Chapter 3

The Linear Model

We often wonder how to draw a line between two categories; right versus wrong, personal versus professional life, useful email versus spam, to name a few. A line is intuitively our first choice for a decision boundary. In learning, as in life, a line is also a good first choice.

In Chapter 1, we (and the machine ☺) learned a procedure to 'draw a line' between two categories based on data (the perceptron learning algorithm). We started by taking the hypothesis set $\mathcal{H}$ that included all possible lines (actually hyperplanes). The algorithm then searched for a good line in $\mathcal{H}$ by iteratively correcting the errors made by the current candidate line, in an attempt to improve E_{in}. As we saw in Chapter 2, the linear model set of lines has a small VC dimension and so is able to generalize well from E_{in} to E_{out}.

The aim of this chapter is to further develop the basic linear model into a powerful tool for learning from data. We branch into three important problems: the classification problem that we have seen and two other important problems called *regression* and *probability estimation*. The three problems come with different but related algorithms, and cover a lot of territory in learning from data. As a rule of thumb, when faced with learning problems, it is generally a winning strategy to try a linear model first.

3.1 Linear Classification

The linear model for classifying data into two classes uses a hypothesis set of linear classifiers, where each h has the form

$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^{\mathsf{T}}\mathbf{x}),$$

for some column vector $\mathbf{w} \in \mathbb{R}^{d+1}$, where d is the dimensionality of the input space, and the added coordinate $x_0 = 1$ corresponds to the bias 'weight' w_0 (recall that the input space $\mathcal{X} = \{1\} \times \mathbb{R}^d$ is considered d-dimensional since the added coordinate $x_0 = 1$ is fixed). We will use h and $\mathbf{w}$ interchangeably

to refer to the hypothesis when the context is clear. When we left Chapter 1, we had two basic criteria for learning:

1. Can we make sure that $E_{\text{out}}(g)$ is close to $E_{\text{in}}(g)$? This ensures that what we have learned in sample will generalize out of sample.

2. Can we make $E_{\text{in}}(g)$ small? This ensures that what we have learned in sample is a good hypothesis.

The first criterion was studied in Chapter 2. Specifically, the VC dimension of the linear model is only $d+1$ (Exercise 2.4). Using the VC generalization bound (2.12), and the bound (2.10) on the growth function in terms of the VC dimension, we conclude that with high probability,

$$E_{\text{out}}(g) = E_{\text{in}}(g) + O\left(\sqrt{\frac{d}{N}\ln N}\right). \tag{3.1}$$

Thus, when N is sufficiently large, E_{in} and E_{out} will be close to each other (see the definition of $O(\cdot)$ in the Notation table), and the first criterion for learning is fulfilled.

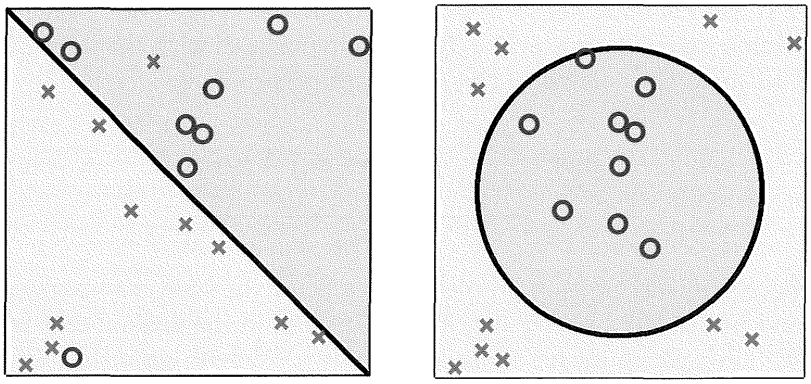
The second criterion, making sure that E_{in} is small, requires first and foremost that there is *some* linear hypothesis that has small E_{in}. If there isn't such a linear hypothesis, then learning certainly can't find one. So, let's suppose for the moment that there is a linear hypothesis with small E_{in}. In fact, let's suppose that the data is linearly separable, which means there is some hypothesis $\mathbf{w}^*$ with $E_{\text{in}}(\mathbf{w}^*) = 0$. We will deal with the case when this is not true shortly.

In Chapter 1, we introduced the perceptron learning algorithm (PLA). Start with an arbitrary weight vector $\mathbf{w}(0)$. Then, at every time step $t \geq 0$, select *any* misclassified data point $(\mathbf{x}(t), y(t))$, and update $\mathbf{w}(t)$ as follows:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + y(t)\mathbf{x}(t).$$

The intuition is that the update is attempting to correct the error in classifying $\mathbf{x}(t)$. The remarkable thing is that this incremental approach of learning based on one data point at a time works. As discussed in Problem 1.3, it can be proved that the PLA will eventually stop updating, ending at a solution $\mathbf{w}_{\text{PLA}}$ with $E_{\text{in}}(\mathbf{w}_{\text{PLA}}) = 0$. Although this result applies to a restricted setting (linearly separable data), it is a significant step. The PLA is clever – it doesn't naïvely test every linear hypothesis to see if it (the hypothesis) separates the data; that would take infinitely long. Using an iterative approach, the PLA manages to search an *infinite* hypothesis set and output a linear separator in (provably) *finite* time.

As far as PLA is concerned, linear separability is a property of the *data*, not the *target*. A linearly separable $\mathcal{D}$ could have been generated either from a linearly separable target, or (by chance) from a target that is not linearly separable. The convergence proof of PLA guarantees that the algorithm will



(a) Few noisy data. (b) Nonlinearly separable.

Figure 3.1: Data sets that are not linearly separable but are (a) linearly separable after discarding a few examples, or (b) separable by a more so phisticated curve.

work in both these cases, and produce a hypothesis with $E_{\text{in}} = 0$. Further, in both cases, you can be confident that this performance will generalize well out of sample, according to the VC bound.

Exercise 3.1

Will PLA ever stop updating if the data is not linearly separable?

3.1.1 Non-Separable Data

We now address the case where the data is not linearly separable. Figure 3.1 shows two data sets that are not linearly separable. In Figure 3.1(a), the data becomes linearly separable after the removal of just two examples, which could be considered noisy examples or outliers. In Figure 3.1(b), the data can be separated by a circle rather than a line. In both cases, there will always be a misclassified training example if we insist on using a linear hypothesis, and hence PLA will never terminate. In fact, its behavior becomes quite unstable, and can jump from a good perceptron to a very bad one within one update; the quality of the resulting E_{in} cannot be guaranteed. In Figure 3.1(a), it seems appropriate to stick with a line, but to somehow tolerate noise and output a hypothesis with a small E_{in}, not necessarily $E_{\text{in}} = 0$. In Figure 3.1(b), the linear model does not seem to be the correct model in the first place, and we will discuss a technique called nonlinear transformation for this situation in Section 3.4.

The situation in Figure 3.1(a) is actually encountered very often: even though a linear classifier seems appropriate, the data may not be linearly separable because of outliers or noise. To find a hypothesis with the minimum E_{in}, we need to solve the combinatorial optimization problem:

$$\min_{\mathbf{w}\in\mathbb{R}^{d+1}} \underbrace{\frac{1}{N}\sum_{n=1}^{N} [\![\text{sign}(\mathbf{w}^{\text{T}}\mathbf{x}_n) \neq y_n]\!]}_{E_{\text{in}}(\mathbf{w})}. \tag{3.2}$$

The difficulty in solving this problem arises from the discrete nature of both $\text{sign}(\cdot)$ and $[\![\cdot]\!]$. In fact, minimizing $E_{\text{in}}(\mathbf{w})$ in (3.2) in the general case is known to be NP-hard, which means there is no known efficient algorithm for it, and if you discovered one, you would become really, really famous ☺. Thus, one has to resort to approximately minimizing E_{in}.

One approach for getting an approximate solution is to extend PLA through a simple modification into what is called the *pocket algorithm*. Essentially, the pocket algorithm keeps 'in its pocket' the best weight vector encountered up to iteration t in PLA. At the end, the best weight vector will be reported as the final hypothesis. This simple algorithm is shown below.

The pocket algorithm:

1: Set the pocket weight vector $\hat{\mathbf{w}}$ to $\mathbf{w}(0)$ of PLA.
2: **for** $t = 0, \ldots, T \quad 1$ **do**
3: Run PLA for one update to obtain $\mathbf{w}(t+1)$.
4: Evaluate $E_{\text{in}}(\mathbf{w}(t+1))$.
5: If $\mathbf{w}(t+1)$ is better than $\hat{\mathbf{w}}$ in terms of E_{in}, set $\hat{\mathbf{w}}$ to $\mathbf{w}(t+1)$.
6: Return $\hat{\mathbf{w}}$.

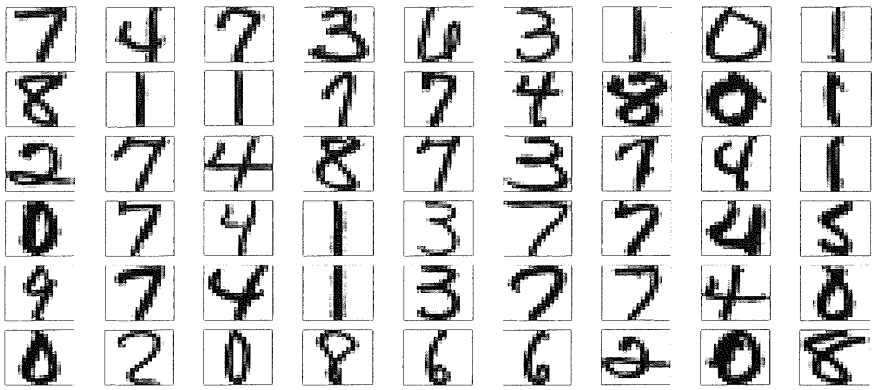
The original PLA only checks *some* of the examples using $\mathbf{w}(t)$ to identify $(\mathbf{x}(t), y(t))$ in each iteration, while the pocket algorithm needs an additional step that evaluates *all* examples using $\mathbf{w}(t+1)$ to get $E_{\text{in}}(\mathbf{w}(t+1))$. The additional step makes the pocket algorithm much slower than PLA. In addition, there is no guarantee for how fast the pocket algorithm can converge to a good E_{in}. Nevertheless, it is a useful algorithm to have on hand because of its simplicity. Other, more efficient approaches for obtaining good approximate solutions have been developed based on different optimization techniques, as shown later in this chapter.

Exercise 3.2

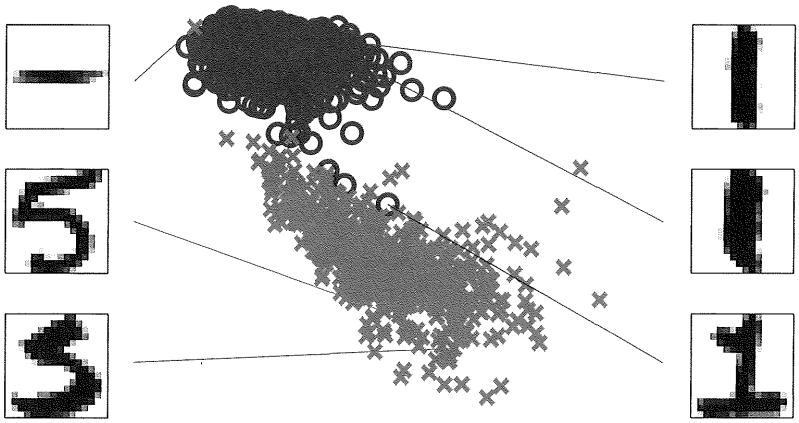
Take $d = 2$ and create a data set $\mathcal{D}$ of size $N = 100$ that is not linearly separable. You can do so by first choosing a random line in the plane as your target function and the inputs $\mathbf{x}_n$ of the data set as random points in the plane. Then, evaluate the target function on each $\mathbf{x}_n$ to get the corresponding output y_n. Finally, flip the labels of $\frac{N}{10}$ randomly selected y_n's and the data set will likely become non separable.

Now, try the pocket algorithm on your data set using $T = 1,000$ iterations. Repeat the experiment 20 times. Then, plot the average $E_{\text{in}}(\mathbf{w}(t))$ and the average $E_{\text{in}}(\hat{\mathbf{w}})$ (which is also a function of t) on the same figure and see how they behave when t increases. Similarly, use a test set of size $1,000$ and plot a figure to show how $E_{\text{out}}(\mathbf{w}(t))$ and $E_{\text{out}}(\hat{\mathbf{w}})$ behave.

Example 3.1 (Handwritten digit recognition)**.** We sample some digits from the US Postal Service Zip Code Database. These 16×16 pixel images are preprocessed from the scanned handwritten zip codes. The goal is to recognize the digit in each image. We alluded to this task in part (b) of Exercise 1.1. A quick look at the images reveals that this is a non-trivial task (even for a human), and typical human E_{out} is about 2.5%. Common confusion occurs between the digits $\{4, 9\}$ and $\{2, 7\}$. A machine-learned hypothesis which can achieve such an error rate would be highly desirable.



Let's first decompose the big task of separating ten digits into smaller tasks of separating two of the digits. Such a decomposition approach from *multiclass* to *binary* classification is commonly used in many learning algorithms. We will focus on digits $\{1, 5\}$ for now. A human approach to determining the digit corresponding to an image is to look at the shape (or other properties) of the black pixels. Thus, rather than carrying all the information in the 256 pixels, it makes sense to summarize the information contained in the image into a few *features*. Let's look at two important features here: intensity and symmetry. Digit 5 usually occupies more black pixels than digit 1, and hence the average pixel intensity of digit 5 is higher. On the other hand, digit 1 is symmetric while digit 5 is not. Therefore, if we define asymmetry as the average absolute difference between an image and its flipped versions, and symmetry as the negation of asymmetry, digit 1 would result in a higher symmetry value. A scatter plot for these intensity and symmetry features for some of the digits is shown next.



While the digits can be roughly separated by a line in the plane representing these two features, there are poorly written digits (such as the '5' depicted in the top-left corner) that prevent a perfect linear separation.

We now run PLA and pocket on the data set and see what happens. Since the data set is not linearly separable, PLA will not stop updating. In fact, as can be seen in Figure 3.2(a), its behavior can be quite unstable. When it is forcibly terminated at iteration $1,000$, PLA gives a line that has a poor $E_{\text{in}} = 2.24\%$ and $E_{\text{out}} = 6.37\%$. On the other hand, if the pocket algorithm is applied to the same data set, as shown in Figure 3.2(b), we can obtain a line that has a better $E_{\text{in}} = 0.45\%$ and a better $E_{\text{out}} = 1.89\%$. □

3.2 Linear Regression

Linear regression is another useful linear model that applies to real-valued target functions.[1] It has a long history in statistics, where it has been studied in great detail, and has various applications in social and behavioral sciences. Here, we discuss linear regression from a learning perspective, where we derive the main results with minimal assumptions.

Let us revisit our application in credit approval, this time considering a regression problem rather than a classification problem. Recall that the bank has customer records that contain information fields related to personal credit, such as annual salary, years in residence, outstanding loans, etc. Such variables can be used to learn a linear classifier to decide on credit approval. Instead of just making a binary decision (approve or not), the bank also wants to set a proper credit limit for each approved customer. Credit limits are traditionally determined by human experts. The bank wants to automate this task, as it did with credit approval.

[1] Regression, a term inherited from earlier work in statistics, means y is real valued.

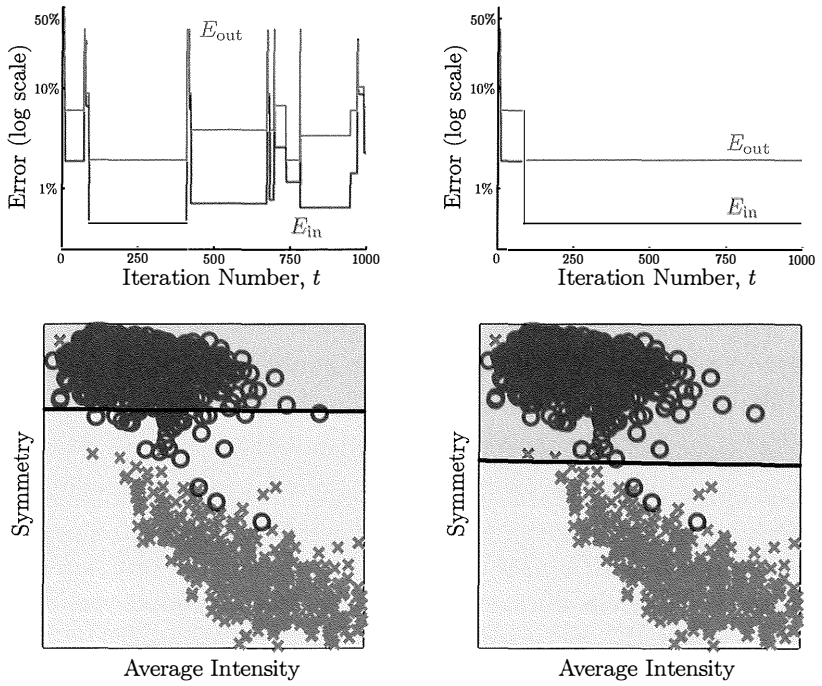


Figure 3.2: Comparison of two linear classification algorithms for sep arating digits 1 and 5. E_{in} and E_{out} are plotted versus iteration number and below that is the learned hypothesis g. (a) A version of the PLA which selects a random training example and updates $\mathbf{w}$ if that example is misclas sified (hence the flat regions when no update is made). This version avoids searching all the data at every iteration. (b) The pocket algorithm.

This is a regression learning problem. The bank uses historical records to construct a data set $\mathcal{D}$ of examples $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$, where $\mathbf{x}_n$ is customer information and y_n is the credit limit set by one of the human experts in the bank. Note that y_n is now a real number (positive in this case) instead of just a binary value ± 1. The bank wants to use learning to find a hypothesis g that replicates how human experts determine credit limits.

Since there is more than one human expert, and since each expert may not be perfectly consistent, our target will not be a deterministic function $y = f(\mathbf{x})$. Instead, it will be a noisy target formalized as a distribution of the random variable y that comes from the different views of different experts as well as the variation within the views of each expert. That is, the label y_n comes from some distribution $P(y \mid \mathbf{x})$ instead of a deterministic function $f(\mathbf{x})$. Nonetheless, as we discussed in previous chapters, the nature of the problem is not changed. We have an *unknown* distribution $P(\mathbf{x}, y)$ that generates

each $(\mathbf{x}_n, y_n)$, and we want to find a hypothesis g that minimizes the error between $g(\mathbf{x})$ and y with respect to that distribution.

The choice of a linear model for this problem presumes that there is a linear combination of the customer information fields that would properly approximate the credit limit as determined by human experts. If this assumption does not hold, we cannot achieve a small error with a linear model. We will deal with this situation when we discuss nonlinear transformation later in the chapter.

3.2.1 The Algorithm

The linear regression algorithm is based on minimizing the squared error between $h(\mathbf{x})$ and y.[2]

$$E_{\text{out}}(h) = \mathbb{E}\left[(h(\mathbf{x}) \quad y)^2\right],$$

where the expected value is taken with respect to the joint probability distribution $P(\mathbf{x}, y)$. The goal is to find a hypothesis that achieves a small $E_{\text{out}}(h)$. Since the distribution $P(\mathbf{x}, y)$ is unknown, $E_{\text{out}}(h)$ cannot be computed. Similar to what we did in classification, we resort to the in-sample version instead,

$$E_{\text{in}}(h) = \frac{1}{N}\sum_{n=1}^{N}(h(\mathbf{x}_n) \quad y_n)^2.$$

In linear regression, h takes the form of a linear combination of the components of $\mathbf{x}$. That is,

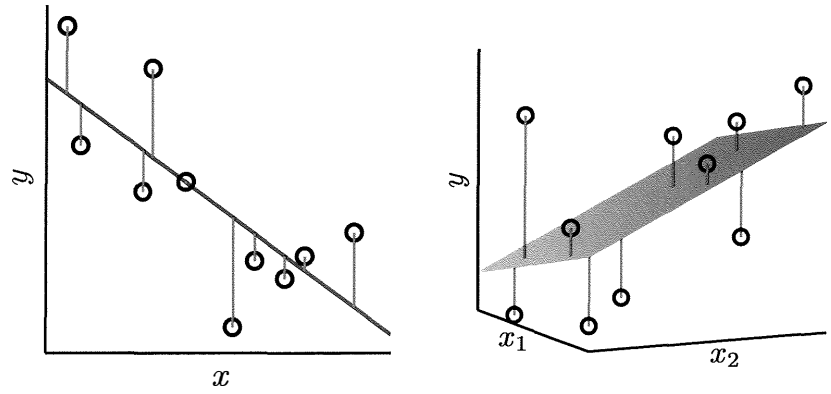
$$h(\mathbf{x}) = \sum_{i=0}^{d} w_i x_i = \mathbf{w}^{\mathrm{T}}\mathbf{x},$$

where $x_0 = 1$ and $\mathbf{x} \in \{1\} \times \mathbb{R}^d$ as usual, and $\mathbf{w} \in \mathbb{R}^{d+1}$. For the special case of linear h, it is very useful to have a matrix representation of $E_{\text{in}}(h)$. First, define the data matrix $\mathrm{X} \in \mathbb{R}^{N\times(d+1)}$ to be the $N \times (d+1)$ matrix whose rows are the inputs $\mathbf{x}_n$ as row vectors, and define the target vector $\mathbf{y} \in \mathbb{R}^N$ to be the column vector whose components are the target values y_n. The in-sample error is a function of $\mathbf{w}$ and the data $\mathrm{X}, \mathbf{y}$:

$$\begin{aligned} E_{\text{in}}(\mathbf{w}) &= \frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n \quad y_n)^2 \\ &= \frac{1}{N}\|\mathrm{X}\mathbf{w} - \mathbf{y}\|^2 \qquad (3.3) \\ &= \frac{1}{N}(\mathbf{w}^{\mathrm{T}}\mathrm{X}^{\mathrm{T}}\mathrm{X}\mathbf{w} - 2\mathbf{w}^{\mathrm{T}}\mathrm{X}^{\mathrm{T}}\mathbf{y} + \mathbf{y}^{\mathrm{T}}\mathbf{y}), \qquad (3.4) \end{aligned}$$

where $\|\cdot\|$ is the Euclidean norm of a vector, and (3.3) follows because the nth component of the vector $\mathrm{X}\mathbf{w} - \mathbf{y}$ is exactly $\mathbf{w}^{\mathrm{T}}\mathbf{x}_n \quad y_n$. The linear regression

[2] The term 'linear regression' has been historically confined to squared error measures.



(a) one dimension (line) (b) two dimensions (hyperplane)

Figure 3.3: The solution hypothesis (in blue) of the linear regression algorithm in one and two dimensions. The sum of squared errors is minimized.

algorithm is derived by minimizing $E_{\text{in}}(\mathbf{w})$ over all possible $\mathbf{w} \in \mathbb{R}^{d+1}$, as formalized by the following optimization problem:

$$\mathbf{w}_{\text{lin}} = \underset{\mathbf{w}\in\mathbb{R}^{d+1}}{\operatorname{argmin}} E_{\text{in}}(\mathbf{w}). \tag{3.5}$$

Figure 3.3 illustrates the solution in one and two dimensions. Since Equation (3.4) implies that $E_{\text{in}}(\mathbf{w})$ is differentiable, we can use standard matrix calculus to find the $\mathbf{w}$ that minimizes $E_{\text{in}}(\mathbf{w})$ by requiring that the gradient of E_{in} with respect to $\mathbf{w}$ is the zero vector, i.e., $\nabla E_{\text{in}}(\mathbf{w}) = \mathbf{0}$. The gradient is a (column) vector whose ith component is $[\nabla E_{\text{in}}(\mathbf{w})]_i = \frac{\partial}{\partial w_i} E_{\text{in}}(\mathbf{w})$. By explicitly computing $\frac{\partial}{\partial w_i}$, the reader can verify the following gradient identities,

$$\nabla_{\mathbf{w}}(\mathbf{w}^{\mathrm{T}}\mathrm{A}\mathbf{w}) = (\mathrm{A} + \mathrm{A}^{\mathrm{T}})\mathbf{w}, \qquad \nabla_{\mathbf{w}}(\mathbf{w}^{\mathrm{T}}\mathbf{b}) = \mathbf{b}.$$

These identities are the matrix analog of ordinary differentiation of quadratic and linear functions. To obtain the gradient of E_{in}, we take the gradient of each term in (3.4) to obtain

$$\nabla E_{\text{in}}(\mathbf{w}) = \frac{2}{N}(\mathrm{X}^{\mathrm{T}}\mathrm{X}\mathbf{w} - \mathrm{X}^{\mathrm{T}}\mathbf{y}).$$

Note that both $\mathbf{w}$ and $\nabla E_{\text{in}}(\mathbf{w})$ are column vectors. Finally, to get $\nabla E_{\text{in}}(\mathbf{w})$ to be $\mathbf{0}$, one should solve for $\mathbf{w}$ that satisfies

$$\mathrm{X}^{\mathrm{T}}\mathrm{X}\mathbf{w} = \mathrm{X}^{\mathrm{T}}\mathbf{y}.$$

If $\mathrm{X}^{\mathrm{T}}\mathrm{X}$ is invertible, $\mathbf{w} = \mathrm{X}^{\dagger}\mathbf{y}$ where $\mathrm{X}^{\dagger} = (\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathrm{T}}$ is the *pseudo-inverse* of X. The resulting $\mathbf{w}$ is the unique optimal solution to (3.5). If $\mathrm{X}^{\mathrm{T}}\mathrm{X}$ is not

invertible, a pseudo-inverse can still be defined, but the solution will not be unique (see Problem 3.15). In practice, $\mathrm{X}^{\mathrm{T}}\mathrm{X}$ is invertible in most of the cases since N is often much bigger than $d+1$, so there will likely be $d+1$ linearly independent vectors $\mathbf{x}_n$. We have thus derived the following *linear regression algorithm.*

Linear regression algorithm:

1: Construct the matrix X and the vector $\mathbf{y}$ from the data set $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)$, where each $\mathbf{x}$ includes the $x_0 = 1$ bias coordinate, as follows

$$\underbrace{\mathrm{X} = \begin{bmatrix} —\mathbf{x}_1^{\mathrm{T}}— \\ —\mathbf{x}_2^{\mathrm{T}}— \\ \vdots \\ —\mathbf{x}_N^{\mathrm{T}}— \end{bmatrix}}_{\text{input data matrix}}, \qquad \underbrace{\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\text{target vector}}.$$

2: Compute the pseudo-inverse $\mathrm{X}^{\dagger}$ of the matrix X. If $\mathrm{X}^{\mathrm{T}}\mathrm{X}$ is invertible,

$$\mathrm{X}^{\dagger} = (\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathrm{T}}.$$

3: Return $\mathbf{w}_{\mathrm{lin}} = \mathrm{X}^{\dagger}\mathbf{y}$.

This algorithm is sometimes referred to as *ordinary least squares* (OLS). It may seem that, compared with the perceptron learning algorithm, linear regression doesn't really look like 'learning', in the sense that the hypothesis $\mathbf{w}_{\mathrm{lin}}$ comes from an analytic solution (matrix inversion and multiplications) rather than from iterative learning steps. Well, as long as the hypothesis $\mathbf{w}_{\mathrm{lin}}$ has a decent out-of-sample error, then learning *has* occurred. Linear regression is a rare case where we have an analytic formula for learning that is easy to evaluate. This is one of the reasons why the technique is so widely used. It should be noted that there are methods for computing the pseudo-inverse directly without inverting a matrix, and that these methods are numerically more stable than matrix inversion.

Linear regression has been analyzed in great detail in statistics. We would like to mention one of the analysis tools here since it relates to in-sample and out-of-sample errors, and that is the *hat matrix* H. Here is how H is defined. The linear regression weight vector $\mathbf{w}_{\mathrm{lin}}$ is an attempt to map the inputs X to the outputs $\mathbf{y}$. However, $\mathbf{w}_{\mathrm{lin}}$ does not produce $\mathbf{y}$ exactly, but produces an estimate

$$\hat{\mathbf{y}} = \mathrm{X}\mathbf{w}_{\mathrm{lin}}$$

which differs from $\mathbf{y}$ due to in-sample error. Substituting the expression for $\mathbf{w}_{\mathrm{lin}}$ (assuming $\mathrm{X}^{\mathrm{T}}\mathrm{X}$ is invertible), we get

$$\hat{\mathbf{y}} = \mathrm{X}(\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathrm{T}}\mathbf{y}.$$

Therefore the estimate $\hat{\mathbf{y}}$ is a linear transformation of the actual $\mathbf{y}$ through matrix multiplication with H, where

$$\mathrm{H} = \mathrm{X}(\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathrm{T}}. \tag{3.6}$$

Since $\hat{\mathbf{y}} = \mathrm{H}\mathbf{y}$, the matrix H 'puts a hat' on $\mathbf{y}$, hence the name. The hat matrix is a very special matrix. For one thing, $\mathrm{H}^2 = \mathrm{H}$, which can be verified using the above expression for H. This and other properties of H will facilitate the analysis of in-sample and out-of-sample errors of linear regression.

Exercise 3.3

Consider the hat matrix $\mathrm{H} = \mathrm{X}(\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathrm{T}}$, where X is an N by $d+1$ matrix, and $\mathrm{X}^{\mathrm{T}}\mathrm{X}$ is invertible.

(a) Show that H is symmetric.

(b) Show that $\mathrm{H}^K = \mathrm{H}$ for any positive integer K.

(c) If I is the identity matrix of size N, show that $(\mathrm{I} - \mathrm{H})^K = \mathrm{I} - \mathrm{H}$ for any positive integer K.

(d) Show that $\mathrm{trace}(\mathrm{H}) = d + 1$, where the trace is the sum of diagonal elements. *[Hint:* $\mathrm{trace}(\mathrm{AB}) = \mathrm{trace}(\mathrm{BA})$.*]*

3.2.2 Generalization Issues

Linear regression looks for the optimal weight vector in terms of the in-sample error E_{in}, which leads to the usual generalization question: Does this guarantee decent out-of-sample error E_{out}? The short answer is yes. There is a regression version of the VC generalization bound (3.1) that similarly bounds E_{out}. In the case of linear regression in particular, there are also exact formulas for the expected E_{out} and E_{in} that can be derived under simplifying assumptions. The general form of the result is

$$E_{\mathrm{out}}(g) = E_{\mathrm{in}}(g) + O\left(\frac{d}{N}\right),$$

where $E_{\mathrm{out}}(g)$ and $E_{\mathrm{in}}(g)$ are the expected values. This is comparable to the classification bound in (3.1).

Exercise 3.4

Consider a noisy target $y = \mathbf{w}^{*\mathrm{T}}\mathbf{x} + \epsilon$ for generating the data, where ϵ is a noise term with zero mean and σ^2 variance, independently generated for every example $(\mathbf{x}, y)$. The expected error of the best possible linear fit to this target is thus σ^2.

For the data $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, denote the noise in y_n as ϵ_n and let $\boldsymbol{\epsilon} = [\epsilon_1, \epsilon_2, \ldots, \epsilon_N]^{\mathrm{T}}$; assume that $\mathrm{X}^{\mathrm{T}}\mathrm{X}$ is invertible. By following

(continued on next page)

the steps below, show that the expected in sample error of linear regression with respect to $\mathcal{D}$ is given by

$$\mathbb{E}_{\mathcal{D}}[E_{\text{in}}(\mathbf{w}_{\text{lin}})] = \sigma^2\left(1 - \frac{d+1}{N}\right).$$

(a) Show that the in sample estimate of $\mathbf{y}$ is given by $\hat{\mathbf{y}} = \mathrm{X}\mathbf{w}^* + \mathrm{H}\boldsymbol{\epsilon}$.

(b) Show that the in sample error vector $\hat{\mathbf{y}} - \mathbf{y}$ can be expressed by a matrix times $\boldsymbol{\epsilon}$. What is the matrix?

(c) Express $E_{\text{in}}(\mathbf{w}_{\text{lin}})$ in terms of $\boldsymbol{\epsilon}$ using (b), and simplify the expression using Exercise 3.3(c).

(d) Prove that $\mathbb{E}_{\mathcal{D}}[E_{\text{in}}(\mathbf{w}_{\text{lin}})] = \sigma^2\left(1 - \frac{d+1}{N}\right)$ using (c) and the independence of $\epsilon_1, \cdots, \epsilon_N$. *[Hint: The sum of the diagonal elements of a matrix (the trace) will play a role. See Exercise 3.3(d).]*

For the expected out of sample error, we take a special case which is easy to analyze. Consider a test data set $\mathcal{D}_{\text{test}} = \{(\mathbf{x}_1, y_1'), \ldots, (\mathbf{x}_N, y_N')\}$, which shares the same input vectors $\mathbf{x}_n$ with $\mathcal{D}$ but with a different realization of the noise terms. Denote the noise in y_n' as ϵ_n' and let $\boldsymbol{\epsilon}' = [\epsilon_1', \epsilon_2', \ldots, \epsilon_N']^{\mathrm{T}}$. Define $E_{\text{test}}(\mathbf{w}_{\text{lin}})$ to be the average squared error on $\mathcal{D}_{\text{test}}$.

(e) Prove that $\mathbb{E}_{\mathcal{D},\boldsymbol{\epsilon}'}[E_{\text{test}}(\mathbf{w}_{\text{lin}})] = \sigma^2\left(1 + \frac{d+1}{N}\right)$.

The special test error E_{test} is a very restricted case of the general out-of sample error. Some detailed analysis shows that similar results can be obtained for the general case, as shown in Problem 3.11.

Figure 3.4 illustrates the learning curve of linear regression under the assumptions of Exercise 3.4. The best possible linear fit has expected error σ^2. The expected in-sample error is smaller, equal to $\sigma^2(1 - \frac{d+1}{N})$ for $N \geq d+1$. The learned linear fit has eaten into the in-sample noise as much as it could with the $d+1$ degrees of freedom that it has at its disposal. This occurs because the fitting cannot distinguish the noise from the 'signal.' On the other hand, the expected out-of-sample error is $\sigma^2(1 + \frac{d+1}{N})$, which is more than the unavoidable error of σ^2. The additional error reflects the drift in $\mathbf{w}_{\text{lin}}$ due to fitting the in-sample noise.

3.3 Logistic Regression

The core of the linear model is the 'signal' $s = \mathbf{w}^{\mathrm{T}}\mathbf{x}$ that combines the input variables linearly. We have seen two models based on this signal, and we are now going to introduce a third. In linear regression, the signal itself is taken as the output, which is appropriate if you are trying to predict a real response that could be unbounded. In linear classification, the signal is thresholded at zero to produce a ± 1 output, appropriate for binary decisions. A third possibility, which has wide application in practice, is to output a *probability*,