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

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

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.

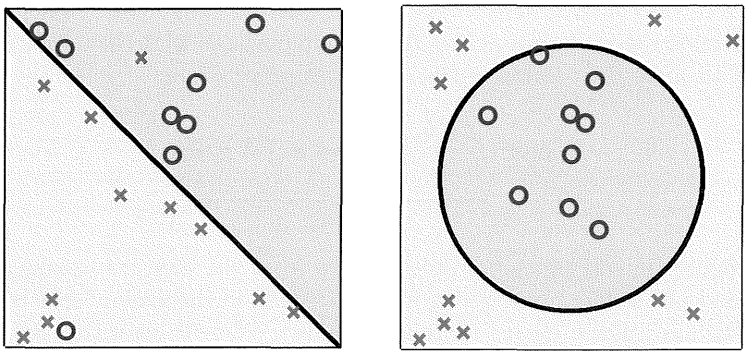
The second criterion, making sure that E_{in} is small, requires first and foremost that there is *some* linear hypothesis that has small E_{in}. If there isn't such a linear hypothesis, then learning certainly can't find one. So, let's suppose for the moment that there is a linear hypothesis with small 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 sophisticated 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** $t = 0, \ldots, T-1$ **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 $\hat{\mathbf{w}}$.

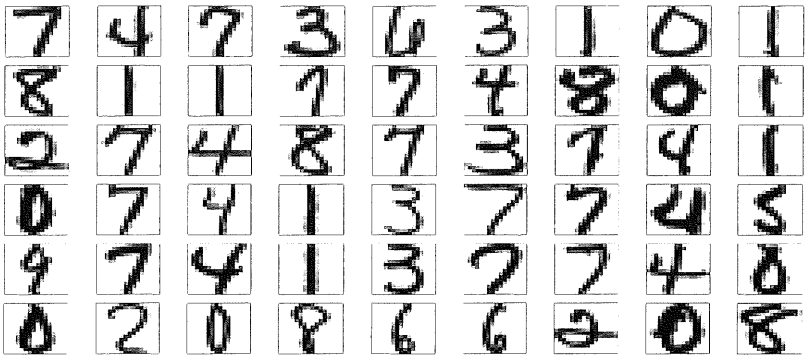
The original PLA only checks *some* of the examples using $\mathbf{w}(t)$ to identify $(\mathbf{x}(t), y(t))$ in each iteration, while the pocket algorithm needs an additional step that evaluates *all* examples using $\mathbf{w}(t+1)$ to get $E_{\text{in}}(\mathbf{w}(t+1))$. The additional step makes 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

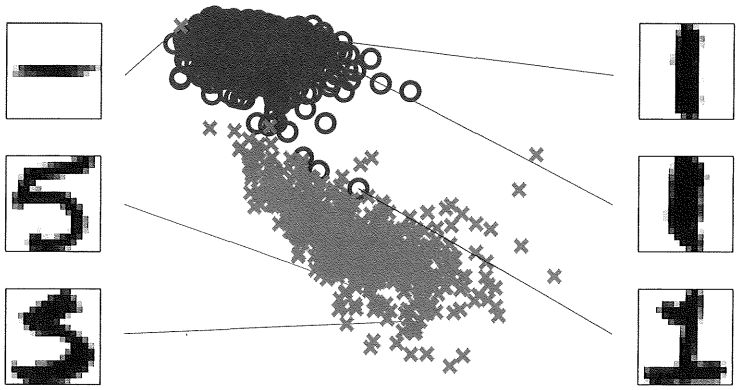
Take $d = 2$ and create a data set $\mathcal{D}$ of size $N = 100$ that is not linearly separable. You can do so by first choosing a random line in the plane as your target function and the inputs $\mathbf{x}_n$ of the data set as random points in the plane. Then, evaluate the target function on each $\mathbf{x}_n$ 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.

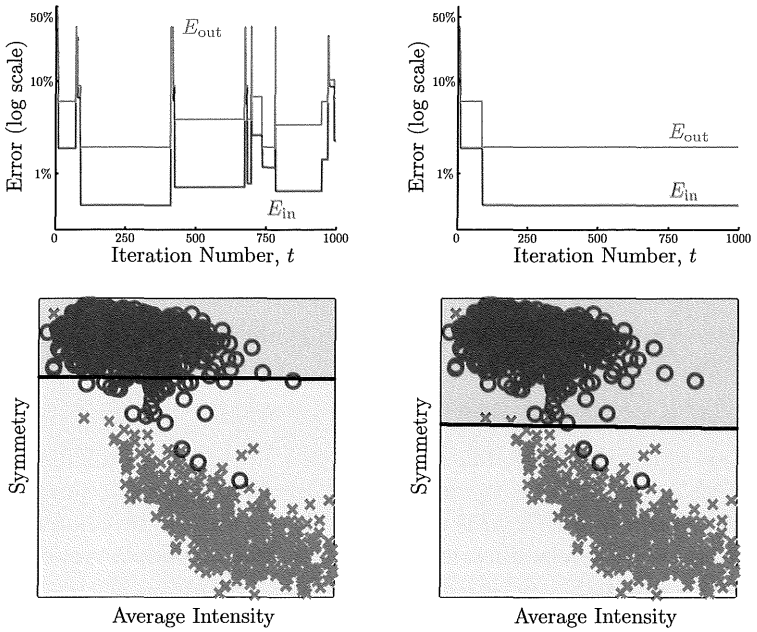
We now run PLA and pocket on the data set and see what happens. Since the data set is not linearly separable, PLA will not stop updating. In fact, as can be seen in Figure 3.2(a), its behavior can be quite unstable. When it is forcibly terminated at iteration $1,000$, PLA gives a line that has a poor $E_{\text{in}} = 2.24\%$ 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.



(a) PLA (b) Pocket

Figure 3.2: Comparison of two linear classification algorithms for separating 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 misclassified (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]

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

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,

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

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

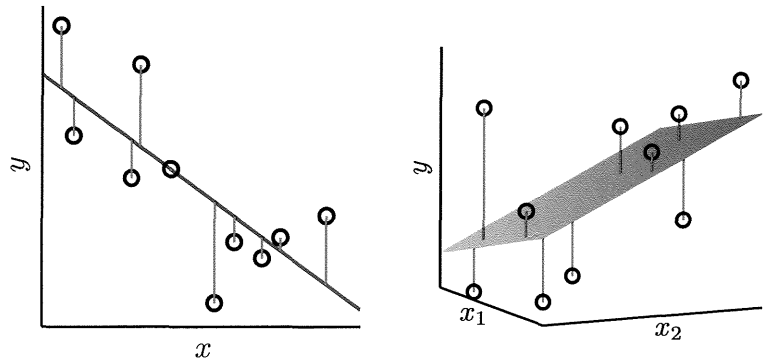
$$h(\mathbf{x}) = \sum_{i=0}^{d} w_i x_i = \mathbf{w}^{\mathrm{T}}\mathbf{x},$$

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 $\mathrm{X}, \mathbf{y}$:

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

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 $\mathbf{w}^{\mathrm{T}}\mathbf{x}_n - y_n$. 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*,

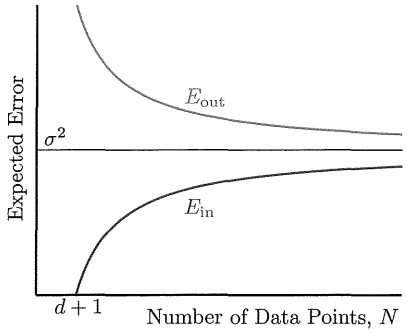


Figure 3.4: The learning curve for linear regression.

a value between 0 and 1. Our new model is called *logistic* regression. It has similarities to both previous models, as the output is real (like regression) but bounded (like classification).

Example 3.2 (Prediction of heart attacks)**.** Suppose we want to predict the occurrence of heart attacks based on a person's cholesterol level, blood pressure, age, weight, and other factors. Obviously, we cannot predict a heart attack with any certainty, but we may be able to predict how likely it is to occur given these factors. Therefore, an output that varies continuously between 0 and 1 would be a more suitable model than a binary decision. The closer y is to 1, the more likely that the person will have a heart attack. □

3.3.1 Predicting a Probability

Linear classification uses a hard threshold on the signal $s = \mathbf{w}^{\mathrm{T}}\mathbf{x}$,

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

while linear regression uses no threshold at all,

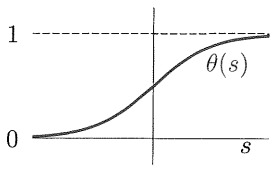
$$h(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}.$$

In our new model, we need something in between these two cases that smoothly restricts the output to the probability range $[0, 1]$. One choice that accomplishes this goal is the logistic regression model,

$$h(\mathbf{x}) = \theta(\mathbf{w}^{\mathrm{T}}\mathbf{x}),$$

where θ is the so-called *logistic* function $\theta(s) = \frac{e^s}{1+e^s}$ whose output is between 0 and 1.

The output can be interpreted as a probability for a binary event (heart attack or no heart attack, digit '1' versus digit '5', etc.). Linear classification also deals with a binary event, but the difference is that the 'classification' in logistic regression is allowed to be uncertain, with intermediate values between 0 and 1 reflecting this uncertainty. The logistic function θ is referred to as a *soft threshold*, in contrast to the hard threshold in classification. It is also called a *sigmoid* because its shape looks like a flattened out 's'.



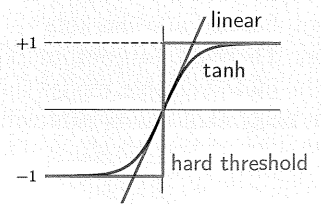
Exercise 3.5

Another popular soft threshold is the hyperbolic tangent

$$\tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}.$$

(a) How is tanh related to the logistic function θ? *[Hint: shift and scale]*

(b) Show that $\tanh(s)$ converges to a hard threshold for large $|s|$, and converges to no threshold for small $|s|$. *[Hint: Formalize the figure below.]*



The specific formula of $\theta(s)$ will allow us to define an error measure for learning that has analytical and computational advantages, as we will see shortly. Let us first look at the target that logistic regression is trying to learn. The target is a probability, say of a patient being at risk for heart attack, that depends on the input $\mathbf{x}$ (the characteristics of the patient). Formally, we are trying to learn the target function

$$f(\mathbf{x}) = \mathbb{P}[y = +1 \mid \mathbf{x}].$$

The data does not give us the value of f explicitly. Rather, it gives us samples generated by this probability, e.g., patients who had heart attacks and patients who didn't. Therefore, the data is in fact generated by a noisy target $P(y \mid \mathbf{x})$,

$$P(y \mid \mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{for } y = +1; \\ 1 - f(\mathbf{x}) & \text{for } y = -1. \end{cases} \tag{3.7}$$

To learn from such data, we need to define a proper error measure that gauges how close a given hypothesis h is to f in terms of these noisy ± 1 examples.

Error measure. The standard error measure $\mathsf{e}(h(\mathbf{x}), y)$ used in logistic regression is based on the notion of *likelihood*; how 'likely' is it that we would get this output y from the input $\mathbf{x}$ if the target distribution $P(y \mid \mathbf{x})$ was indeed captured by our hypothesis $h(\mathbf{x})$? Based on (3.7), that likelihood would be

$$P(y \mid \mathbf{x}) = \begin{cases} h(\mathbf{x}) & \text{for } y = +1; \\ 1 - h(\mathbf{x}) & \text{for } y = -1. \end{cases}$$

We substitute for $h(\mathbf{x})$ by its value $\theta(\mathbf{w}^{\mathrm{T}}\mathbf{x})$, and use the fact that $1 - \theta(s) = \theta(-s)$ (easy to verify) to get

$$P(y \mid \mathbf{x}) = \theta(y\, \mathbf{w}^{\mathrm{T}}\mathbf{x}). \tag{3.8}$$

One of our reasons for choosing the mathematical form $\theta(s) = e^s/(1+e^s)$ is that it leads to this simple expression for $P(y \mid \mathbf{x})$.

Since the data points $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ are independently generated, the probability of getting all the y_n's in the data set from the corresponding $\mathbf{x}_n$'s would be the product

$$\prod_{n=1}^{N} P(y_n \mid \mathbf{x}_n).$$

The method of *maximum likelihood* selects the hypothesis h which maximizes this probability.[3] We can equivalently minimize a more convenient quantity,

$$-\frac{1}{N} \ln \left(\prod_{n=1}^{N} P(y_n \mid \mathbf{x}_n) \right) = \frac{1}{N} \sum_{n=1}^{N} \ln \left(\frac{1}{P(y_n \mid \mathbf{x}_n)} \right),$$

since '$-\frac{1}{N}\ln(\cdot)$' is a monotonically decreasing function. Substituting with Equation (3.8), we would be minimizing

$$\frac{1}{N} \sum_{n=1}^{N} \ln \left(\frac{1}{\theta(y_n \mathbf{w}^{\mathrm{T}}\mathbf{x}_n)} \right)$$

with respect to the weight vector $\mathbf{w}$. The fact that we are *minimizing* this quantity allows us to treat it as an 'error measure.' Substituting the functional form for $\theta(y_n \mathbf{w}^{\mathrm{T}}\mathbf{x}_n)$ produces the in-sample error measure for logistic regression,

$$E_{\mathrm{in}}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \ln \left(1 + e^{-y_n \mathbf{w}^{\mathrm{T}}\mathbf{x}_n} \right). \tag{3.9}$$

The implied pointwise error measure is $\mathsf{e}(h(\mathbf{x}_n), y_n) = \ln(1+e^{-y_n \mathbf{w}^{\mathrm{T}}\mathbf{x}_n})$. Notice that this error measure is small when $y_n \mathbf{w}^{\mathrm{T}}\mathbf{x}_n$ is large and *positive*, which would imply that $\mathrm{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n) = y_n$. Therefore, as our intuition would expect, the error measure encourages $\mathbf{w}$ to 'classify' each $\mathbf{x}_n$ correctly.

[3] Although the method of maximum likelihood is intuitively plausible, its rigorous justification as an inference tool continues to be discussed in the statistics community.

Exercise 3.6 [Cross-entropy error measure]

(a) More generally, if we are learning from ± 1 data to predict a noisy target $P(y \mid \mathbf{x})$ with candidate hypothesis h, show that the maximum likelihood method reduces to the task of finding h that minimizes

$$E_{\text{in}}(\mathbf{w}) = \sum_{n=1}^{N} [\![y_n = +1]\!] \ln \frac{1}{h(\mathbf{x}_n)} + [\![y_n = -1]\!] \ln \frac{1}{1 - h(\mathbf{x}_n)}.$$

(b) For the case $h(\mathbf{x}) = \theta(\mathbf{w}^{\mathrm{T}}\mathbf{x})$, argue that minimizing the in-sample error in part (a) is equivalent to minimizing the one in (3.9).

For two probability distributions $\{p, 1-p\}$ and $\{q, 1-q\}$ with binary outcomes, the cross-entropy (from information theory) is

$$p \log \frac{1}{q} + (1-p) \log \frac{1}{1-q}.$$

The in-sample error in part (a) corresponds to a cross-entropy error measure on the data point $(\mathbf{x}_n, y_n)$, with $p = [\![y_n = +1]\!]$ and $q = h(\mathbf{x}_n)$.

For linear classification, we saw that minimizing E_{in} for the perceptron is a combinatorial optimization problem; to solve it, we introduced a number of algorithms such as the perceptron learning algorithm and the pocket algorithm. For linear regression, we saw that training can be done using the analytic pseudo-inverse algorithm for minimizing E_{in} by setting $\nabla E_{\text{in}}(\mathbf{w}) = \mathbf{0}$. These algorithms were developed based on the specific form of linear classification or linear regression, so none of them would apply to logistic regression.

To train logistic regression, we will take an approach similar to linear regression in that we will try to set $\nabla E_{\text{in}}(\mathbf{w}) = \mathbf{0}$. Unfortunately, unlike the case of linear regression, the mathematical form of the gradient of E_{in} for logistic regression is not easy to manipulate, so an analytic solution is not feasible.

Exercise 3.7

For logistic regression, show that

$$\begin{aligned} \nabla E_{\text{in}}(\mathbf{w}) &= -\frac{1}{N} \sum_{n=1}^{N} \frac{y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{w}^{\mathrm{T}} \mathbf{x}_n}} \\ &= \frac{1}{N} \sum_{n=1}^{N} -y_n \mathbf{x}_n \theta(-y_n \mathbf{w}^{\mathrm{T}} \mathbf{x}_n). \end{aligned}$$

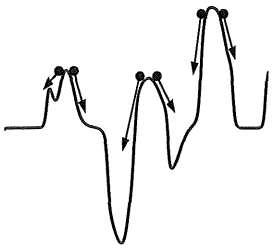
Argue that a 'misclassified' example contributes more to the gradient than a correctly classified one.

Instead of analytically setting the gradient to zero, we will *iteratively* set it to zero. To do so, we will introduce a new algorithm, *gradient descent*. Gradient

descent is a very general algorithm that can be used to train many other learning models with smooth error measures. For logistic regression, gradient descent has particularly nice properties.

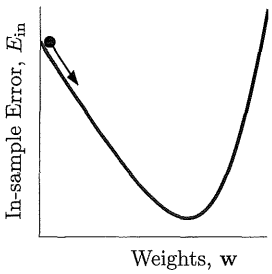
3.3.2 Gradient Descent

Gradient descent is a general technique for minimizing a twice-differentiable function, such as $E_{\text{in}}(\mathbf{w})$ in logistic regression. A useful physical analogy of gradient descent is a ball rolling down a hilly surface. If the ball is placed on a hill, it will roll down, coming to rest at the bottom of a valley. The same basic idea underlies gradient descent. $E_{\text{in}}(\mathbf{w})$ is a 'surface' in a high-dimensional space. At step 0, we start somewhere on this surface, at $\mathbf{w}(0)$, and try to roll down this surface, thereby decreasing E_{in}.



One thing which you immediately notice from the physical analogy is that the ball will not necessarily come to rest in the lowest valley of the entire surface. Depending on where you start the ball rolling, you will end up at the bottom of one of the valleys – a *local minimum*. In general, the same applies to gradient descent. Depending on your starting weights, the path of descent will take you to a local minimum in the error surface.

A particular advantage for logistic regression with the cross-entropy error is that the picture looks much nicer. There is only one valley! So, it does not matter where you start your ball rolling, it will always roll down to the same (unique) *global minimum*. This is a consequence of the fact that $E_{\text{in}}(\mathbf{w})$ is a *convex* function of $\mathbf{w}$, a mathematical property that implies a single 'valley' as shown to the right. This means that gradient descent will not be trapped in local minima when minimizing such convex error measures.[4]



Let's now determine how to 'roll' down the E_{in}-surface. We would like to take a step in the direction of steepest descent, to gain the biggest bang for our buck. Suppose that we take a small step of size η in the direction of a unit vector $\hat{\mathbf{v}}$. The new weights are $\mathbf{w}(0) + \eta\hat{\mathbf{v}}$. Since η is small, using the Taylor expansion to first order, we compute the change in E_{in} as

$$\begin{aligned} \Delta E_{\text{in}} &= E_{\text{in}}(\mathbf{w}(0) + \eta\hat{\mathbf{v}}) - E_{\text{in}}(\mathbf{w}(0)) \\ &= \eta \nabla E_{\text{in}}(\mathbf{w}(0))^{\text{T}}\hat{\mathbf{v}} + O(\eta^2) \\ &\geq -\eta \|\nabla E_{\text{in}}(\mathbf{w}(0))\|, \end{aligned}$$

[4] In fact, the squared in-sample error in linear regression is also convex, which is why the analytic solution found by the pseudo-inverse is guaranteed to have optimal in-sample error.

where we have ignored the small term $O(\eta^2)$. Since $\hat{\mathbf{v}}$ is a unit vector, equality holds if and only if

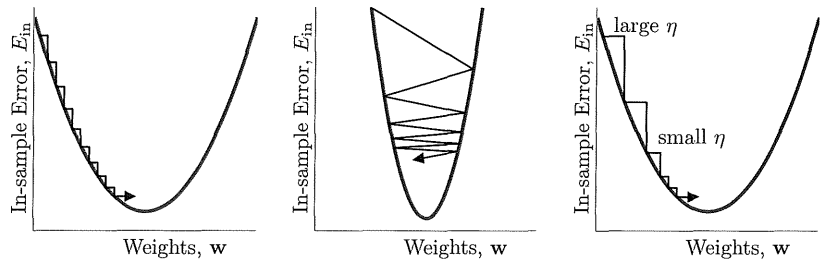
$$\hat{\mathbf{v}} = -\frac{\nabla E_{\text{in}}(\mathbf{w}(0))}{\|\nabla E_{\text{in}}(\mathbf{w}(0))\|}. \tag{3.10}$$

This direction, specified by $\hat{\mathbf{v}}$, leads to the largest decrease in E_{in} for a given step size η.

Exercise 3.8

The claim that $\hat{\mathbf{v}}$ is the direction which gives largest decrease in E_{in} only holds for small η. Why?

There is nothing to prevent us from continuing to take steps of size η, re-evaluating the direction $\hat{\mathbf{v}}_t$ at each iteration $t = 0, 1, 2, \ldots$. How large a step should one take at each iteration? This is a good question, and to gain some insight, let's look at the following examples.



η too small η too large variable η – just right

A fixed step size (if it is too small) is inefficient when you are far from the local minimum. On the other hand, too large a step size when you are close to the minimum leads to bouncing around, possibly even increasing E_{in}. Ideally, we would like to take large steps when far from the minimum to get in the right ballpark quickly, and then small (more careful) steps when close to the minimum. A simple heuristic can accomplish this: far from the minimum, the norm of the gradient is typically large, and close to the minimum, it is small. Thus, we could set $\eta_t = \eta\|\nabla E_{\text{in}}\|$ to obtain the desired behavior for the variable step size; choosing the step size proportional to the norm of the gradient will also conveniently cancel the term normalizing the unit vector $\hat{\mathbf{v}}$ in Equation (3.10), leading to the *fixed learning rate gradient descent* algorithm for minimizing E_{in} (with redefined η):

Fixed learning rate gradient descent:

1: Initialize the weights at time step $t = 0$ to $\mathbf{w}(0)$.
2: **for** $t = 0, 1, 2, \ldots$ **do**
3: Compute the gradient $\mathbf{g}_t = \nabla E_{\text{in}}(\mathbf{w}(t))$.
4: Set the direction to move, $\mathbf{v}_t = -\mathbf{g}_t$.
5: Update the weights: $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \mathbf{v}_t$.
6: Iterate to the next step until it is time to stop.
7: Return the final weights.

In the algorithm, $\mathbf{v}_t$ is a direction that is no longer restricted to unit length. The parameter η (the *learning rate*) has to be specified. A typically good choice for η is around 0.1 (a purely practical observation). To use gradient descent, one must compute the gradient. This can be done explicitly for logistic regression (see Exercise 3.7).

Example 3.3. Gradient descent is a general algorithm for minimizing twice-differentiable functions. We can apply it to the logistic regression in-sample error to return weights that approximately minimize

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

Logistic regression algorithm:

1: Initialize the weights at time step $t = 0$ to $\mathbf{w}(0)$.
2: **for** $t = 0, 1, 2, \ldots$ **do**
3: Compute the gradient

$$\mathbf{g}_t = -\frac{1}{N} \sum_{n=1}^{N} \frac{y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{w}^{\mathsf{T}}(t) \mathbf{x}_n}}.$$

4: Set the direction to move, $\mathbf{v}_t = -\mathbf{g}_t$.
5: Update the weights: $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \mathbf{v}_t$.
6: Iterate to the next step until it is time to stop.
7: Return the final weights $\mathbf{w}$.

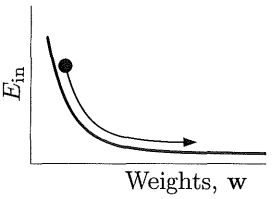
□

Initialization and termination. We have two more loose ends to tie: the first is how to choose $\mathbf{w}(0)$, the initial weights, and the second is how to set the criterion for "...until it is time to stop" in step 6 of the gradient descent algorithm. In some cases, such as logistic regression, initializing the weights $\mathbf{w}(0)$ as zeros works well. However, in general, it is safer to initialize the weights randomly, so as to avoid getting stuck on a perfectly symmetric hilltop. Choosing each weight independently from a Normal distribution with zero mean and small variance usually works well in practice.

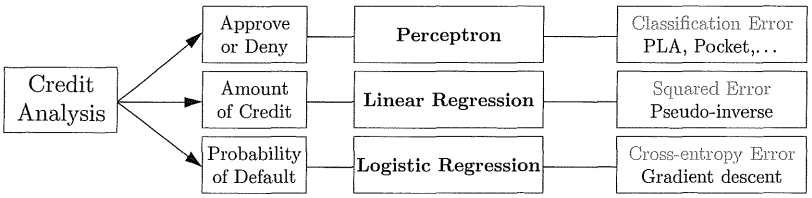
That takes care of initialization, so we now move on to termination. How do we decide when to stop? Termination is a non-trivial topic in optimization. One simple approach, as we encountered in the pocket algorithm, is to set an upper bound on the number of iterations, where the upper bound is typically in the thousands, depending on the amount of training time we have. The problem with this approach is that there is no guarantee on the quality of the final weights.

Another plausible approach is based on the gradient being zero at any minimum. A natural termination criterion would be to stop once $\|\mathbf{g}_t\|$ drops below a certain threshold. Eventually this must happen, but we do not know when it will happen. For logistic regression, a combination of the two conditions (setting a large upper bound for the number of iterations, and a small lower bound for the size of the gradient) usually works well in practice.

There is a problem with relying solely on the size of the gradient to stop, which is that you might stop prematurely as illustrated on the right. When the iteration reaches a relatively flat region (which is more common than you might suspect), the algorithm will prematurely stop when we may want to continue. So one solution is to require that termination occurs only if the error change is small and the error itself is small. Ultimately a combination of termination criteria (a maximum number of iterations, marginal error improvement, coupled with small value for the error itself) works reasonably well.



Example 3.4. By way of summarizing linear models, we revisit our old friend the credit example. If the goal is to decide whether to approve or deny, then we are in the realm of classification; if you want to assign an amount of credit line, then linear regression is appropriate; if you want to predict the probability that someone will default, use logistic regression.



The three linear models have their respective goals, error measures, and algorithms. Nonetheless, they not only share similar sets of linear hypotheses, but are in fact related in other ways. We would like to point out one important relationship: Both logistic regression and linear regression can be used in linear classification. Here is how.

Logistic regression produces a final hypothesis $g(\mathbf{x})$ which is our estimate of $\mathbb{P}[y = +1 \mid \mathbf{x}]$. Such an estimate can easily be used for classification by

setting a threshold on $g(\mathbf{x})$; a natural threshold is $\frac{1}{2}$, which corresponds to classifying $+1$ if $+1$ is more likely. This choice for threshold corresponds to using the logistic regression weights as weights in the perceptron for classification. Not only can logistic regression weights be used for classification in this way, but they can also be used as a way to train the perceptron model. The perceptron learning problem (3.2) is a very hard combinatorial optimization problem. The convexity of E_{in} in logistic regression makes the optimization problem much easier to solve. Since the logistic function is a soft version of a hard threshold, the logistic regression weights should be good weights for classification using the perceptron.

A similar relationship exists between classification and linear regression. Linear regression can be used with any real-valued target function, which includes real values that are ± 1. If $\mathbf{w}_{\text{lin}}^{\text{T}}\mathbf{x}$ is fit to ± 1 values, $\text{sign}(\mathbf{w}_{\text{lin}}^{\text{T}}\mathbf{x})$ will likely agree with these values and make good classification predictions. In other words, the linear regression weights $\mathbf{w}_{\text{lin}}$, which are easily computed using the pseudo-inverse, are also an approximate solution for the perceptron model. The weights can be directly used for classification, or used as an initial condition for the pocket algorithm to give it a head start. □

Exercise 3.9

Consider pointwise error measures $\text{e}_{\text{class}}(s,y) = [\![y \neq \text{sign}(s)]\!]$, $\text{e}_{\text{sq}}(s,y) = (y-s)^2$, and $\text{e}_{\text{log}}(s,y) = \ln(1+\exp(-ys))$, where the signal $s = \mathbf{w}^{\text{T}}\mathbf{x}$.

(a) For $y = +1$, plot e_{class}, e_{sq} and $\frac{1}{\ln 2}\text{e}_{\text{log}}$ versus s, on the same plot.

(b) Show that $\text{e}_{\text{class}}(s,y) \leq \text{e}_{\text{sq}}(s,y)$, and hence that the classification error is upper bounded by the squared error.

(c) Show that $\text{e}_{\text{class}}(s,y) \leq \frac{1}{\ln 2}\text{e}_{\text{log}}(s,y)$, and, as in part (b), get an upper bound (up to a constant factor) using the logistic regression error.

These bounds indicate that minimizing the squared or logistic regression error should also decrease the classification error, which justifies using the weights returned by linear or logistic regression as approximations for classification.

Stochastic gradient descent. The version of gradient descent we have described so far is known as *batch* gradient descent – the gradient is computed for the error on the whole data set before a weight update is done. A sequential version of gradient descent known as *stochastic gradient descent* (SGD) turns out to be very efficient in practice. Instead of considering the full batch gradient on all N training data points, we consider a stochastic version of the gradient. First, pick a training data point $(\mathbf{x}_n, y_n)$ uniformly at random (hence the name 'stochastic'), and consider only the error on that data point

(in the case of logistic regression),

$$\mathrm{e}_n(\mathbf{w}) = \ln\left(1 + e^{-y_n \mathbf{w}^{\mathrm{T}} \mathbf{x}_n}\right).$$

The gradient of this single data point's error is used for the weight update in exactly the same way that the gradient was used in batch gradient descent. The gradient needed for the weight update of SGD is (see Exercise 3.7)

$$\nabla \mathrm{e}_n(\mathbf{w}) = \frac{-y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{w}^{\mathrm{T}} \mathbf{x}_n}},$$

and the weight update is $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla \mathrm{e}_n(\mathbf{w})$. Insight into why SGD works can be gained by looking at the expected value of the change in the weight (the expectation is with respect to the random point that is selected). Since n is picked uniformly at random from $\{1, \ldots, N\}$, the expected weight change is

$$-\eta \cdot \frac{1}{N} \sum_{n=1}^{N} \nabla \mathrm{e}_n(\mathbf{w}).$$

This is exactly the same as the deterministic weight change from the batch gradient descent weight update. That is, 'on average' the minimization proceeds in the right direction, but is a bit wiggly. In the long run, these random fluctuations cancel out. The computational cost is cheaper by a factor of N, though, since we compute the gradient for only one point per iteration, rather than for all N points as we do in batch gradient descent.

Notice that SGD is similar to PLA in that it decreases the error with respect to one data point at a time. Minimizing the error on one data point may interfere with the error on the rest of the data points that are not considered at that iteration. However, also similar to PLA, the interference cancels out on average as we have just argued.

Exercise 3.10

(a) Define an error for a single data point $(\mathbf{x}_n, y_n)$ to be

$$\mathrm{e}_n(\mathbf{w}) = \max(0, -y_n \mathbf{w}^{\mathrm{T}} \mathbf{x}_n).$$

Argue that PLA can be viewed as SGD on e_n with learning rate $\eta = 1$.

(b) For logistic regression with a very large $\mathbf{w}$, argue that minimizing E_{in} using SGD is similar to PLA. This is another indication that the logistic regression weights can be used as a good approximation for classification.

SGD is successful in practice, often beating the batch version and other more sophisticated algorithms. In fact, SGD was an important part of the algorithm that won the million-dollar Netflix competition, discussed in Section 1.1. It scales well to large data sets, and is naturally suited to online learning, where

a stream of data present themselves to the learning algorithm sequentially. The randomness introduced by processing one data point at a time can be a plus, helping the algorithm to avoid flat regions and local minima in the case of a complicated error surface. However, it is challenging to choose a suitable termination criterion for SGD. A good stopping criterion should consider the total error on all the data, which can be computationally demanding to evaluate at each iteration.

3.4 Nonlinear Transformation

All formulas for the linear model have used the sum

$$\mathbf{w}^{\mathrm{T}}\mathbf{x} = \sum_{i=0}^{d} w_i x_i \tag{3.11}$$

as the main quantity in computing the hypothesis output. This quantity is linear, not only in the x_i's but also in the w_i's. A closer inspection of the corresponding learning algorithms shows that *the linearity in w_i's* is the key property for deriving these algorithms; the x_i's are just constants as far as the algorithm is concerned. This observation opens the possibility for allowing nonlinear versions of x_i's while still remaining in the analytic realm of linear models, because the form of Equation (3.11) remains linear in the w_i parameters.

Consider the credit limit problem for instance. It makes sense that the 'years in residence' field would affect a person's credit since it is correlated with stability. However, it is less plausible that the credit limit would grow *linearly* with the number of years in residence. More plausibly, there is a threshold (say 1 year) below which the credit limit is affected negatively and another threshold (say 5 years) above which the credit limit is affected positively. If x_i is the input variable that measures years in residence, then two nonlinear 'features' derived from it, namely $[\![x_i < 1]\!]$ and $[\![x_i > 5]\!]$, would allow a linear formula to reflect the credit limit better.

We have already seen the use of features in the classification of handwritten digits, where intensity and symmetry features were derived from input pixels. Nonlinear transforms can be further applied to those features, as we will see shortly, creating more elaborate features and improving the performance. The scope of linear methods expands significantly when we represent the input by a set of appropriate features.

3.4.1 The $\mathcal{Z}$ Space

Consider the situation in Figure 3.1(b) where a linear classifier can't fit the data. By transforming the inputs x_1, x_2 in a nonlinear fashion, we will be able to separate the data with more complicated boundaries while still using the

simple PLA as a building block. Let's start by looking at the circle in Figure 3.5(a), which is a replica of the non-separable case in Figure 3.1(b). The circle represents the following equation:

$$x_1^2 + x_2^2 = 0.6.$$

That is, the nonlinear hypothesis $h(\mathbf{x}) = \text{sign}(-0.6 + x_1^2 + x_2^2)$ separates the data set perfectly. We can view the hypothesis as a *linear* one after applying a nonlinear transformation on $\mathbf{x}$. In particular, consider $z_0 = 1$, $z_1 = x_1^2$ and $z_2 = x_2^2$,

$$\begin{aligned} h(\mathbf{x}) &= \text{sign}\left(\underbrace{(-0.6)}_{\tilde{w}_0}\cdot\underbrace{1}_{z_0} + \underbrace{1}_{\tilde{w}_1}\cdot\underbrace{x_1^2}_{z_1} + \underbrace{1}_{\tilde{w}_2}\cdot\underbrace{x_2^2}_{z_2}\right) \\ &= \text{sign}\left(\underbrace{[\tilde{w}_0\ \tilde{w}_1\ \tilde{w}_2]}_{\tilde{\mathbf{w}}^T}\underbrace{\begin{bmatrix}1\\ z_1\\ z_2\end{bmatrix}}_{\mathbf{z}}\right). \end{aligned}$$

where the vector $\mathbf{z}$ is obtained from $\mathbf{x}$ through a nonlinear transform Φ,

$$\mathbf{z} = \Phi(\mathbf{x}).$$

We can plot the data in terms of $\mathbf{z}$ instead of $\mathbf{x}$, as depicted in Figure 3.5(b). For instance, the point $\mathbf{x}_1$ in Figure 3.5(a) is transformed to the point $\mathbf{z}_1$ in Figure 3.5(b) and the point $\mathbf{x}_2$ is transformed to the point $\mathbf{z}_2$. The space $\mathcal{Z}$, which contains the $\mathbf{z}$ vectors, is referred to as the *feature space* since its coordinates are higher-level features derived from the raw input $\mathbf{x}$. We designate different quantities in $\mathcal{Z}$ with a tilde version of their counterparts in $\mathcal{X}$, e.g., the dimensionality of $\mathcal{Z}$ is $\tilde{d}$ and the weight vector is $\tilde{\mathbf{w}}$.[5] The transform Φ that takes us from $\mathcal{X}$ to $\mathcal{Z}$ is called a *feature transform*, which in this case is

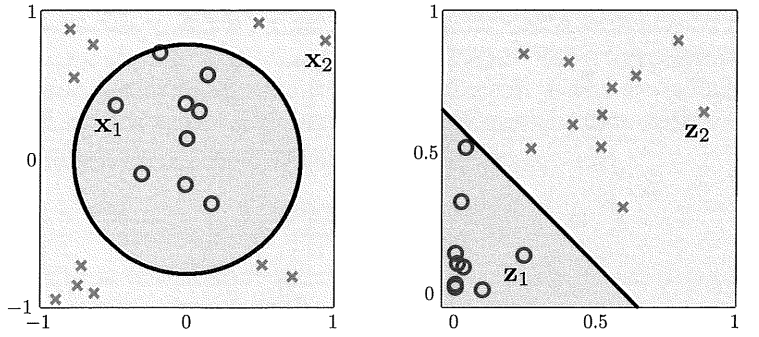
$$\Phi(\mathbf{x}) = (1, x_1^2, x_2^2). \tag{3.12}$$

In general, some points in the $\mathcal{Z}$ space may not be valid transforms of any $\mathbf{x} \in \mathcal{X}$, and multiple points in $\mathcal{X}$ may be transformed to the same $\mathbf{z} \in \mathcal{Z}$, depending on the nonlinear transform Φ.

The usefulness of the transform above is that the nonlinear hypothesis h (circle) in the $\mathcal{X}$ space can be represented by a linear hypothesis (line) in the $\mathcal{Z}$ space. Indeed, any linear hypothesis $\tilde{h}$ in $\mathbf{z}$ corresponds to a (possibly nonlinear) hypothesis of $\mathbf{x}$ given by

$$h(\mathbf{x}) = \tilde{h}(\Phi(\mathbf{x})).$$

[5] $\mathcal{Z} = \{1\} \times \mathbb{R}^{\tilde{d}}$, where $\tilde{d} = 2$ in this case. We treat $\mathcal{Z}$ as $\tilde{d}$-dimensional since the added coordinate $z_0 = 1$ is fixed.



(a) Data in $\mathcal{X}$-space (b) Transformed data in $\mathcal{Z}$-space

$$\mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \qquad \mathbf{z} = \Phi(\mathbf{x}) = \begin{bmatrix} 1 \\ x_1^2 \\ x_2^2 \end{bmatrix}$$

Figure 3.5: (a) The original data set that is not linearly separable, but separable by a circle. (b) The transformed data set that is linearly separable in the $\mathcal{Z}$ space. In the figure, $\mathbf{x}_1$ maps to $\mathbf{z}_1$ and $\mathbf{x}_2$ maps to $\mathbf{z}_2$; the circular separator in the $\mathcal{X}$-space maps to the linear separator in the $\mathcal{Z}$-space.

The set of these hypotheses h is denoted by $\mathcal{H}_\Phi$. For instance, when using the feature transform in (3.12), each $h \in \mathcal{H}_\Phi$ is a quadratic curve in $\mathcal{X}$ that corresponds to some line $\tilde{h}$ in $\mathcal{Z}$.

Exercise 3.11

Consider the feature transform Φ in (3.12). What kind of boundary in $\mathcal{X}$ does a hyperplane $\tilde{\mathbf{w}}$ in $\mathcal{Z}$ correspond to in the following cases? Draw a picture that illustrates an example of each case.

(a) $\tilde{w}_1 > 0, \tilde{w}_2 < 0$

(b) $\tilde{w}_1 > 0, \tilde{w}_2 = 0$

(c) $\tilde{w}_1 > 0, \tilde{w}_2 > 0, \tilde{w}_0 < 0$

(d) $\tilde{w}_1 > 0, \tilde{w}_2 > 0, \tilde{w}_0 > 0$

Because the transformed data set $(\mathbf{z}_1, y_1), \cdots, (\mathbf{z}_N, y_N)$ in Figure 3.5(b) is linearly separable in the feature space $\mathcal{Z}$, we can apply PLA on the transformed data set to obtain $\tilde{\mathbf{w}}_{\text{PLA}}$, the PLA solution, which gives us a final hypothesis $g(\mathbf{x}) = \text{sign}(\tilde{\mathbf{w}}_{\text{PLA}}^{\text{T}}\mathbf{z})$ in the $\mathcal{X}$ space, where $\mathbf{z} = \Phi(\mathbf{x})$. The whole process of applying the feature transform before running PLA for linear classification is depicted in Figure 3.6.

The in-sample error in the input space $\mathcal{X}$ is the same as in the feature space $\mathcal{Z}$, so $E_{\text{in}}(g) = 0$. Hyperplanes that achieve $E_{\text{in}}(\tilde{\mathbf{w}}_{\text{PLA}}) = 0$ in $\mathcal{Z}$ correspond to separating curves in the original input space $\mathcal{X}$. For instance,

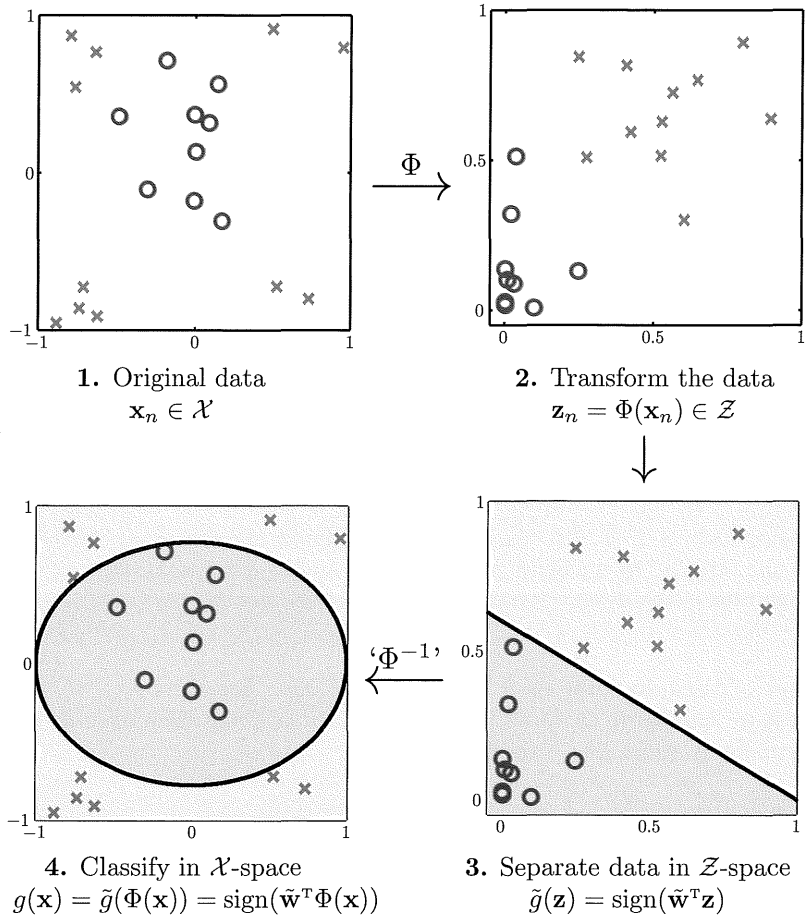


Figure 3.6: The nonlinear transform for separating non-separable data.

as shown in Figure 3.6, the PLA may select the line $\tilde{\mathbf{w}}_{\text{PLA}} = (-0.6, 0.6, 1)$ that separates the transformed data $(\mathbf{z}_1, y_1), \cdots, (\mathbf{z}_N, y_N)$. The corresponding hypothesis $g(\mathbf{x}) = \text{sign}(-0.6 + 0.6 \cdot x_1^2 + x_2^2)$ will separate the original data $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)$. In this case, the decision boundary is an ellipse in $\mathcal{X}$.

How does the feature transform affect the VC bound (3.1)? If we honestly decide on the transform Φ before seeing the data, then with probability at least $1 - \delta$, the bound (3.1) remains true by using $d_{\text{VC}}(\mathcal{H}_\Phi)$ as the VC dimension. For instance, consider the feature transform Φ in (3.12). We know that $\mathcal{Z} = \{1\} \times \mathbb{R}^2$. Since $\mathcal{H}_\Phi$ is the perceptron in $\mathcal{Z}$, $d_{\text{VC}}(\mathcal{H}_\Phi) \leq 3$ (the $\leq$ is because some points $\mathbf{z} \in \mathcal{Z}$ may not be valid transforms of any $\mathbf{x}$, so some dichotomies may not be realizable). We can then substitute N, $d_{\text{VC}}(\mathcal{H}_\Phi)$, and δ into the VC bound. After running PLA on the transformed data set, if we succeed in

getting some g with $E_{\text{in}}(g) = 0$, we can claim that g will perform well out of sample.

It is very important to understand that the claim above is valid only if you decide on Φ *before* seeing the data or trying any algorithms. What if we first try using lines to separate the data, fail, and then use the circles? Then we are effectively using a model that contains both lines and circles, and d_{VC} is no longer 3.

Exercise 3.12

We know that in the Euclidean plane, the perceptron model $\mathcal{H}$ cannot implement all 16 dichotomies on 4 points. That is, $m_{\mathcal{H}}(4) < 16$. Take the feature transform Φ in (3.12).

(a) Show that $m_{\mathcal{H}_\Phi}(3) = 8$.

(b) Show that $m_{\mathcal{H}_\Phi}(4) < 16$.

(c) Show that $m_{\mathcal{H} \cup \mathcal{H}_\Phi}(4) = 16$.

That is, if you used lines, $d_{\text{VC}} = 3$; if you used elipses, $d_{\text{VC}} = 3$; if you used lines and elipses, $d_{\text{VC}} > 3$.

Worse yet, if you actually *look* at the data (e.g., look at the points in Figure 3.1(a)) before deciding on a suitable Φ, you forfeit most of what you learned in Chapter 2 ☹. You have inadvertently explored a huge hypothesis space in your mind to come up with a specific Φ that would work *for this data set.* If you invoke a generalization bound now, you will be charged for the VC dimension of the full space that you explored in your mind, not just the space that Φ creates.

This does not mean that Φ should be chosen blindly. In the credit limit problem for instance, we suggested nonlinear features based on the 'years in residence' field that may be more suitable for linear regression than the raw input. This was based on our understanding of the problem, not on 'snooping' into the training data. Therefore, we pay no price in terms of generalization, and we may well gain a dividend in performance because of a good choice of features.

The feature transform Φ can be general, as long as it is chosen before seeing the data set (as if we cannot emphasize this enough). For instance, you may have noticed that the feature transform in (3.12) only allows us to get very limited types of quadratic curves. Ellipses that do not center at the origin in $\mathcal{X}$ cannot correspond to a hyperplane in $\mathcal{Z}$. To get all possible quadratic curves in $\mathcal{X}$, we could consider the more general feature transform $\mathbf{z} = \Phi_2(\mathbf{x})$,

$$\Phi_2(\mathbf{x}) = (1, x_1, x_2, x_1^2, x_1 x_2, x_2^2), \tag{3.13}$$

which gives us the flexibility to represent any quadratic curve in $\mathcal{X}$ by a hyperplane in $\mathcal{Z}$ (the subscript 2 of Φ is for polynomials of degree 2 - quadratic curves). The price we pay is that $\mathcal{Z}$ is now five-dimensional instead of two-dimensional, and hence d_{VC} is doubled from 3 to 6.

Exercise 3.13

Consider the feature transform $\mathbf{z} = \Phi_2(\mathbf{x})$ in (3.13). How can we use a hyperplane $\tilde{\mathbf{w}}$ in $\mathcal{Z}$ to represent the following boundaries in $\mathcal{X}$?

(a) The parabola $(x_1 - 3)^2 + x_2 = 1$.

(b) The circle $(x_1 - 3)^2 + (x_2 - 4)^2 = 1$.

(c) The ellipse $2(x_1 - 3)^2 + (x_2 - 4)^2 = 1$.

(d) The hyperbola $(x_1 - 3)^2 - (x_2 - 4)^2 = 1$.

(e) The ellipse $2(x_1 + x_2 - 3)^2 + (x_1 - x_2 - 4)^2 = 1$.

(f) The line $2x_1 + x_2 = 1$.

One may further extend Φ_2 to a feature transform Φ_3 for cubic curves in $\mathcal{X}$, or more generally define the feature transform Φ_Q for degree-Q curves in $\mathcal{X}$. The feature transform Φ_Q is called the *Qth order polynomial transform.*

The power of the feature transform should be used with care. It may not be worth it to insist on linear separability and employ a highly complex surface to achieve that. Consider the case of Figure 3.1(a). If we insist on a feature transform that linearly separates the data, it may lead to a significant increase of the VC dimension. As we see in Figure 3.7, no line can separate the training examples perfectly, and neither can any quadratic nor any third-order polynomial curves. Thus, we need to use a fourth-order polynomial transform:

$$\Phi_4(\mathbf{x}) = (1, x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, x_1^2x_2, x_1x_2^2, x_2^3, x_1^4, x_1^3x_2, x_1^2x_2^2, x_1x_2^3, x_2^4).$$

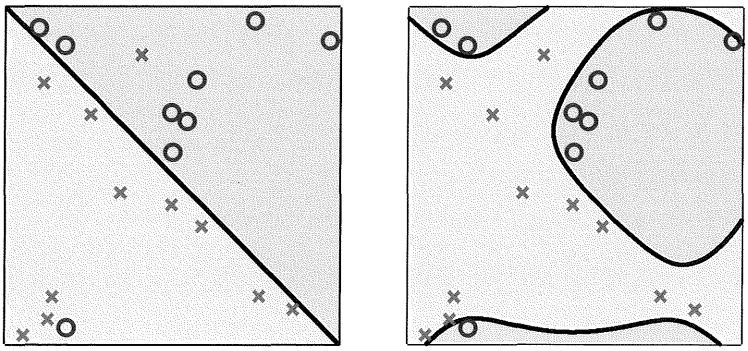
If you look at the fourth-order decision boundary in Figure 3.7(b), you don't need the VC analysis to tell you that this is an overkill that is unlikely to generalize well to new data. A better option would have been to ignore the two misclassified examples in Figure 3.7(a), separate the other examples perfectly with the line, and accept the small but nonzero E_{in}. Indeed, sometimes our best bet is to go with a simpler hypothesis set while tolerating a small E_{in}.

While our discussion of feature transforms has focused on classification problems, these transforms can be applied equally to regression problems. Both linear regression and logistic regression can be implemented in the feature space $\mathcal{Z}$ instead of the input space $\mathcal{X}$. For instance, linear regression is often coupled with a feature transform to perform nonlinear regression. The N by $d + 1$ input matrix X in the algorithm is replaced with the N by $\tilde{d} + 1$ matrix Z, while the output vector $\mathbf{y}$ remains the same.

3.4.2 Computation and Generalization

Although using a larger Q gives us more flexibility in terms of the shape of decision boundaries in $\mathcal{X}$, there is a price to be paid. Computation is one issue, and generalization is the other.

Computation is an issue because the feature transform Φ_Q maps a two-dimensional vector $\mathbf{x}$ to $\tilde{d} = \frac{Q(Q+3)}{2}$ dimensions, which increases the memory



(a) Linear fit (b) 4th order polynomial fit

Figure 3.7: Illustration of the nonlinear transform using a data set that is not linearly separable; (a) a line separates the data after omitting a few points, (b) a fourth-order polynomial separates all the points.

and computational costs. Things could get worse if $\mathbf{x}$ is in a higher dimension to begin with.

Exercise 3.14

Consider the Qth order polynomial transform Φ_Q for $\mathcal{X} = \mathbb{R}^d$. What is the dimensionality $\tilde{d}$ of the feature space $\mathcal{Z}$ (excluding the fixed coordinate $z_0 = 1$). Evaluate your result on $d \in \{2, 3, 5, 10\}$ and $Q \in \{2, 3, 5, 10\}$.

The other important issue is generalization. If Φ_Q is the feature transform of a two-dimensional input space, there will be $\tilde{d} = \frac{Q(Q+3)}{2}$ dimensions in $\mathcal{Z}$, and $d_{\text{VC}}(\mathcal{H}_\Phi)$ can be as high as $\frac{Q(Q+3)}{2} + 1$. This means that the second term in the VC bound (3.1) can grow significantly. In other words, we would have a weaker guarantee that E_{out} will be small. For instance, if we use $\Phi = \Phi_{50}$, the VC dimension of $\mathcal{H}_\Phi$ could be as high as $\frac{(50)(53)}{2} + 1 = 1326$ instead of the original $d_{\text{VC}} = 3$. Applying the rule of thumb that the amount of data needed is proportional to the VC dimension, we would need hundreds of times more data than we would if we didn't use a feature transform, in order to achieve the same level of generalization error.

Exercise 3.15

High-dimensional feature transforms are by no means the only transforms that we can use. We can take the tradeoff in the other direction, and use low-dimensional feature transforms as well (to achieve an even lower generalization error bar).

(continued on next page)

Consider the following feature transform, which maps a d-dimensional $\mathbf{x}$ to a one-dimensional $\mathbf{z}$, keeping only the kth coordinate of $\mathbf{x}$.

$$\Phi_{(k)}(\mathbf{x}) = (1, x_k). \tag{3.14}$$

Let $\mathcal{H}_k$ be the set of perceptrons in the feature space.

(a) Prove that $d_{\text{vc}}(\mathcal{H}_k) = 2$.

(b) Prove that $d_{\text{vc}}(\cup_{k=1}^{d} \mathcal{H}_k) \leq 2(\log_2 d + 1)$.

$\mathcal{H}_k$ is called the *decision stump* model on dimension k.

The problem of generalization when we go to high-dimensional space is sometimes balanced by the advantage we get in approximating the target better. As we have seen in the case of using quadratic curves instead of lines, the transformed data became linearly separable, reducing E_{in} to 0. In general, when choosing the appropriate dimension for the feature transform, we cannot avoid the approximation-generalization tradeoff,

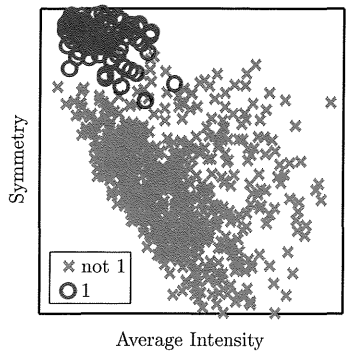
higher $\tilde{d}$	better chance of being linearly separable ($E_{\text{in}} \downarrow$)	$d_{\text{vc}} \uparrow$
lower $\tilde{d}$	possibly not linearly separable ($E_{\text{in}} \uparrow$)	$d_{\text{vc}} \downarrow$

Therefore, choosing a feature transform before seeing the data is a non-trivial task. When we apply learning to a particular problem, some understanding of the problem can help in choosing features that work well. More generally, there are some guidelines for choosing a suitable transform, or a suitable model, which we will discuss in Chapter 4.

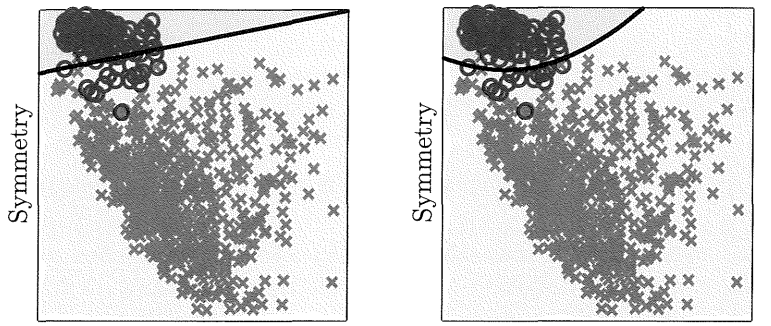
Exercise 3.16

Write down the steps of the algorithm that combines Φ_3 with linear regression. How about using Φ_{10} instead? Where is the main computational bottleneck of the resulting algorithm?

Example 3.5. Let's revisit the handwritten digit recognition example. We can try a different way of decomposing the big task of separating ten digits to smaller tasks. One decomposition is to separate digit 1 from all the other digits. Using intensity and symmetry as our input variables like we did before, the scatter plot of the training data is shown next. A line can roughly separate digit 1 from the rest, but a more complicated curve might do better.



We use linear regression (for classification), first without any feature transform. The results are shown below (LHS). We get $E_{\text{in}} = 2.13\%$ and $E_{\text{out}} = 2.38\%$.



Linear model
$E_{\text{in}} = 2.13\%$
$E_{\text{out}} = 2.38\%$

3rd order polynomial model
$E_{\text{in}} = 1.75\%$
$E_{\text{out}} = 1.87\%$

Classification of the digits data ('1' versus 'not 1') using linear and third order polynomial models.

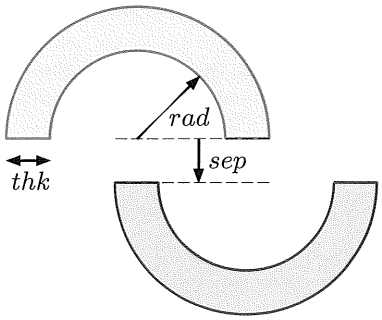
When we run linear regression with Φ_3, the third-order polynomial transform, we obtain a better fit to the data, with a lower $E_{\text{in}} = 1.75\%$. The result is depicted in the RHS of the figure. In this case, the better in-sample fit also resulted in a better out-of-sample performance, with $E_{\text{out}} = 1.87\%$. □

Linear models, a final pitch. The linear model (for classification or regression) is an often overlooked resource in the arena of learning from data. Since efficient learning algorithms exist for linear models, they are low overhead. They are also very robust and have good generalization properties. A sound

policy to follow when learning from data is to *first* try a linear model. Because of the good generalization properties of linear models, not much can go wrong. If you get a good fit to the data (low E_{in}), then you are done. If you do not get a good enough fit to the data and decide to go for a more complex model, you will pay a price in terms of the VC dimension as we have seen in Exercise 3.12, but the price is modest.

3.5 Problems

Problem 3.1 Consider the double semi-circle "toy" learning task below.



There are two semi-circles of width thk with inner radius rad, separated by sep as shown (red is -1 and blue is $+1$). The center of the top semi-circle is aligned with the middle of the edge of the bottom semi-circle. This task is linearly separable when $sep \geq 0$, and not so for $sep < 0$. Set $rad = 10$, $thk = 5$ and $sep = 5$. Then, generate $2,000$ examples uniformly, which means you will have approximately $1,000$ examples for each class.

(a) Run the PLA starting from $\mathbf{w} = \mathbf{0}$ until it converges. Plot the data and the final hypothesis.

(b) Repeat part (a) using the linear regression (for classification) to obtain $\mathbf{w}$. Explain your observations.

Problem 3.2 For the double-semi-circle task in Problem 3.1, vary sep in the range $\{0.2, 0.4, \ldots, 5\}$. Generate $2,000$ examples and run the PLA starting with $\mathbf{w} = \mathbf{0}$. Record the number of iterations PLA takes to converge.

Plot sep versus the number of iterations taken for PLA to converge. Explain your observations. *[Hint: Problem 1.3.]*

Problem 3.3 For the double-semi-circle task in Problem 3.1, set $sep = -5$ and generate $2,000$ examples.

(a) What will happen if you run PLA on those examples?

(b) Run the pocket algorithm for $100,000$ iterations and plot E_{in} versus the iteration number t.

(c) Plot the data and the final hypothesis in part (b).

(continued on next page)

(d) Use the linear regression algorithm to obtain the weights $\mathbf{w}$, and compare this result with the pocket algorithm in terms of computation time and quality of the solution.

(e) Repeat (b) – (d) with a 3rd order polynomial feature transform.

Problem 3.4 In Problem 1.5, we introduced the Adaptive Linear Neuron (Adaline) algorithm for classification. Here, we derive Adaline from an optimization perspective.

(a) Consider $E_n(\mathbf{w}) = (\max(0, 1 - y_n \mathbf{w}^{\mathrm{T}} \mathbf{x}_n))^2$. Show that $E_n(\mathbf{w})$ is continuous and differentiable. Write down the gradient $\nabla E_n(\mathbf{w})$.

(b) Show that $E_n(\mathbf{w})$ is an upper bound for $[\![\mathrm{sign}(\mathbf{w}^{\mathrm{T}} \mathbf{x}_n) \neq y_n]\!]$. Hence, $\frac{1}{N} \sum_{n=1}^{N} E_n(\mathbf{w})$ is an upper bound for the in-sample classification error $E_{\mathrm{in}}(\mathbf{w})$.

(c) Argue that the Adaline algorithm in Problem 1.5 performs stochastic gradient descent on $\frac{1}{N} \sum_{n=1}^{N} E_n(\mathbf{w})$.

Problem 3.5

(a) Consider

$$E_n(\mathbf{w}) = \max(0, 1 - y_n \mathbf{w}^{\mathrm{T}} \mathbf{x}_n).$$

Show that $E_n(\mathbf{w})$ is continuous and differentiable except when $y_n = \mathbf{w}^{\mathrm{T}} \mathbf{x}_n$.

(b) Show that $E_n(\mathbf{w})$ is an upper bound for $[\![\mathrm{sign}(\mathbf{w}^{\mathrm{T}} \mathbf{x}_n) \neq y_n]\!]$. Hence, $\frac{1}{N} \sum_{n=1}^{N} E_n(\mathbf{w})$ is an upper bound for the in-sample classification error $E_{\mathrm{in}}(\mathbf{w})$.

(c) Apply stochastic gradient descent on $\frac{1}{N} \sum_{n=1}^{N} E_n(\mathbf{w})$ (ignoring the singular case of $\mathbf{w}^{\mathrm{T}} \mathbf{x}_n = y_n$) and derive a new perceptron learning algorithm.

Problem 3.6 Derive a linear programming algorithm to fit a linear model for classification using the following steps. A linear program is an optimization problem of the following form:

$$\begin{aligned} \min_{\mathbf{z}} \quad & \mathbf{c}^{\mathrm{T}} \mathbf{z} \\ \text{subject to} \quad & \mathrm{A}\mathbf{z} \leq \mathbf{b}. \end{aligned}$$

A, $\mathbf{b}$ and $\mathbf{c}$ are parameters of the linear program and $\mathbf{z}$ is the optimization variable. This is such a well studied optimization problem that most mathematics software have canned optimization functions which solve linear programs.

(a) For linearly separable data, show that for some $\mathbf{w}$, $y_n(\mathbf{w}^{\mathrm{T}} \mathbf{x}_n) \geq 1$ for $n = 1, \ldots, N$.

(b) Formulate the task of finding a separating $\mathbf{w}$ for separable data as a linear program. You need to specify what the parameters $\mathrm{A}, \mathbf{b}, \mathbf{c}$ are and what the optimization variable $\mathbf{z}$ is.

(c) If the data is not separable, the condition in (a) cannot hold for every n. Thus introduce the violation $\xi_n \geq 0$ to capture the amount of violation for example $\mathbf{x}_n$. So, for $n = 1, \ldots, N$,

$$\begin{aligned} y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n) &\geq 1 - \xi_n, \\ \xi_n &\geq 0. \end{aligned}$$

Naturally, we would like to minimize the amount of violation. One intuitive approach is to minimize $\sum_{n=1}^{N} \xi_n$, i.e., we want $\mathbf{w}$ that solves

$$\begin{aligned} \min_{\mathbf{w},\xi_n} \quad & \sum_{n=1}^{N} \xi_n \\ \text{subject to} \quad & y_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n) \geq 1 - \xi_n, \\ & \xi_n \geq 0, \end{aligned}$$

where the inequalities must hold for $n = 1, \ldots, N$. Formulate this problem as a linear program.

(d) Argue that the linear program you derived in (c) and the optimization problem in Problem 3.5 are equivalent.

Problem 3.7 Use the linear programming algorithm from Problem 3.6 on the learning task in Problem 3.1 for the separable ($sep = 5$) and the non-separable ($sep = -5$) cases.

Compare your results to the linear regression approach with and without the 3rd order polynomial feature transform.

Problem 3.8 For linear regression, the out-of-sample error is

$$E_{\mathrm{out}}(h) = \mathbb{E}\left[(h(\mathbf{x}) - y)^2\right].$$

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

$$h^*(\mathbf{x}) = \mathbb{E}[y \mid \mathbf{x}].$$

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

Problem 3.9 Assuming that $\mathrm{X}^{\mathrm{T}}\mathrm{X}$ is invertible, show by direct comparison with Equation (3.4) that $E_{\mathrm{in}}(\mathbf{w})$ can be written as

$$E_{\mathrm{in}}(\mathbf{w}) = (\mathbf{w} - (\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathrm{T}}\mathbf{y})^{\mathrm{T}}(\mathrm{X}^{\mathrm{T}}\mathrm{X})(\mathbf{w} - (\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathrm{T}}\mathbf{y}) + \mathbf{y}^{\mathrm{T}}(\mathrm{I} - \mathrm{X}(\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathrm{T}})\mathbf{y}.$$

Use this expression for E_{in} to obtain $\mathbf{w}_{\mathrm{lin}}$. What is the in-sample error? *[Hint: The matrix* $\mathrm{X}^{\mathrm{T}}\mathrm{X}$ *is positive definite.]*

Problem 3.10 Exercise 3.3 studied some properties of the hat matrix $\mathrm{H} = \mathrm{X}(\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathrm{T}}$, where X is a N by $d+1$ matrix, and $\mathrm{X}^{\mathrm{T}}\mathrm{X}$ is invertible. Show the following additional properties.

(a) Every eigenvalue of H is either 0 or 1. *[Hint: Exercise 3.3(b).]*

(b) Show that the trace of a symmetric matrix equals the sum of its eigenvalues. *[Hint: Use the spectral theorem and the cyclic property of the trace. Note that the same result holds for non-symmetric matrices, but is a little harder to prove.]*

(c) How many eigenvalues of H are 1? What is the rank of H? *[Hint: Exercise 3.3(d).]*

Problem 3.11 Consider the linear regression problem setup in Exercise 3.4, where the data comes from a genuine linear relationship with added noise. The noise for the different data points is assumed to be iid with zero mean and variance σ^2. Assume that the 2nd moment matrix $\Sigma = \mathbb{E}_{\mathbf{x}}[\mathbf{x}\mathbf{x}^{\mathrm{T}}]$ is non-singular. Follow the steps below to show that, with high probability, the out-of-sample error on average is

$$E_{\mathrm{out}}(\mathbf{w}_{\mathrm{lin}}) = \sigma^2\left(1 + \frac{d+1}{N} + o(\tfrac{1}{N})\right).$$

(a) For a test point $\mathbf{x}$, show that the error $y - g(\mathbf{x})$ is

$$\epsilon - \mathbf{x}^{\mathrm{T}}(\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathrm{T}}\boldsymbol{\epsilon},$$

where ϵ is the noise realization for the test point and $\boldsymbol{\epsilon}$ is the vector of noise realizations on the data.

(b) Take the expectation with respect to the test point, i.e., $\mathbf{x}$ and ϵ, to obtain an expression for E_{out}. Show that

$$E_{\mathrm{out}} = \sigma^2 + \mathrm{trace}\left(\Sigma(\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathrm{T}}\boldsymbol{\epsilon}\boldsymbol{\epsilon}^{\mathrm{T}}\mathrm{X}^{\mathrm{T}}(\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1}\right).$$

[Hints: $a = \mathrm{trace}(a)$ *for any scalar* a; $\mathrm{trace}(\mathrm{AB}) = \mathrm{trace}(\mathrm{BA})$; *expectation and trace commute.]*

(c) What is $\mathbb{E}_{\boldsymbol{\epsilon}}[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^{\mathrm{T}}]$?

(d) Take the expectation with respect to ϵ to show that, on average,

$$E_{\text{out}} = \sigma^2 + \frac{\sigma^2}{N}\text{trace}\left(\Sigma(\tfrac{1}{N}\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1}\right).$$

Note that $\frac{1}{N}\mathrm{X}^{\mathrm{T}}\mathrm{X} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n\mathbf{x}_n^{\mathrm{T}}$ is an N-sample estimate of Σ. So $\frac{1}{N}\mathrm{X}^{\mathrm{T}}\mathrm{X} \approx \Sigma$. If $\frac{1}{N}\mathrm{X}^{\mathrm{T}}\mathrm{X} = \Sigma$, then what is E_{out} on average?

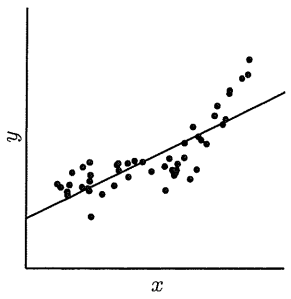
(e) Show that (after taking the expectation over the data noise) with high probability,

$$E_{\text{out}} = \sigma^2\left(1 + \frac{d+1}{N} + o(\tfrac{1}{N})\right).$$

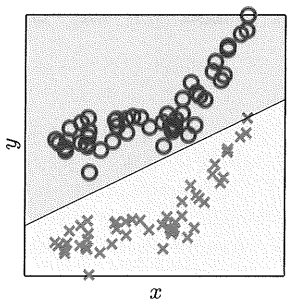
[Hint: By the law of large numbers $\frac{1}{N}\mathrm{X}^{\mathrm{T}}\mathrm{X}$ converges in probability to Σ, and so by continuity of the inverse at Σ, $\left(\frac{1}{N}\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1}$ converges in probability to Σ^{-1}.]

Problem 3.12 In linear regression, the in-sample predictions are given by $\hat{\mathbf{y}} = \mathrm{H}\mathbf{y}$, where $\mathrm{H} = \mathrm{X}(\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathrm{T}}$. Show that H is a projection matrix, i.e. $\mathrm{H}^2 = \mathrm{H}$. So $\hat{\mathbf{y}}$ is the projection of $\mathbf{y}$ onto some space. What is this space?

Problem 3.13 This problem creates a linear regression algorithm from a good algorithm for linear classification. As illustrated, the idea is to take the original data and shift it in one direction to get the $+1$ data points; then, shift it in the opposite direction to get the -1 data points.



Original data for the one-dimensional regression problem



Shifted data viewed as a two-dimensional classification problem

More generally, The data $(\mathbf{x}_n, y_n)$ can be viewed as data points in $\mathbb{R}^{d+1}$ by treating the y-value as the $(d+1)$th coordinate.

(continued on next page)

Now, construct positive and negative points

$$\begin{aligned}\mathcal{D}_+ &= (\mathbf{x}_1, y_1) + \mathbf{a}, \ldots, (\mathbf{x}_N, y_N) + \mathbf{a} \\ \mathcal{D}_- &= (\mathbf{x}_1, y_1) - \mathbf{a}, \ldots, (\mathbf{x}_N, y_N) - \mathbf{a},\end{aligned}$$

where $\mathbf{a}$ is a perturbation parameter. You can now use the linear programming algorithm in Problem 3.6 to separate $\mathcal{D}_+$ from $\mathcal{D}_-$. The resulting separating hyperplane can be used as the regression 'fit' to the original data.

(a) How many weights are learned in the classification problem? How many weights are needed for the linear fit in the regression problem?

(b) The linear fit requires weights $\mathbf{w}$, where $h(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}$. Suppose the weights returned by solving the classification problem are $\mathbf{w}_{\text{class}}$. Derive an expression for $\mathbf{w}$ as a function of $\mathbf{w}_{\text{class}}$.

(c) Generate a data set $y_n = x_n^2 + \sigma\epsilon_n$ with $N = 50$, where x_n is uniform on $[0, 1]$ and ϵ_n is zero mean Gaussian noise; set $\sigma = 0.1$. Plot $\mathcal{D}_+$ and $\mathcal{D}_-$ for $\mathbf{a} = \begin{bmatrix} 0 \\ 0.1 \end{bmatrix}$.

(d) Give comparisons of the resulting fits from running the classification approach and the analytic pseudo-inverse algorithm for linear regression.

Problem 3.14 In a regression setting, assume the target function is linear, so $f(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{w}_f$, and $\mathbf{y} = \mathrm{X}\mathbf{w}_f + \boldsymbol{\epsilon}$, where the entries in $\boldsymbol{\epsilon}$ are zero mean, iid with variance σ^2. In this problem derive the bias and variance as follows.

(a) Show that the average function is $\bar{g}(\mathbf{x}) = f(\mathbf{x})$, no matter what the size of the data set, as long as $\mathrm{X}^{\mathrm{T}}\mathrm{X}$ is invertible. What is the bias?

(b) What is the variance? *[Hint: Problem 3.11]*

Problem 3.15 In the text we derived that the linear regression solution weights must satisfy $\mathrm{X}^{\mathrm{T}}\mathrm{X}\mathbf{w} = \mathrm{X}^{\mathrm{T}}\mathbf{y}$. If $\mathrm{X}^{\mathrm{T}}\mathrm{X}$ is not invertible, the solution $\mathbf{w}_{\text{lin}} = (\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathrm{T}}\mathbf{y}$ won't work. In this event, there will be many solutions for $\mathbf{w}$ that minimize E_{in}. Here, you will derive one such solution. Let ρ be the rank of X. Assume that the singular value decomposition (SVD) of X is $\mathrm{X} = \mathrm{U}\Gamma\mathrm{V}^{\mathrm{T}}$, where $\mathrm{U} \in \mathbb{R}^{N\times\rho}$ satisfies $\mathrm{U}^{\mathrm{T}}\mathrm{U} = \mathrm{I}_\rho$, $\mathrm{V} \in \mathbb{R}^{(d+1)\times\rho}$ satisfies $\mathrm{V}^{\mathrm{T}}\mathrm{V} = \mathrm{I}_\rho$, and $\Gamma \in \mathbb{R}^{\rho\times\rho}$ is a positive diagonal matrix.

(a) Show that $\rho < d + 1$.

(b) Show that $\mathbf{w}_{\text{lin}} = \mathrm{V}\Gamma^{-1}\mathrm{U}^{\mathrm{T}}\mathbf{y}$ satisfies $\mathrm{X}^{\mathrm{T}}\mathrm{X}\mathbf{w}_{\text{lin}} = \mathrm{X}^{\mathrm{T}}\mathbf{y}$, and hence is a solution.

(c) Show that for any other solution that satisfies $\mathrm{X}^{\mathrm{T}}\mathrm{X}\mathbf{w} = \mathrm{X}^{\mathrm{T}}\mathbf{y}$, $\|\mathbf{w}_{\text{lin}}\| < \|\mathbf{w}\|$. That is, the solution we have constructed is the minimum norm set of weights that minimizes E_{in}.

Problem 3.16 In Example 3.4, it is mentioned that the output of the final hypothesis $g(\mathbf{x})$ learned using logistic regression can be thresholded to get a 'hard' (± 1) classification. This problem shows how to use the risk matrix introduced in Example 1.1 to obtain such a threshold.

Consider fingerprint verification, as in Example 1.1. After learning from the data using logistic regression, you produce the final hypothesis

$$g(\mathbf{x}) = \mathbb{P}[y = +1 \mid \mathbf{x}],$$

which is your estimate of the probability that $y = +1$. Suppose that the cost matrix is given by

		True classification	
		$+1$ (correct person)	-1 (intruder)
you say	$+1$	0	c_a
	-1	c_r	0

For a new person with fingerprint $\mathbf{x}$, you compute $g(\mathbf{x})$ and you now need to decide whether to accept or reject the person (i.e., you need a hard classification). So, you will accept if $g(\mathbf{x}) \geq \kappa$, where κ is the threshold.

(a) Define the cost(accept) as your expected cost if you accept the person. Similarly define cost(reject). Show that

$$\begin{aligned} \text{cost(accept)} &= (1 - g(\mathbf{x}))c_a, \\ \text{cost(reject)} &= g(\mathbf{x})c_r. \end{aligned}$$

(b) Use part (a) to derive a condition on $g(\mathbf{x})$ for accepting the person and hence show that

$$\kappa = \frac{c_a}{c_a + c_r}.$$

(c) Use the cost-matrices for the Supermarket and CIA applications in Example 1.1 to compute the threshold κ for each of these two cases. Give some intuition for the thresholds you get.

Problem 3.17 Consider a function

$$E(u, v) = e^u + e^{2v} + e^{uv} + u^2 - 3uv + 4v^2 - 3u - 5v,$$

(a) Approximate $E(u + \Delta u, v + \Delta v)$ by $\hat{E}_1(\Delta u, \Delta v)$, where $\hat{E}_1$ is the first-order Taylor's expansion of E around $(u, v) = (0, 0)$. Suppose $\hat{E}_1(\Delta u, \Delta v) = a_u \Delta u + a_v \Delta v + a$. What are the values of a_u, a_v, and a?

(continued on next page)

(b) Minimize $\hat{E}_1$ over all possible $(\Delta u, \Delta v)$ such that $\|(\Delta u, \Delta v)\| = 0.5$. In this chapter, we proved that the optimal column vector $\begin{bmatrix}\Delta u \\ \Delta v\end{bmatrix}$ is parallel to the column vector $-\nabla E(u, v)$, which is called the *negative gradient direction*. Compute the optimal $(\Delta u, \Delta v)$ and the resulting $E(u + \Delta u, v + \Delta v)$.

(c) Approximate $E(u+\Delta u, v+\Delta v)$ by $\hat{E}_2(\Delta u, \Delta v)$, where $\hat{E}_2$ is the second-order Taylor's expansion of E around $(u, v) = (0, 0)$. Suppose

$$\hat{E}_2(\Delta u, \Delta v) = b_{uu}(\Delta u)^2 + b_{vv}(\Delta v)^2 + b_{uv}(\Delta u)(\Delta v) + b_u \Delta u + b_v \Delta v + b.$$

What are the values of b_{uu}, b_{vv}, b_{uv}, b_u, b_v, and b?

(d) Minimize $\hat{E}_2$ over all possible $(\Delta u, \Delta v)$ (regardless of length). Use the fact that $\nabla^2 E(u, v)\big|_{(0,0)}$ (the Hessian matrix at $(0, 0)$) is positive definite to prove that the optimal column vector

$$\begin{bmatrix}\Delta u^* \\ \Delta v^*\end{bmatrix} = -\left(\nabla^2 E(u, v)\right)^{-1} \nabla E(u, v),$$

which is called the *Newton direction*.

(e) Numerically compute the following values:

(i) the vector $(\Delta u, \Delta v)$ of length 0.5 along the Newton direction, and the resulting $E(u + \Delta u, v + \Delta v)$.

(ii) the vector $(\Delta u, \Delta v)$ of length 0.5 that minimizes $E(u+\Delta u, v+\Delta v)$, and the resulting $E(u + \Delta u, v + \Delta v)$. (*Hint: Let* $\Delta u = 0.5 \sin\theta$.)

Compare the values of $E(u+\Delta u, v+\Delta v)$ in (b), (e-i), and (e-ii). Briefly state your findings.

The negative gradient direction and the Newton direction are quite fundamental for designing optimization algorithms. It is important to understand these directions and put them in your toolbox for designing learning algorithms.

Problem 3.18 Take the feature transform Φ_2 in Equation (3.13) as Φ.

(a) Show that $d_{\text{VC}}(\mathcal{H}_\Phi) \leq 6$.

(b) Show that $d_{\text{VC}}(\mathcal{H}_\Phi) > 4$. *[Hint: Exercise 3.12]*

(c) Give an upper bound on $d_{\text{VC}}(\mathcal{H}_{\Phi_k})$ for $\mathcal{X} = \mathbb{R}^d$.

(d) Define

$$\tilde{\Phi}_2 : \mathbf{x} \to (1, x_1, x_2, x_1 + x_2, x_1 - x_2, x_1^2, x_1 x_2, x_2 x_1, x_2^2) \text{ for } \mathbf{x} \in \mathbb{R}^2.$$

Argue that $d_{\text{VC}}(\mathcal{H}_{\Phi_2}) = d_{\text{VC}}(\mathcal{H}_{\tilde{\Phi}_2})$. In other words, while $\tilde{\Phi}_2(\mathcal{X}) \in \mathbb{R}^9$, $d_{\text{VC}}(\mathcal{H}_{\tilde{\Phi}_2}) \leq 6 < 9$. Thus, the dimension of $\Phi(\mathcal{X})$ only gives an upper bound of $d_{\text{VC}}(\mathcal{H}_\Phi)$, and the exact value of $d_{\text{VC}}(\mathcal{H}_\Phi)$ can depend on the components of the transform.

Problem 3.19 A Transformer thinks the following procedures would work well in learning from two-dimensional data sets of any size. Please point out if there are any potential problems in the procedures:

(a) Use the feature transform

$$\Phi(\mathbf{x}) = \begin{cases} (\underbrace{0, \cdots, 0}_{n-1}, 1, 0, \cdots) & \text{if } \mathbf{x} = \mathbf{x}_n \\ (0, 0, \cdots, 0) & \text{otherwise .} \end{cases}$$

before running PLA.

(b) Use the feature transform Φ with

$$\phi_n(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_n\|^2}{2\gamma^2}\right)$$

using some very small γ.

(c) Use the feature transform Φ that consists of all

$$\phi_{i,j}(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - (i,j)\|^2}{2\gamma^2}\right),$$

before running PLA, with $i \in \{0, \frac{1}{100}, \ldots, 1\}$ and $j \in \{0, \frac{1}{100}, \ldots, 1\}$.

Chapter 4

Overfitting

Paraskavedekatriaphobia[1] (fear of Friday the 13th), and superstitions in general, are perhaps the most illustrious cases of the human ability to overfit. Unfortunate events are memorable, and given a few such memorable events, it is natural to *try* and find an explanation. In the future, will there be more unfortunate events on Friday the 13th's than on any other day?

Overfitting is the phenomenon where fitting the observed facts (data) well no longer indicates that we will get a decent out-of-sample error, and may actually lead to the opposite effect. You have probably seen cases of overfitting when the learning model is more complex than is necessary to represent the target function. The model uses its additional degrees of freedom to fit idiosyncrasies in the data (for example, noise), yielding a final hypothesis that is inferior. Overfitting can occur even when the hypothesis set contains only functions which are *far simpler* than the target function, and so the plot thickens ☺.

The ability to deal with overfitting is what separates professionals from amateurs in the field of learning from data. We will cover three themes: When does overfitting occur? What are the tools to combat overfitting? How can one estimate the degree of overfitting and 'certify' that a model is good, or better than another? Our emphasis will be on techniques that work well in practice.

4.1 When Does Overfitting Occur?

Overfitting literally means "Fitting the data more than is warranted." The main case of overfitting is when you pick the hypothesis with lower E_{in}, and it results in higher E_{out}. This means that E_{in} alone is no longer a good guide for learning. Let us start by identifying the cause of overfitting.

[1] from the Greek *paraskevi* (Friday), *dekatreis* (thirteen), *phobia* (fear)