Step 3: Generate a training set with 200 examples by randomly shuffling the rows of xt and accordingly the components of yt:

```
rand('state',15)
N1 = randperm(200);
x = xw(:,N1);
y = yw(N1);
```

Data set $\mathcal{D}_{\text{ns}}$ generated above will be used as the training data for classification data set outside $\mathcal{D}_{\text{ns}}$.

(iii) Apply PLA to the data set $\mathcal{D} = \{(\boldsymbol{x}_n, y_n),\ n = 1, 2, ..., 200\}$ generated in part (i) with w(0) = zeros(3,1). Report the numerical results in terms of the optimized weight $\boldsymbol{w}^*$ and in-sample error E_{in} (percentage of misclassified data pairs); include a plot showing the training data as points with binary labels and the decision boundary generated by $\boldsymbol{w}^*$; and include a plot showing the profile of E_{in} versus iterations. To earn credit, include your MATLAB code that implements PLA as applied to the present training data.

(iv) Apply a certain number of PLA iterations to the non-separable data set $\mathcal{D}_{\text{ns}} = \{(\boldsymbol{x}_n, y_n),$ $n = 1, 2, ..., 200\}$ generated in part (ii) with w(0) = zeros(3,1). Note that PLA does not converge when dealing with nonseparable data, hence the algorithm needs to be modified to just run a given number (*K*) of iterations, then terminate. Try your best to figure out a good K that yields satisfactory results. Report the numerical results in terms of the weight $\boldsymbol{w}^*$ produced by PLA and in-sample error E_{in} (percentage of misclassified data pairs); include a plot showing the training data as points with binary labels and the decision boundary generated by $\boldsymbol{w}^*$; and include a plot showing the profile of E_{in} versus iterations. To earn credit, include your MATLAB code that implements PLA as applied to the present training data.

3.2

(i) Apply the pocket algorithm (PCA) to the data set $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), \ n = 1, 2, ..., 200\}$ generated in part (i) with w(0) = zeros(3,1). Report the numerical results in terms of the optimized weight $\boldsymbol{w}^*$ and in-sample error E_{in} (percentage of misclassified data pairs); include a plot showing the training data as points with binary labels and the decision boundary generated by $\boldsymbol{w}^*$; and include a plot showing the profile of E_{in} versus iterations. To earn credit, include your MATLAB code that implements PLA as applied to the present training data.

(ii) Apply a certain number of PCA iterations to the non-separable data set $\mathcal{D}_{\text{ns}} = \{(\boldsymbol{x}_n, y_n),$ $n = 1, 2, ..., 200\}$ generated in part (ii) with w(0) = zeros(3,1). Note that PLA does not converge when dealing with nonseparable data, hence the algorithm needs to be modified to just run a given number (*K*) of iterations, then terminate. Try your best to figure out a good K that yields satisfactory results. Report the numerical results in terms of the weight $\boldsymbol{w}^*$ produced by PLA and in-sample error E_{in} (percentage of misclassified data pairs); include a plot showing the training data as points with binary labels and the decision boundary generated by $\boldsymbol{w}^*$; and include a plot showing the profile of E_{in} versus iterations. To earn credit, include your MATLAB code that implements PLA as applied to the present training data.

3.3 [8] Recall that the Adaline algorithm for classification was introduced in Sec. 2.4. Here the same algorithm is derived with an optimization perspective.

(i) Define $E_n(\boldsymbol{w}) = \max\{0, 1 - y_n \boldsymbol{w}^T \boldsymbol{x}_n\}^2$. Prove that $E_n(\boldsymbol{w})$ is continuous and differentiable. Derive an expression for the gradient of $E_n(\boldsymbol{w})$.

(ii) Show that $E_n(\boldsymbol{w})$ is an upper bound for $[\![\text{sign}(\boldsymbol{w}^T \boldsymbol{x}_n) \neq y_n]\!]$. Therefore, $\frac{1}{N}\sum_{n=1}^{N} E_n(\boldsymbol{w})$ is an upper bound for the in-sample classification error $E_{\text{in}}(\boldsymbol{w})$ (see Eq. (2.14)).

(iii) Argue that the Adaline algorithm described in Sec. 2.4 performs stochastic descent on $\frac{1}{N}\sum_{n=1}^{N} E_n(\boldsymbol{w})$.

3.4 [8] Following the steps described below to derive an alternative-form linear programming (LP) algorithm, namely

$$\begin{aligned} &\underset{\boldsymbol{z}}{\text{minimize}} \quad \boldsymbol{c}^T\boldsymbol{z} \\ &\text{subject to: } \boldsymbol{A}\boldsymbol{z} \le \boldsymbol{b} \end{aligned} \qquad \text{(P2.1)}$$

that fits a linear model for classification.

(i) For linearly separable data, show that there exists a weight vector $\boldsymbol{w}$ such that

$$y_n(\boldsymbol{w}^T\boldsymbol{x}_n) \ge 1 \text{ for } n = 1,\dots,N \qquad \text{(P2.2)}$$

(ii) Formulate the problem of finding a separating w for separable data as an LP problem. You need to specify what the parameters $\boldsymbol{A}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ are and what the optimization variable $\boldsymbol{z}$ is.

(iii) If the data is non-separable, the condition in (P2.2) cannot hold for every n. To deal with this problem, violation $\xi_n \ge 0$ may be introduced to capture the amount of violation for example $\boldsymbol{x}_n$. In this way, (P2.2) is modified to

$$\begin{aligned} y_n(\boldsymbol{w}^T\boldsymbol{x}_n) &\ge 1 - \xi_n \\ \xi_n &\ge 0 \end{aligned} \qquad \text{(P2.3)}$$

Evidently, we want to minimize the total violation $\sum_{n=1}^N \xi_n$, thus we seek to find a weight vector w that solve the constrained problem

$$\begin{aligned} &\underset{w,\, \xi_1,\dots,\, \xi_N}{\text{minimize}} \qquad \sum_{n=1}^N \xi_n \\ &\text{subject to: } y_n(\boldsymbol{w}^T\boldsymbol{x}_n) \ge 1 - \xi_n \\ &\qquad\qquad\qquad \xi_n \ge 0 \end{aligned} \qquad \text{(P2.4)}$$

Re-formulate problem (P2.4) into a "standard" LP problem as seen in (P2.1) with parameters $\boldsymbol{A}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ specified.

3.5 [8] For linear regression, the out-of-sample error is given by

$$E_{\text{out}}(h) = E\left[\left(h(\boldsymbol{x}) - y\right)^2\right]$$

Show that among all hypotheses, the one that minimizes E_{out} is given by

$$h^*(\boldsymbol{x}) = E\left[y \mid \boldsymbol{x}\right]$$

The function h^* can be treated as a deterministic target function, in which case we can write $y = h^*(\boldsymbol{x}) + \varepsilon(\boldsymbol{x})$ where $\varepsilon(\boldsymbol{x})$ is an input-dependent noise variable. Show that the expected value of $\varepsilon(\boldsymbol{x})$ is zero.

3.6 Apply linear regression to the classification problem whose training data is generated part (i) in Problem 3.1. Note that you are applying a machine learning algorithm designed for regression problems to solve a classification problem. Report the numerical results in terms of the optimized weight $\boldsymbol{w}^*$ and in-sample error E_{in} (percentage of misclassified data pairs); include a plot showing the training data as points with binary labels and the decision boundary generated by $\boldsymbol{w}^*$. Also comment on the pros and cons of the linear regression algorithm relative to PLA and PCA. To earn credit, include your MATLAB code that implements the linear regression method as applied to the present training data.

3.7 Apply linear regression to the classification problem whose training data is generated part (ii) in Problem 3.1. Note that you are applying a machine learning algorithm designed for regression problems to solve a classification problem. Report the numerical results in terms of the optimized weight $\boldsymbol{w}^*$ and in-sample error E_{in} (percentage of misclassified data pairs); include a plot showing the training data as points with binary labels and the decision boundary generated by $\boldsymbol{w}^*$. Also comment on the pros and cons of the linear regression algorithm relative to PLA and PCA that were applied to the same data set.

To earn credit, include your MATLAB code that implements the linear regression method as applied to the present training data.

3.8 Apply logistic regression to the classification problem whose training data is generated part (ii) in Problem 3.1. Note that you are applying a machine learning algorithm designed for probability prediction to solve a classification problem. Use SDM with backtracking (BT) line search to minimize the logistic loss unction in (3.17). Report the numerical results in terms of the optimized weight $\boldsymbol{w}^*$ and in-sample error E_{in} (percentage of misclassified data pairs); include a plot showing the training data as points with binary labels and the decision boundary generated by $\boldsymbol{w}^*$. Also comment on the pros and cons of the logistic regression algorithm relative to PLA, PCA, and linear regression that were applied to the same data set. To earn credit, include your MATLAB code that implements the logistic regression algorithm as applied to the present training data.

3.9 Apply logistic regression to the classification problem whose training data is generated part (ii) in Problem 3.1. Note that you are applying a machine learning algorithm designed for probability prediction to solve a classification problem. Use BFGS-BT to minimize the logistic loss unction in (3.17). Report the numerical results in terms of the optimized weight $\boldsymbol{w}^*$ and in-sample error E_{in} (percentage of misclassified data pairs); include a plot showing the training data as points with binary labels and the decision boundary generated by $\boldsymbol{w}^*$. Also comment on the pros and cons of the present version of the logistic regression algorithm relative to that using SDM-BT. To earn credit, include your MATLAB code that implements the logistic regression algorithm as applied to the present training data.

3.10 Apply logistic regression to the classification problem whose training data is generated part (ii) in Problem 3.1. Note that you are applying a machine learning algorithm designed for probability prediction to solve a classification problem. Use SGD to minimize the logistic loss unction in (3.17). Report the numerical results in terms of the optimized weight $\boldsymbol{w}^*$ and in-sample error E_{in} (percentage of misclassified data pairs); include a plot showing the training data as points with binary labels and the decision boundary generated by $\boldsymbol{w}^*$. Also comment on the pros and cons of the present version of the logistic regression algorithm relative to those using SDM-BT and BFGS-BT. To earn credit, include your MATLAB code that implements the logistic regression algorithm as applied to the present training data.

3.11 Apply linear regression with 2nd-order nonlinear transform (see (3.25)) to the classification problem whose training data is generated part (ii) in Problem 3.1. Report the numerical results in terms of the optimized weight $\boldsymbol{w}^*$ and in-sample error E_{in} (percentage of misclassified data pairs); include a plot showing the training data as points with binary labels and the decision boundary generated by $\boldsymbol{w}^*$. Compare the performance of this version of linear regression algorithm with that studied in Prob. 3.7. To earn credit, include your MATLAB code that implements the linear regression with 2nd-order nonlinear transform as applied to the present training data.

3.12 Apply linear regression with 3rd-order nonlinear transform (see (3.26)) to the classification problem whose training data is generated part (ii) in Problem 3.1. Report the numerical results in terms of the optimized weight $\boldsymbol{w}^*$ and in-sample error E_{in} (percentage of misclassified data pairs); include a plot showing the training data as points with binary labels and the decision boundary generated by $\boldsymbol{w}^*$. Compare the performance of this version of linear regression algorithm with that studied in Prob. 3.10. To earn credit, include your MATLAB code that implements the linear regression with 3rd-order nonlinear transform as applied to the present training data.

3.13 Apply logistic regression with 2nd-order nonlinear transform (see (3.25)) to the classification problem whose training data is generated part (ii) in Problem 3.1. Note that you are applying a machine learning algorithm designed for probability prediction to solve a classification problem. Use BFGS-BT to minimize the logistic loss unction in (3.17). Report the numerical results in terms of the optimized weight $\boldsymbol{w}^*$ and in-sample error E_{in} (percentage of misclassified data pairs); include a plot showing the training data as points with binary labels and the decision boundary generated by $\boldsymbol{w}^*$. Compare the performance of this version of logistic regression algorithm with that studied in Prob. 3.11. To earn credit, include your MATLAB code that implements the logistic regression algorithm with nonlinear transform as applied to the present training data.