Chapter 3

Regression[1]

In this Chapter we formally describe the regression problem, or the fitting of a representative line or curve (in higher dimensions a hyperplane or general surface) to a set of input/output data points as first broadly detailed in Section 1.2.1. Regression in general may be performed for a variety of reasons: to produce a so-called trend line (or curve) that can be used to help visually summarize, drive home a particular point about the data under study, or to learn a model so that precise predictions can be made regarding output values in the future. Here we also discuss more formally the notion of feature design for regression, in particular focusing on rare low dimensional instances (like the one outlined in Example 1.7) when very specific feature transformations of the data can be proposed. We finally end by discussing regression problems that have non-convex cost functions associated with them and a commonly used approach, called ℓ_2 regularization, for ameliorating some of the problems associated with the minimization of such functions.

3.1 The basics of linear regression

With linear regression we aim to fit a line (or hyperplane in higher dimensions) to a scattering of data. In this Section we describe the fundamental concepts underlying this procedure.

[1]This document is part of a book currently under development titled "Machine Learning Refined" (Cambridge University Press, late 2016) by Jeremy Watt, Reza Borhani, and Aggelos Katsaggelos. Please do not distribute. Feedback regarding any errors, comments on substance and style, recommendations, etc. is greatly appreciated! Contact: jermwatt@gmail.edu

3.1.1 Notation and modeling

Data for regression problems comes in the form of a training set of P input/output observation pairs

$$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_P, y_P)\}, \tag{3.1}$$

or $\{(\mathbf{x}_p, y_p)\}_{p=1}^{P}$ for short, where $\mathbf{x}_p$ and y_p denote the p^{th} input and output respectively. In many instances of regression, like the one discussed in Example 1.8, the input to regression problems is scalar-valued (the output will always be considered scalar-valued here) and hence the linear regression problem is geometrically speaking one of fitting a line to the associated scatter of data points in 2-dimensional space. In general however each input $\mathbf{x}_p$ may be a column vector of length N

$$\mathbf{x}_p = \begin{bmatrix} x_{1,p} \\ x_{2,p} \\ \vdots \\ x_{N,p} \end{bmatrix}. \tag{3.2}$$

in which case the linear regression problem is analogously one of fitting a hyperplane to a scatter of points in $N + 1$ dimensional space.

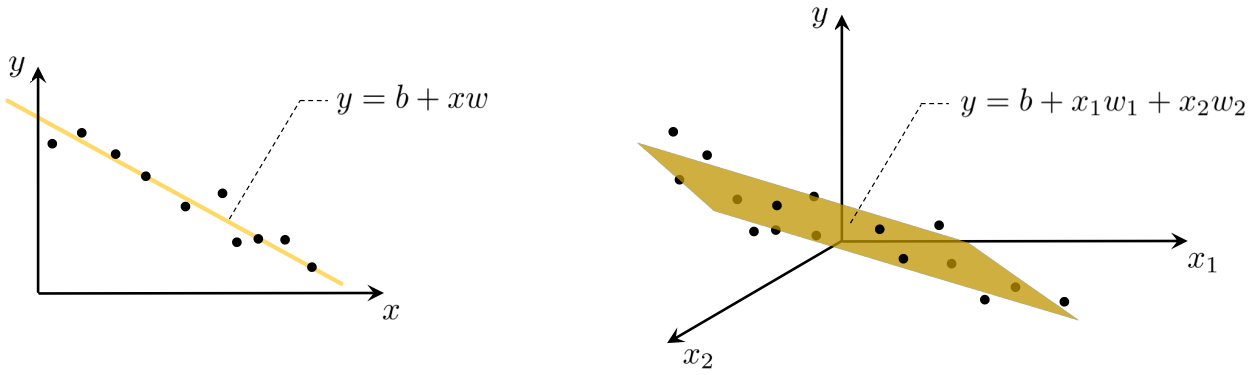

Figure 3.1. *(left panel) A dataset in two dimensions along with a well-fitting line. A line in two dimensions is defined as $b + xw = y$, where b is referred to as the bias and w the weight, and a point (x_p, y_p) lies close to it if $b + x_p w \approx y_p$. (right panel) A simulated three dimensional dataset along with a well-fitting hyperplane. A hyperplane is defined as $b + \mathbf{x}^T\mathbf{w} = y$, where again b is called the bias and $\mathbf{w}$ the weight vector, and a point $(\mathbf{x}_p, y_p)$ lies close to it if $b + \mathbf{x}_p^T\mathbf{w} \approx y_p$.*

In the case of scalar input the fitting a line to the data (see Figure 3.1) requires we determine a slope w and bias (or "y-intercept") b so that the approximate linear relationship holds between the input/output data

$$b + x_p w \approx y_p, \quad p = 1, ..., P. \tag{3.3}$$

Notice that we have used the *approximately equal* sign in (3.3) because we cannot be sure that all data lies completely on a single line. More generally when the input dimension is $N \geq 1$ then we have a bias and N associated weights

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} \tag{3.4}$$

to tune properly in order to fit a hyperplane (see Figure 3.1). Likewise the linear relationship in (3.3) is then more generally given as

$$b + \mathbf{x}_p^T \mathbf{w} \approx y_p, \quad p = 1, ..., P. \tag{3.5}$$

The elements of an input vector $\mathbf{x}_p$ are referred to as *input features* to a regression problem. For instance the student debt data described in Example 1.1 has only one feature: *year*. Conversely in the GDP growth rate data described in Example 3.1 the first element of the input feature vector might contain the feature *unemployment rate* (that is, the unemployment data from each country under study), the second might contain the feature *education level*, and so on.

Example 3.1. Predicting Gross Domestic Product growth rates

As an example of a regression problem with vector-valued input consider the problem of predicting the growth rate of a country's Gross Domestic Product (GDP), which is the value of all goods and services produced within a country during a single year. Economists are often interested in understanding factors (e.g., unemployment rate, education level, population count, land area, income level, investment rate, life expectancy, etc.,) which determine a country's GDP growth rate in order to inform better financial policy making. To understand how these various *features* of a country relate to its GDP growth rate economists often perform linear regression [22, 61].

In Figure 3.2 we show a heat map of the world where countries are color-coded based on their GDP growth rate in 2013, reported by the International Monetary Fund (IMF) (data from this Figure was taken from [1]).

3.1.2 The Least Squares cost function for linear regression

To find the parameters of the hyperplane which best fits a regression dataset, it is common practice to first form the *Least Squares cost function*. For a given set of parameters $(b, \mathbf{w})$

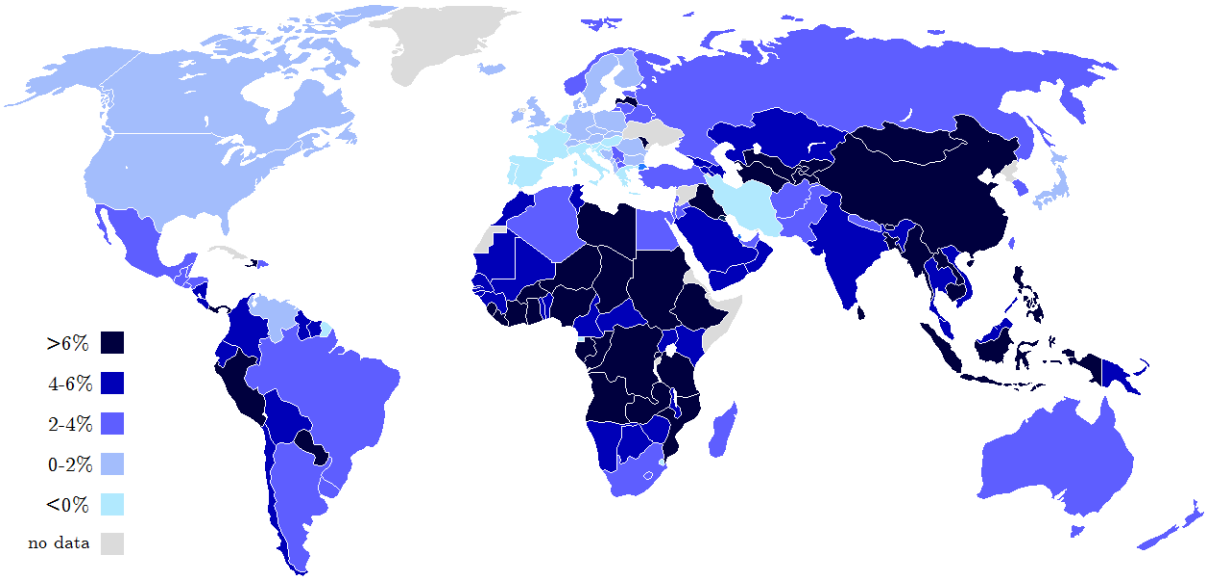

Figure 3.2. *A map of the world where countries are color-coded by their GDP growth rates (the darker color the higher the growth rate) as reported by the International Monetary Fund (IMF) in 2013.*

this cost function computes the total squared error between the associated hyperplane and the data (as illustrated pictorially in Figure 3.3), giving a good measure of how well the particular linear model fits the dataset. Naturally then the best fitting hyperplane is the one whose parameters minimize this error.

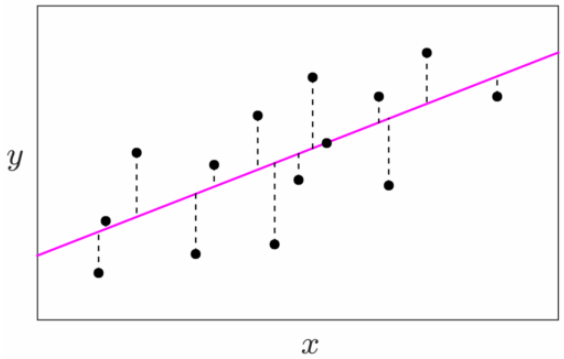

Figure 3.3. *A simulated two dimensional training dataset along with a line (in magenta) fit to the data using the Least Squares framework, which aims at recovering the line that minimizes the total squared length of the dashed error bars.*

Because we aim to have the system of equations in (3.5) hold as well as possible, to form

the desired cost we simply square the difference (or error) between the linear model $b + \mathbf{x}_p^T\mathbf{w}$ and the corresponding output y_p over the entire dataset. This gives the Least Squares cost function

$$g\left(b, \mathbf{w}\right) = \sum_{p=1}^{P} \left(b + \mathbf{x}_p^T\mathbf{w} - y_p\right)^2. \tag{3.6}$$

We of course want to find a parameter pair $(b, \mathbf{w})$ that provides a small value for $g\left(b, \mathbf{w}\right)$ since the larger this value the larger the squared error between the corresponding linear model and the data, and hence the poorer we represent the given data. Therefore we aim to *minimize* g over the bias and weight vector in order to recover the best pair $(b, \mathbf{w})$, which is written formally (see Section 2.2) as

$$\underset{b,\,\mathbf{w}}{\text{minimize}} \sum_{p=1}^{P} \left(b + \mathbf{x}_p^T\mathbf{w} - y_p\right)^2. \tag{3.7}$$

By checking the second order definition of convexity (see Exercise 3.3) we can easily see that the Least Squares cost function in (3.6) is convex. Figure 3.4 illustrates the Least Squares cost associated with the student loan data in Example 1.1, whose 'upward bending' shape confirms its convexity visually in the instance of that particular dataset.

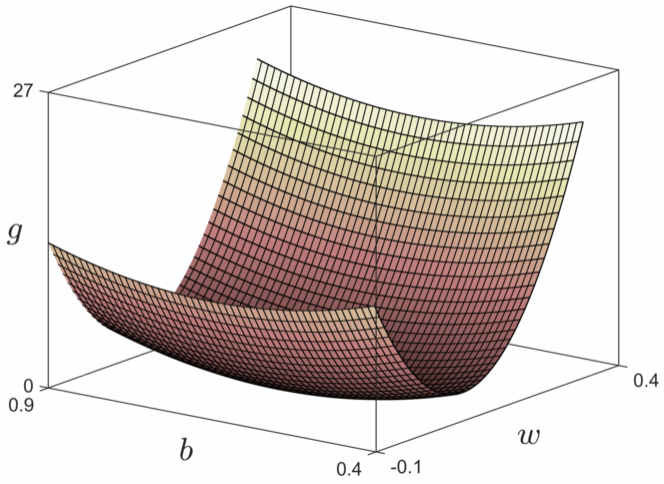

Figure 3.4. *The surface generated by the Least Squares cost function using the student loan debt data shown in Figure 1.8, is clearly convex. However regardless of the dataset, the Least Squares cost for linear regression is always convex.*

3.1.3 Minimization of the Least Squares cost function

Now that we have a minimization problem to solve we can employ the tools described in Chapter 2. To perform calculations it will first be convenient to use the following more

compact notation

$$\tilde{\mathbf{x}}_p = \begin{bmatrix} 1 \\ \mathbf{x}_p \end{bmatrix} \qquad \tilde{\mathbf{w}} = \begin{bmatrix} b \\ \mathbf{w} \end{bmatrix}. \tag{3.8}$$

With this notation we can rewrite the cost function shown in (3.6) in terms of the single vector $\tilde{\mathbf{w}}$ of parameters as

$$g\left(\tilde{\mathbf{w}}\right) = \sum_{p=1}^{P} \left(\tilde{\mathbf{x}}_p^T \tilde{\mathbf{w}} - y_p\right)^2. \tag{3.9}$$

To compute the gradient of this cost we simply apply the chain rule from calculus which gives

$$\nabla g\left(\tilde{\mathbf{w}}\right) = 2\sum_{p=1}^{P} \tilde{\mathbf{x}}_p \left(\tilde{\mathbf{x}}_p^T \tilde{\mathbf{w}} - y_p\right) = 2\left(\sum_{p=1}^{P} \tilde{\mathbf{x}}_p \tilde{\mathbf{x}}_p^T\right) \tilde{\mathbf{w}} - 2\sum_{p=1}^{P} \tilde{\mathbf{x}}_p y_p. \tag{3.10}$$

Using this we can perform gradient descent to minimize the cost. However in this (rare) instance we can actually solve the first order system directly in order to recover a global minimum. Setting the gradient above to zero and solving for $\tilde{\mathbf{w}}$ gives the system of linear equations

$$\left(\sum_{p=1}^{P} \tilde{\mathbf{x}}_p \tilde{\mathbf{x}}_p^T\right) \tilde{\mathbf{w}} = \sum_{p=1}^{P} \tilde{\mathbf{x}}_p y_p. \tag{3.11}$$

In particular one algebraic solution to this system[6], if the matrix $\sum_{p=1}^{P} \tilde{\mathbf{x}}_p \tilde{\mathbf{x}}_p^T$ is invertible[7], may be written as

$$\tilde{\mathbf{w}}^\star = \left(\sum_{p=1}^{P} \tilde{\mathbf{x}}_p \tilde{\mathbf{x}}_p^T\right)^{-1} \sum_{p=1}^{P} \tilde{\mathbf{x}}_p y_p. \tag{3.12}$$

However while an algebraically expressed solution is appealing it is typically more computationally efficient in practice to solve the original linear system using numerical linear algebra software.

---

[6] By setting the input vectors $\tilde{\mathbf{x}}_p$ columnwise to form the matrix $\tilde{\mathbf{X}}$ and by stacking the output y_p into the column vector $\mathbf{y}$ we may write the linear system in equation (3.11) equivalently as $\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T\tilde{\mathbf{w}} = \tilde{\mathbf{X}}\mathbf{y}$.

[7] In instances where the linear system in (3.11) has more than one solution, or in other words when $\sum_{p=1}^{P} \tilde{\mathbf{x}}_p \tilde{\mathbf{x}}_p^T$ is not invertible one can choose the solution with the smallest length (or ℓ_2 norm), sometimes written as $\tilde{\mathbf{w}}^\star = \left(\sum_{p=1}^{P} \tilde{\mathbf{x}}_p \tilde{\mathbf{x}}_p^T\right)^\dagger \sum_{p=1}^{P} \tilde{\mathbf{x}}_p y_p$, where $(\cdot)^\dagger$ denotes the pseudo-inverse of its input matrix. See Appendix C for further details.

3.1.4 The efficacy of a learned model

With optimal parameters $\tilde{\mathbf{w}}^\star = \begin{bmatrix} b^* \\ \mathbf{w}^* \end{bmatrix}$ we can compute the efficacy of the linear model in representing the training set by computing the mean squared error (or MSE)

$$MSE = \frac{1}{P}\sum_{p=1}^{P}\left(b^\star + \mathbf{x}_p^T\mathbf{w}^\star - y_p\right)^2. \tag{3.13}$$

When possible it is also a good idea to compute the MSE of a learned regression model on a set of new testing data, i.e., data that was not used to learn the model itself, to provide some assurance that the learned model will perform well on future datapoints. This is explored further in Chapter 5 in the context of *cross-validation*.

3.1.5 Predicting the value of new input data

With optimal parameters $(b^\star, \mathbf{w}^\star)$, found by minimizing the Least Squares cost, we can predict the output y_{new} of a new input feature $\mathbf{x}_{\text{new}}$ by simply plugging the new input into the tuned linear model and estimating the associated output as

$$y_{\text{new}} = b^\star + \mathbf{x}_{\text{new}}^T\mathbf{w}^\star. \tag{3.14}$$

This is illustrated pictorially on a toy dataset for the case when $N = 1$ in Figure 3.5.

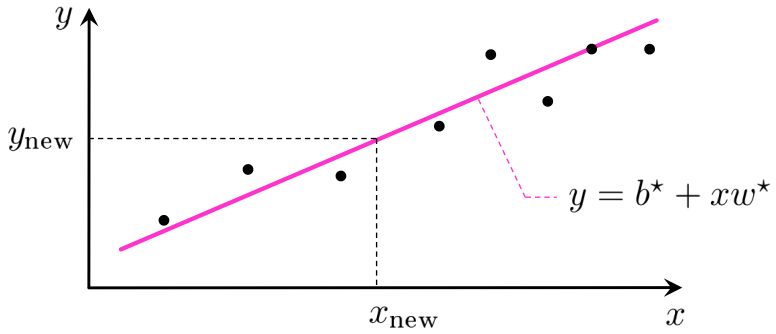

Figure 3.5. *Once a line/hyperplane has been fit to a dataset via minimizing the Least Squares cost function it may be used to predict the output value of future input. Here a line has been fit to a two dimensional dataset in this manner, giving optimal parameters $b^\star$ and $w^\star$, and the output value of a new point x_{new} is made using the learned linear model as $y_{new} = b^\star + x_{new}w^\star$.*

3.2 Knowledge-driven feature design for regression

In many regression problems the relationship between input feature(s) and output values is nonlinear as in Figure 3.6 which illustrates a simulated dataset where the scalar feature x and the output y are related in a nonlinear fashion. In such instances a linear model would clearly fail at representing how the input and output are related. In this brief Section we present two simple examples through which we discuss how to fit a nonlinear model to the data when we have significant understanding or *knowledge* about the data itself. This knowledge may originate from our prior understanding or intuition about the phenomenon under study or simply our ability to visualize low dimensional data. As we now see, based on this knowledge we can propose an appropriate nonlinear feature transformation which allows us to employ the linear regression framework as described in the previous Section.

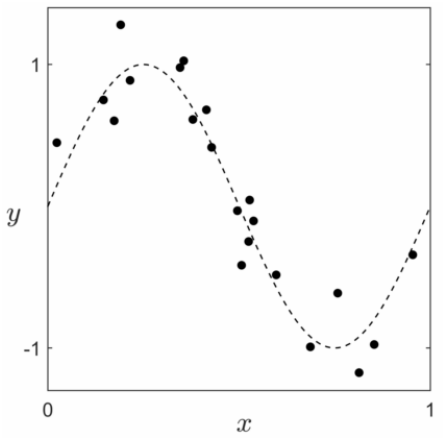

Figure 3.6. *A simulated regression dataset where the relationship between the input feature x and the output y is not linear. However because we can visualize this dataset we can see that there is clearly a structured nonlinear relationship between its input and output. Our knowledge in this instance, based on our ability to visualize the data, allows us to design a new feature for the data and formulate a corresponding function (shown here in dashed black) that appears to be generating the data.*

Example 3.2. Sinusoidal pattern

In the left panel of Figure 3.7 we show a simulated regression dataset (first shown in Figure 3.6) consisting of $P = 21$ data points. Visually analyzing this data it appears to trace out (with some noise) one period of a sine wave over the interval $[0, 1]$. Therefore we can reasonably propose that some weighted version of the sinusoidal function $f(x) = \sin(2\pi x)$, i.e., $y = b + f(x)\, w$ where b and w are respectively a bias and weight to learn, will properly

describe this data. In machine learning the function $f(x)$, in this instance a sinusoid, is referred to as a *feature transformation* of the original input. One could of course propose a more curvy feature, but the sinusoid seems to explain the data fairly well while remaining relatively simple.

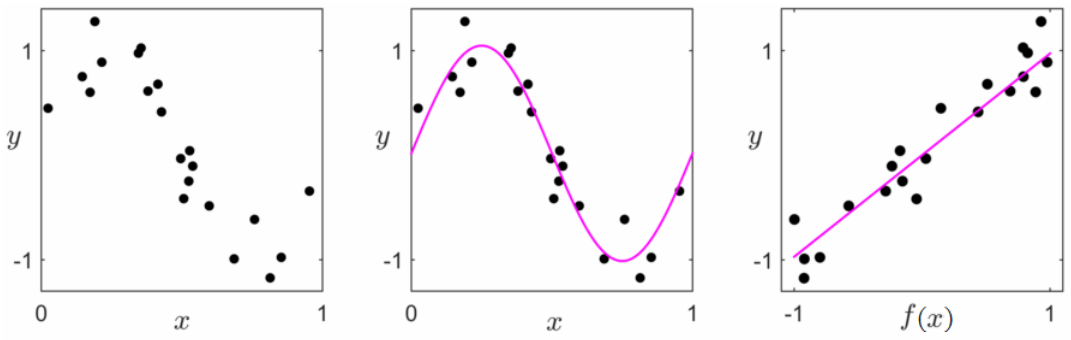

Figure 3.7. *(left panel) A simulated regression dataset. (middle panel) A weighted form of a simple sinusoid* $y = b + f(x)\,w$ *(in magenta), where* $f(x) = sin(2\pi x)$ *and where* b *and* w *are tuned properly, describes the data quite well. (right panel) Fitting a sinusoid in the original feature space is equivalent to fitting a line in the transformed feature space where the input feature has undergone feature transformation* $x \longrightarrow f(x) = sin(2\pi x)$.

By fitting a simple weighted sinusoid to the data, we would like to find the parameter pair (b, w) so that

$$b + f(x_p)\,w = b + \sin(2\pi x_p)\,w \approx y_p, \quad p = 1, ..., P. \tag{3.15}$$

Note that while this is nonlinear in the input x, it is still linear in both b and w. In other words, the relationship between the output y and the new feature $f(x) = \sin(2\pi x)$ is linear. Plotting $\{(f(x_p), y_p)\}_{p=1}^{P}$ in the *transformed feature space* (i.e., the space whose input is given by the new feature $f(x)$ and whose output is still y) shown in the right panel of Figure 3.7, we can see that the new feature and given output are now indeed linearly related.

After creating the new features for the data by transforming the input as $f_p = f(x_p) = \sin(2\pi x_p)$, we may solve for the parameter pair by minimizing the Least Squares cost function formed by summing the squared error between the model containing each transformed input $b + f_p w$ and the corresponding output value y_p (so that (3.15) holds as well as possible) as

$$\underset{b,\,w}{\text{minimize}} \sum_{p=1}^{P} (b + f_p w - y_p)^2. \tag{3.16}$$

The cost function here is still convex, and can be minimized by a numerical scheme like gradient descent or by directly solving its first order system to recover a global minimum.

By using the compact notation

$$\tilde{\mathbf{f}}_p = \begin{bmatrix} 1 \\ f_p \end{bmatrix} \qquad \tilde{\mathbf{w}} = \begin{bmatrix} b \\ w \end{bmatrix} \tag{3.17}$$

we can rewrite the cost function in terms of the single vector $\tilde{\mathbf{w}}$ of parameters as

$$g\left(\tilde{\mathbf{w}}\right) = \sum_{p=1}^{P} \left(\tilde{\mathbf{f}}_p^T \tilde{\mathbf{w}} - y_p\right)^2. \tag{3.18}$$

Mirroring the discussion in Section 3.1.3, setting the gradient of the above to zero then gives the linear system of equations to solve

$$\left(\sum_{p=1}^{P} \tilde{\mathbf{f}}_p \tilde{\mathbf{f}}_p^T\right) \tilde{\mathbf{w}} = \sum_{p=1}^{P} \tilde{\mathbf{f}}_p y_p. \tag{3.19}$$

In the middle and right panels of Figure 3.7 we show the resulting fit to the data $y = b^\star + f\left(x\right) w^\star$ in magenta, where $b^\star$ and $w^\star$ are recovered by solving this system. We refer to this fit as the *estimated data generating function* since it is our estimation of the underlying continuous function generating this dataset (shown in dashed black in Figure 3.6). Note that this fit is a sinusoid in the original feature space (middle panel), and a line in the transformed feature space (right panel).

Example 3.3. Galileo and uniform acceleration

Recall Galileo's acceleration experiment, first described in Example 1.7. In the left panel of Figure 3.8 we show data consisting of $P = 6$ data points from a modern reenactment of this experiment [64], where the input x denotes the time and the output y denotes the corresponding portion of the ramp traversed. Several centuries ago Galileo saw data very similar looking to the data shown here. To describe the data he saw Galileo proposed the relation $y = f\left(x\right) w$, where $f\left(x\right) = x^2$ is a simple quadratic feature and w is some weight to be tuned to the data. Note that in this specific example there is no need to add a bias parameter b to the quadratic model since at time zero the ball has not moved at all, and hence the output must be precisely zero.

Looking at how the data is distributed in the left panel of Figure 3.8, we too can intuit that such a quadratic feature of the input appears to be a reasonable guess at what lies beneath the data shown.

By fitting a simple weighted quadratic to the data, we would like to find parameter w such that

$$f\left(x_p\right) w = x_p^2 w \approx y_p, \quad p = 1, ..., P. \tag{3.20}$$

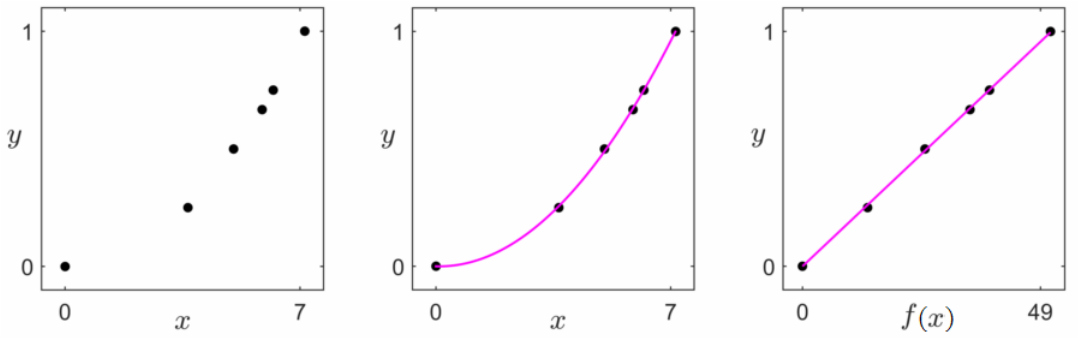

Figure 3.8. *Data from a modern reenactment of Galileo's ramp experiment. (left panel) The raw data seems to reflect a quadratic relationship between the input and output variables. (middle panel) A weighted form of a simple quadratic feature $y = f(x)\,w$ (in magenta) where $f(x) = x^2$ and where w is tuned properly, describes the data quite well. (right panel) Fitting a quadratic to the data in the original feature space is equivalent to fitting a line to the data in a transformed feature space wherein the input feature has undergone feature transformation $x \longrightarrow f(x) = x^2$.*

Although the relationship between the input feature x and output y is nonlinear, this model is still linear in the weight w. Thus we may tune w by minimizing the Least Squares cost function after forming the new features. That is, transforming each input as $f_p = f(x_p) = x_p^2$ we can find the optimal w by solving

$$\underset{w}{\text{minimize}} \sum_{p=1}^{P} \left(f_p w - y_p\right)^2. \tag{3.21}$$

The cost function in (3.21) is again convex and we may solve for the optimal w by simply checking the first order condition. Setting the derivative of the cost function equal to zero and solving for w, after a small amount of algebraic rearrangement, gives

$$w^\star = \frac{\sum_{p=1}^{P} f_p y_p}{\sum_{p=1}^{P} f_p^2}. \tag{3.22}$$

We show in the middle panel of Figure 3.8 the weighted quadratic fit $y = f(x)\,w^\star$ (our estimated data generating function) to the data (in magenta) in the original feature space. In the right panel of this Figure we show the same fit, this time in the transformed feature space where the fit is linear.

3.2.1 General conclusions

The two Examples discussed above are very special. In each case by using our ability to visualize the data we have been able to design an excellent new feature $f(x)$ explicitly using common algebraic functions. As we have seen in these two examples a properly designed feature (or set of features more generally) for linear regression is one that provides a good *nonlinear* fit in the original space while, simultaneously, a good *linear* fit in the transformed feature space. In other words, a properly designed set of features for linear regression produces a good linear fit to the feature-transformed data[13].

A properly designed feature (or set of features) for linear regression provides a good *nonlinear* fit in the original feature space and, simultaneously, a good *linear* fit in the transformed feature space.

However just because we can visualize a low dimensional regression dataset does not mean we can easily design a proper feature 'by eye' as we have done in Examples 3.2 and 3.3. For instance, in Figure 3.9 we show a simulated dataset built by randomly taking $P = 30$ inputs x_p on the interval $[0, 1]$, evaluating each through a rather wild function[15] $y(x)$ (shown in dashed black in the Figure), and then adding a small amount of noise to each output. Here even though we can clearly see a structured nonlinear relationship in the data, it is not immediately obvious how to formulate a proper feature $f(x)$ to recover the original data generating function $y(x)$. No common algebraic function (e.g., a quadratic, a sine wave, an exponential, etc.,) seems to be a reasonable candidate and hence our knowledge, in this case the fact that we can visualize the data itself, is not enough to form a proper feature (or set of features) for this data.

For vector-valued input we can say something very similar. While we can imagine forming a proper set of features for a dataset with vector-valued input, the fact that we cannot visualize the data prohibits us from 'seeing' the right sort of feature(s) to use.

In fact rarely in practice can we use our knowledge of a dataset to construct perfect features. Often we may only be able to make a rough guess at a proper feature transformation given our intuition about the data at hand, or can make no educated guess at all. Thankfully, we can *learn* feature transformations automatically from the data itself that can ameliorate this problem. This process will be described later in Chapter 5.

[13] Technically speaking there is one subtle yet important caveat to the use of the word 'good' in this statement, being that we do not want to 'overfit' the data (an issue we discuss at length in Chapter 5). However for now this issue will not concern us.

[15] Here $y(x) = e^{3x}\frac{\sin\left(3\pi^2(x-0.5)\right)}{3\pi^2(x-0.5)}$.

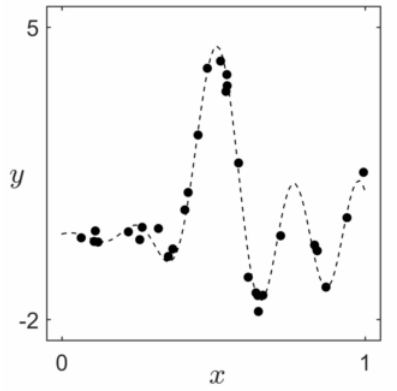

Figure 3.9. *A simulated dataset generated as noisy samples of a data generating function $y(x)$. We show $y(x)$ here in dashed black. Unlike the previous two cases in Figures 3.7 and 3.8, it is not so clear what sort of function would serve as a proper feature $f(x)$ here.*

3.3 Nonlinear regression and ℓ_2 regularization

In the previous Section we discussed examples of regression where the nonlinear relationship in a given dataset could be determined by intuiting a nonlinear feature transformation, and where (once the data is transformed) the associated cost functions remained convex and linear in their parameters. Because the input/output relationship associated to each of these examples was linear in its parameters (see (3.16) and (3.21)), each is still referred to as a *linear* regression problem. In this Section we explore the consequences of employing nonlinear models for regression where the corresponding cost function is non-convex and the input/output relationship is nonlinear in its parameters (referred to as instances of *nonlinear* regression). We also introduce a common tool for partially ameliorating the practical inconveniences of non-convex cost functions, referred to as the ℓ_2 regularizer. Specifically we describe how the ℓ_2 regularizer helps numerical optimization techniques avoid poor stationary points of non-convex cost functions. Because of this utility regularization is often used with non-convex cost functions, as we will see later with e.g., neural networks in Chapters 5 - 7 [16].

While the themes of this Section are broadly applicable for the purpose of clarity we frame our discussion of nonlinear regression and ℓ_2 regularization around a single classic example referred to as *logistic regression* (which we will also see arise in the context of classification in the next Chapter). Further examples of nonlinear regression are explored in the Chapter exercises.

[16] Additionally, ℓ_2 regularization also arises in the context of the (convex) support vector machine classifier, as we will see in Section 4.3. Another popular use of ℓ_2 regularization is discussed later in Section 7.3 in the context of 'cross-validation'.

3.3.1 Logistic regression

At the heart of the classic logistic regression problem is the so-called *logistic sigmoid function*, illustrated in the left panel of Figure 3.10, and defined mathematically as

$$\sigma(t) = \frac{1}{1 + e^{-t}}, \tag{3.23}$$

where t can take on any real value. Invented in the early 19$^{\text{th}}$ century by mathematician Pierre François Verhulst [68], this function was designed in his pursuit of modeling how a population (of microbes, animal species, etc.) grows over time, taking into account the realistic assumption that regardless of the kind of organism under study the system in which it lives has only a finite amount of resources[19]. Thus, as a result, there should be a strict cap on the total population in any biological system. According to Verhulst's model, the initial stages of growth should follow an exponential trend until a saturation level where, due to lack of required resources (e.g., space, food, etc.,), the growth stabilizes and levels off[20].

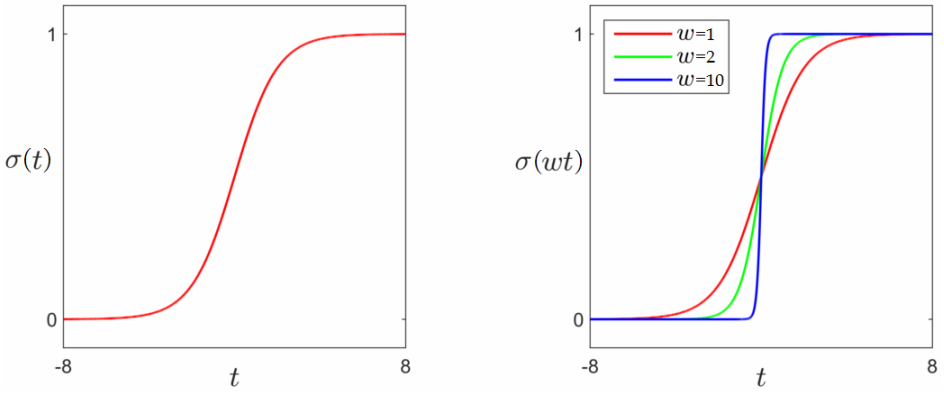

Figure 3.10. *(left panel) Plot of the logistic sigmoid function defined in (3.23). Note that the output of this function is always between 0 and 1. (right panel) By increasing the weight w of the sigmoid function $\sigma(wt)$ from $w = 1$ (red) to $w = 2$ (green) and finally to $w = 10$ (blue), the sigmoid becomes an increasingly good approximator of a 'step function', that is a function that only takes on the values 0 and 1 with a sharp transition between the two.*

[19] Beyond its classical use in modeling population growth, we will see in the next Chapter that logistic regression can also be used for the task of classification.

[20] Like any good mathematician Verhulst first phrased this ideal population growth model in terms of a differential equation. Denoting the desired function f and the maximum population level of the system as 1, he supposed that the population growth rate $\frac{df}{dt}$ should, at any time t, be proportional to both the current population level f as well as the remaining capacity left in the system $1 - f$. Together this gives the differential equation $\frac{df}{dt} = f(1 - f)$. One can check by substitution that the logistic sigmoid function satisfies this relationship with initial condition $f(0) = 1/2$.

If a dataset of P points $\{(x_p, y_p)\}_{p=1}^{P}$ is roughly distributed like a sigmoid function then this data satisfies

$$\sigma\left(b + x_p w\right) \approx y_p, \quad p = 1, ..., P, \tag{3.24}$$

where b and w are parameters which must be properly tuned. The weight w, as illustrated in the right panel of Figure 3.10, controls how quickly the system saturates, and the bias term b shifts the curve left and right along the horizontal axis. Likewise when the input is N-dimensional the system of equations given in (3.24) may be written analogously as

$$\sigma\left(b + \mathbf{x}_p^T \mathbf{w}\right) \approx y_p, \quad p = 1, ..., P, \tag{3.25}$$

where as usual $\mathbf{x}_p = \begin{bmatrix} x_{1,p} & x_{2,p} & \dots & x_{N,p} \end{bmatrix}^T$ and $\mathbf{w} = \begin{bmatrix} w_1 & w_2 & \dots & w_N \end{bmatrix}^T$. Notice that unlike the analogous set of equations with linear regression given in e.g., equation (3.5) each of these equations is nonlinear[22] in b and $\mathbf{w}$. These nonlinearities lead to a non-convex Least Squares cost function which is formed by summing the squared differences of equation (3.25) over all p

$$g\left(b, \mathbf{w}\right) = \sum_{p=1}^{P} \left(\sigma\left(b + \mathbf{x}_p^T \mathbf{w}\right) - y_p\right)^2. \tag{3.26}$$

Using the compact notation $\tilde{\mathbf{x}}_p = \begin{bmatrix} 1 \\ \mathbf{x}_p \end{bmatrix}$ and $\tilde{\mathbf{w}} = \begin{bmatrix} b \\ \mathbf{w} \end{bmatrix}$, the fact that the derivative of the sigmoid is given as $\sigma'\left(t\right) = \sigma\left(t\right)\left(1 - \sigma\left(t\right)\right)$ (see footnote 20), and the chain rule from calculus the gradient of this cost can be calculated as

$$\nabla g\left(\tilde{\mathbf{w}}\right) = 2\sum_{p=1}^{P} \left(\sigma\left(\tilde{\mathbf{x}}_p^T \tilde{\mathbf{w}}\right) - y_p\right) \sigma\left(\tilde{\mathbf{x}}_p^T \tilde{\mathbf{w}}\right) \left(1 - \sigma\left(\tilde{\mathbf{x}}_p^T \tilde{\mathbf{w}}\right)\right) \tilde{\mathbf{x}}_p. \tag{3.27}$$

Due to the many nonlinearities involved in the above system of equations solving the first order system directly is a fruitless venture, instead a numerical technique (i.e., gradient descent or Newton's method) must be used to find a useful minimum of the associated cost function.

Example 3.4. Bacterial growth

In the left panel of Figure 3.11 we show a real dataset consisting of $P = 9$ data points corresponding to the normalized cell concentration[23] of a particular bacteria, *Lactobacillus delbrueckii*, in spatially constrained laboratory conditions over the period of 24 hours. Also

[22] In certain circumstances this system may be transformed into one that is linear in its parameters. See exercise 3.10 for further details.

[23] Cell concentration is measured as the mass of organism per unit volume. Here we have normalized the cell concentration values so that they lie strictly in the interval $(0, 1)$.

shown in this panel are two sigmoidal fits (shown in magenta and green) found via minimizing the Least Squares cost in (3.26) using gradient descent. In the middle panel we show the surface of the cost function which is clearly non-convex, having stationary points in the large flat region colored orange as well as global minimum in the long narrow valley highlighted in dark blue. Two paths taken by initializing gradient descent at different values are shown in magenta and green, respectively, on the surface itself. While the initialization of the magenta path in the yellow-green area of the surface leads to the global minimum, which corresponds with the good sigmoidal fit in magenta shown in the left panel, the initialization of the green path in the large flat orange region leads to a poor solution, with corresponding poor fit shown in green in the left panel. In the right panel we show the contour plot of the same surface (along with the two gradient descent paths) that more clearly shows the long narrow valley containing the desired global minimum of the surface.

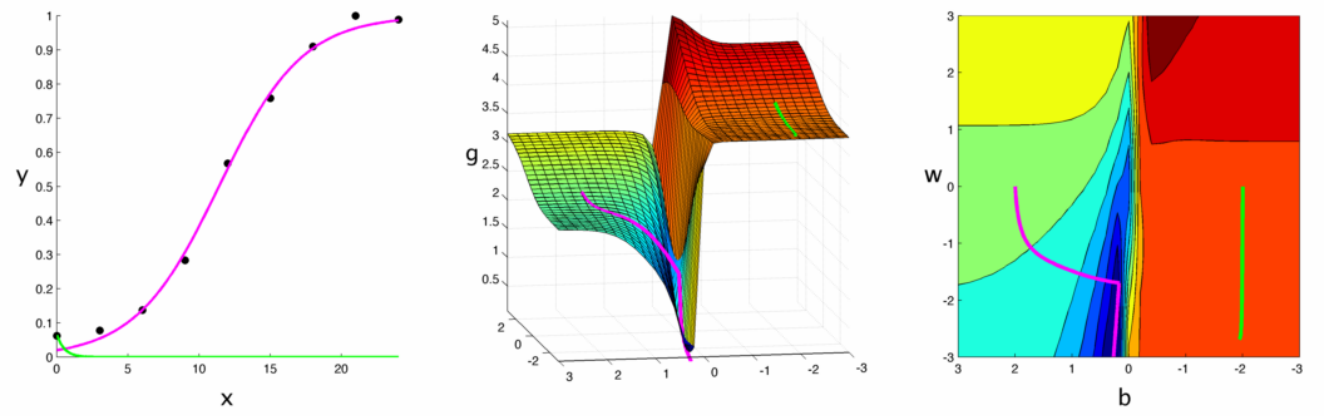

Figure 3.11. *(left panel) A dataset along with two sigmoidal fits (shown in magenta and green) each found via minimizing the Least Squares cost in (3.26) using gradient descent with a different initialization. A surface (middle) and contour (right) plot of this cost function along with the paths taken by the two runs of gradient descent. Each path has been colored to match the resulting sigmoidal fit produced in the left panel (see text for further details). Data in this Figure is taken from* [37].

3.3.2 Non-convex cost functions and ℓ_2 regularization

The problematic flat areas posed by non-convex cost functions like the one shown in Figure 3.11 can be ameliorated by the addition of a *regularizer*. A regularizer is a simple convex function that is often added to such a cost function, slightly convexifying it and thereby helping numerical optimization techniques avoid poor solutions in its flat areas. One of the most common regularizers used in practice is the squared ℓ_2 norm of the weights $\|\mathbf{w}\|_2^2 =$

$\sum_{n=1}^{N} w_n^2$, referred to as the ℓ_2 regularizer. To regularize a cost function $g(b, \mathbf{w})$ with this regularizer we simply add it to g giving the regularized cost function

$$g(b, \mathbf{w}) + \lambda \|\mathbf{w}\|_2^2 . \tag{3.28}$$

Here $\lambda \geq 0$ is a parameter (set by the user in practice) that controls the strength of each term, the original cost function and the regularizer, in the final sum. For example, if $\lambda = 0$ we have our original cost. On the other hand, if λ is set very large then the regularizer drowns out the cost and we have $g(b, \mathbf{w}) + \lambda \|\mathbf{w}\|_2^2 \approx \lambda \|\mathbf{w}\|_2^2$. Typically in practice λ is set fairly small (e.g., $\lambda = 0.1$ or smaller).

In Figure 3.12 we show two simple examples of non-convex cost functions which exemplify how the ℓ_2 regularizer can help numerical techniques avoid many (but not all) poor solutions in practice.

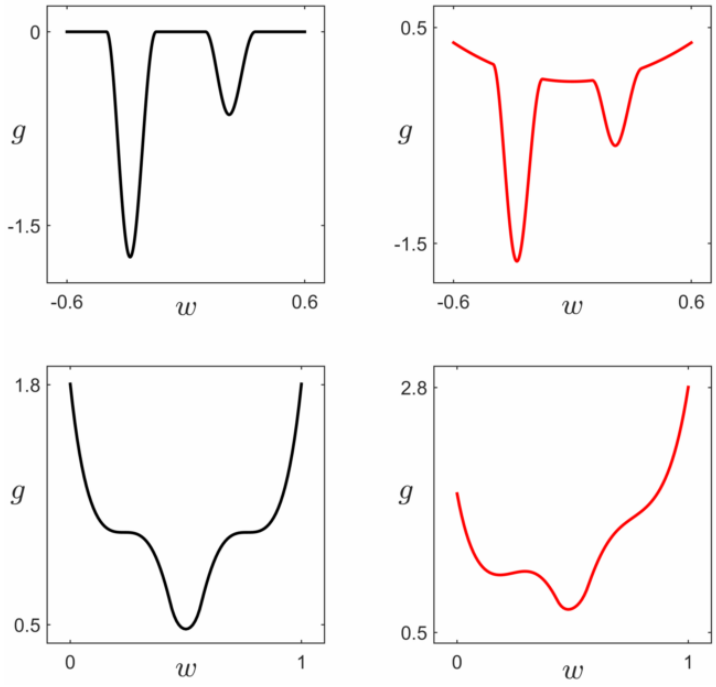

Figure 3.12. *Two examples of non-convex functions with flat regions (top left panel) and saddle points (bottom left panel) where numerical optimization methods can halt undesirably. Using the ℓ_2 regularizer we can slightly convexify each (right panels) which can help avoid some of these undesirable solutions. See text for further details.*

The first non-convex cost function[26], shown in the top left panel of Figure 3.12, is defined over

[26]Here the cost is defined as $g(w) = \max^2(0, e^{-w}\sin(4\pi(w - 0.1)))$, and λ has been set fairly high at $\lambda = 1$ for illustrative purposes only.

a symmetric interval about the origin and has three large flat areas containing undesirable stationary points. This kind of non-convex function is highly problematic because if an algorithm like gradient descent or Newton's method is initialized at any point lying in these flat regions it will immediately halt. In the top right panel we show an ℓ_2 regularized version of the same cost. Notice how regularization slightly convexifies the entire cost function, and in particular how it forces the flat regions to curve upwards. Now if e.g., gradient descent is initialized in either of the two flat regions on the left or right sides it will in fact travel downwards and reach a minimum. Note that both minima have slightly changed position from the original cost, but as long as λ is set relatively small this small change does not typically make a difference in practice. Notice in this instance, however, that regularization has not helped with the problem of gradient descent halting at a poor solution if initialized in the middle flat region of the original cost. That is, by regularizing we have actually created a local minimum near the middle of the original flat region in the regularized cost function, and so if gradient descent is initialized in this region it will halt at this undesirable solution.

The second non-convex cost function[28], shown in the bottom left panel of Figure 3.12, is defined over the unit interval and has two saddle points at which the derivative is zero and so at which e.g., gradient descent will halt undesirably if initialized at a point corresponding to any region on the far left or right. In the lower right panel we show the ℓ_2 regularized cost which no longer has an issue with the saddle point on the right, as the region surrounding it has been curved upwards. However the saddle point on the left is still problematic, as regularizing the original cost has created a local minimum near the point that will cause gradient descent to continue to halt at an undesirable solution.

Example 3.5. ℓ_2 regularized Least Squares for logistic regression

We saw in Figure 3.11 that the initialization of gradient descent in the flat orange region resulted in a poor fit to the bacterial growth data. A second version of all three panels from this Figure is duplicated in Figure 3.13, only here we add the ℓ_2 regularizer with $\lambda = 0.1$ to the original Least Squares logistic cost in (3.26). Formally, this ℓ_2 regularized Least Squares cost function is written as

$$g\left(b, \mathbf{w}\right) = \sum_{p=1}^{P} \left(\sigma\left(b + \mathbf{x}_p^T \mathbf{w}\right) - y_p\right)^2 + \lambda \left\|\mathbf{w}\right\|_2^2. \tag{3.29}$$

Once again in order to minimize this cost we can employ gradient descent (see Exercise 3.13). Comparing the regularized surface in Figure 3.13 to the original shown in Figure 3.11 we can see that regularizing the original cost curves the flat regions of the surface upwards,

[28]Here the cost is defined as $g\left(w\right) = \max^2\left(0,\, \left(3w - 2.3\right)^3 + 1\right) + \max^2\left(0,\, \left(-3w + 0.7\right)^3 + 1\right)$, and λ has been set fairly high at $\lambda = 1$ for illustrative purposes only.

helping gradient descent avoid poor solutions when initialized in these areas. Now both initializations first shown in Figure 3.11 lead gradient descent to the global minimum of the surface.

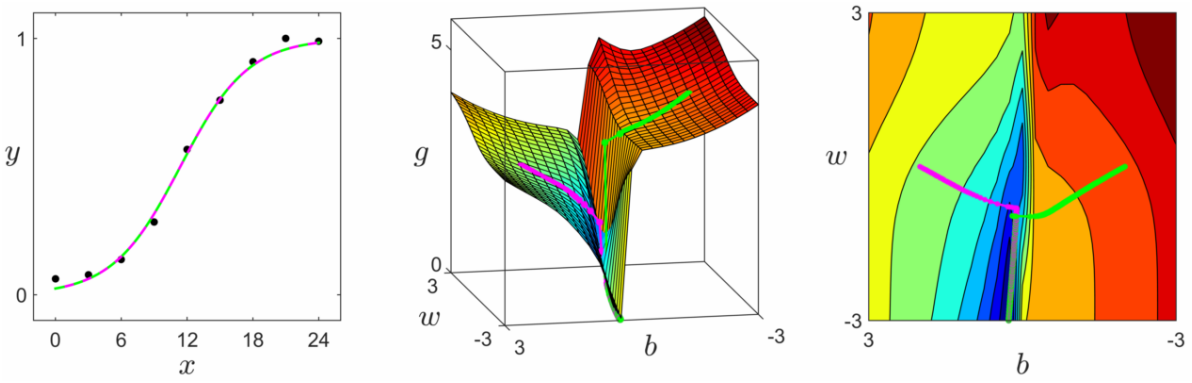

Figure 3.13. *A regularized version of Figure 3.11. (left panel) Plot of the bacterial growth dataset along with two overlapping sigmoidal fits (shown in magenta and green) found via minimizing the ℓ_2 regularized Least Squares cost for logistic regression in (3.29) using gradient descent. (middle and right panels) The surface and contour plot of the regularized cost function along with the paths (in magenta and green) of gradient descent with same two initializations as shown in Figure 3.11. While the surface is still non-convex, the large flat region that originally led the initialization of the green path to a poor solution with the unregularized cost has been curved upwards by the regularizer, allowing the green run of gradient descent to reach the global minimum of the problem. Data in this Figure is taken from* [37].

3.4 Summary

Linear regression is a fundamental predictive learning problem which aims at determining the relationship between continuous-valued input and output data via the fitting of an appropriate model that is linear in its parameters. In this Chapter we first saw how to fit a linear model (i.e., a line or hyperplane in higher dimensions) to a given dataset, culminating in the minimization of the Least Squares cost function at the end of Section 3.1. Due to the parameters being linearly related, this cost function may be minimized by solving the associated first order system.

We then saw in Section 3.2 how in some rare instances our understanding of a phenomenon, typically due to our ability to visualize a low dimensional dataset, can permit us to accurately suggest an appropriate feature transformation to describe our data. This provides a proper nonlinear fit to the original data while simultaneously fitting linearly to the data in an associated transformed feature space.

Finally using the classical example of logistic regression, we saw in Section 3.3 how a nonlinear regression model typically involves the need to minimize an associated non-convex cost function. We then saw how ℓ_2 regularization is used as a way of 'convexifying' a non-convex cost function to help gradient descent avoid some undesirable stationary points of such a function.

3.5 Chapter exercises

Section 3.1 exercises

Exercise 3.1. Fitting a regression line to the student debt data

Fit a linear model to the U.S. student load debt dataset shown in Figure 1.8, called *student_ debt_ data.csv*, by solving the associated linear regression Least Squares problem. If this linear trend continues what will be the total student debt be in 2050?

Exercise 3.2. Kleiber's law and linear regression

After collecting and plotting a considerable amount of data comparing the body mass versus metabolic rate (a measure of at rest energy expenditure) of a variety of animals, early 20^{th} century biologist Max Kleiber noted an interesting relationship between the two values. Denoting by x_p and y_p the body mass (in kg) and metabolic rate (in kJ/day) of a given animal respectively, treating the body mass as the input feature Kleiber noted (by visual inspection) that the natural log of these two values were linearly related. That is,

$$w_0 + \log(x_p)\, w_1 \approx \log(y_p)\,. \tag{3.30}$$

In Figure 3.14 we show a large collection of transformed data points$\left\{\left(\log(x_p), \log(y_p)\right)\right\}_{p=1}^{P}$, each representing an animal ranging from a small black-chinned hummingbird in the bottom left corner to a large walrus in the top right corner.

a) Fit a linear model to the data shown in Figure 3.14 (called *kleibers_ law_ data.csv*). Make sure to take the log of both arguments!

b) Use the optimal parameters you found in part (a) along with the properties of the log function to write the nonlinear relationship between the body mass x and the metabolic rate y.

c) Use your fitted line to determine how many calories an animal weighing 10 kg requires (note each calorie is equivalent to 4.18 J).

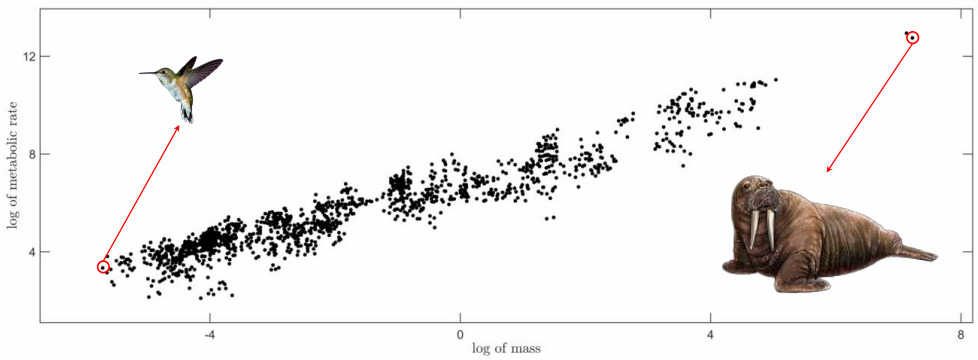

Figure 3.14. *A large set of body mass/metabolic rate data points, transformed by taking the log of each value, for various animals over wide range of different masses.*

Exercise 3.3. The Least Squares cost for linear regression is convex

Show that the Least Squares cost function for linear regression written compactly as in Section 3.1.3

$$g\left(\tilde{\mathbf{w}}\right) = \sum_{p=1}^{P}\left(\tilde{\mathbf{x}}_p^T\tilde{\mathbf{w}} - y_p\right)^2 \tag{3.31}$$

is a convex quadratic function by completing the following steps.

a) Show that $g\left(\tilde{\mathbf{w}}\right)$ can be written as a quadratic function of the form

$$g\left(\tilde{\mathbf{w}}\right) = \frac{1}{2}\tilde{\mathbf{w}}^T\mathbf{Q}\tilde{\mathbf{w}} + \mathbf{r}^T\tilde{\mathbf{w}} + d \tag{3.32}$$

by determining proper $\mathbf{Q}$, $\mathbf{r}$, and d.

b) Show that $\mathbf{Q}$ has all nonnegative eigenvalues (hint: see exercise 2.10).

c) Verify that $\nabla^2 g\left(\tilde{\mathbf{w}}\right) = \mathbf{Q}$ and so that g satisfies the second order definition of convexity, and is therefore convex.

d) Show that applying a single Newton step (see Section 2.2.4) to minimize the Least Squares cost function leads to precisely the first order system of linear equations discussed in Section 3.1.3, i.e., to the system $\left(\sum_{p=1}^{P}\tilde{\mathbf{x}}_p\tilde{\mathbf{x}}_p^T\right)\tilde{\mathbf{w}} = \sum_{p=1}^{P}\tilde{\mathbf{x}}_p y_p$.. (This is because the Least Squares cost is a quadratic function, as in Example 2.7).

Section 3.2 exercises

Exercise 3.4. Reproduce Galileo's example

Use the value for the optimal weight shown in equation (3.22) to reproduce the fits shown in Figure 3.8. The data shown in this Figure is located in the file *Galileo_ data.csv*.

Exercise 3.5. Reproduce the sinusoidal example

a) Setup the first order system associated with the Least Squares cost function being minimized in equation (3.16).

b) Reproduce the sinusoidal and associated linear fit shown in the middle and right panels of Figure 3.7 by solving for the proper weights via the first order system you determined in part **a)**. The dataset shown in this Figure is called *sinusoid_ example_ data.csv*.

Exercise 3.6. Galileo's extended ramp experiment

In this Exercise we modify Galileo's ramp experiment, discussed in Example 3.3, to explore the relationship between the angle x of the ramp and the distance y that the ball travels during a certain fixed amount of time. In Figure 3.15 we plot 6 simulated measurements corresponding to 6 different angle values x (measured in degrees).

a) Propose a suitable nonlinear feature transformation for this dataset such that the relationship between new feature you form and the distance traveled is linear in its weights. *Hint: there is no need for a bias parameter b here.*

b) Formulate and minimize a Least Squares cost function using your new feature for a proper weight w, the data (the data shown is located in the file *another_ ramp_ experiment.csv*). Plot the resulting fit in the data space.

Exercise 3.7. Moore's law and the power of future computers

Gordon Moore, co-founder of Intel corporation, predicted in a 1965 paper[33] that the number of transistors on an integrated circuit would double approximately every two years. This conjecture, referred to nowadays as the Moore's law, has proven to be sufficiently accurate over the past 5 decades. Since the processing power of computers is directly related to the

[33] One can find a modern reprinting of this paper in e.g., [46].

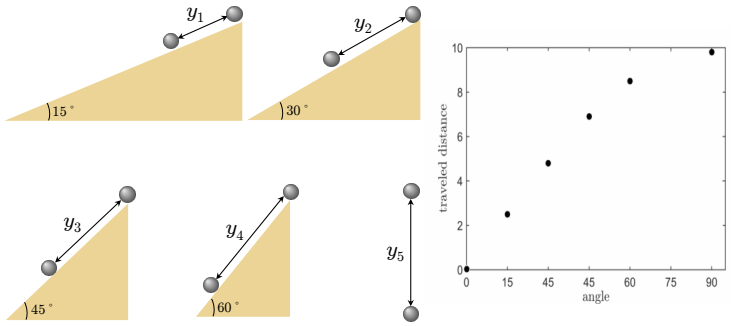

Figure 3.15. *An extended set of data from Galileo's ramp experiment, first described in Example 3.3. See text for details.*

number of transistors in their CPUs, Moore's law provides a trend model to predict the computing power of future microprocessors. Figure 3.16 plots the transistor counts of several microprocessors versus the year they were released, starting from Intel 4004 in 1971 with only 2300 transistors, to Intel's Xeon E7 introduced in 2014 with more than 4.3 billion transistors.

a) Propose an exponential-based transformation of the Moore's law dataset shown in Figure 3.16 so that the transformed input/output data is related linearly. *Hint: to produce a linear relationship you will end up having to transform the output, not the input.*

b) Formulate and minimize a Least Squares cost function for appropriate weights, and fit your model to the data in the original data space. The data shown here is located in the file *transistor_counts.csv.*

Exercise 3.8. Ohm's law and linear regression

Ohm's law, proposed by the German physicist Georg Simon Ohm following a series of experiments made by him in the 1820s, connects the magnitude of the current in a galvanic circuit to the sum of all the exciting forces in the circuit, as well as the length of the circuit. Although he did not publish any account of his experimental results, it is easy to verify his law using a simple experimental setup, shown in the left panel of Figure 3.17, that is very similar to what he then utilized (the data in this Figure is taken from [45]). The spirit lamp heats up the circuit, generating an electromotive force which creates a current in the coil deflecting the needle of the compass. The tangent of the deflection angle is directly proportional to the magnitude of the current passing through the circuit. The magnitude of this current, denoted by I, varies depending on the length of the wire used to close the

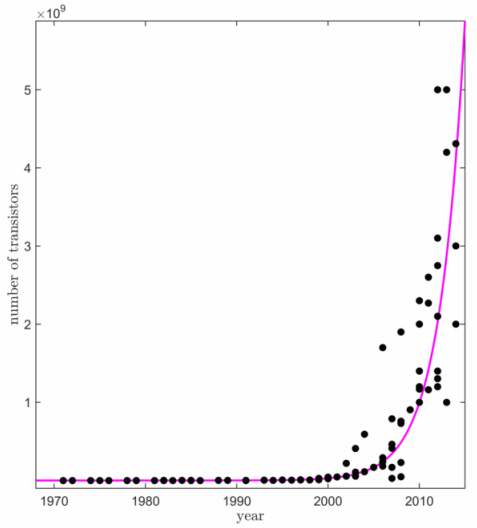

Figure 3.16. *As Moore proposed fifty years ago, the number of transistors in microprocessors versus the year they were invented follows an exponential pattern.*

circuit (dashed curve). In the right panel of Figure 3.17 we plot the readings of the current I (in terms of the tangent of the deflection angle) when the circuit is closed with a wire of length x (in cm), for 5 different values of x.

a) Suggest a suitable nonlinear transformation of the original data to fit (located in the file *ohms_ data.csv*) so that the transformed input/output data is related linearly. *Hint: to produce a linear relationship you will likely end up having to transform the output.*

b) Formulate a proper Least Squares cost function using your transformed data and minimize it to recover ideal parameters for your model.

c) Fit your proposed model to the data and display it in the original data space.

Exercise 3.9. Determining the orbit of celestial bodies

One of the first recorded uses of regression via the Least Squares approach was made by Carl Frederich Gauss, a German mathematician, physicist, and all around polymath, who was interested in calculating the orbit of the asteroid Pallas by leveraging a dataset of recorded observations. Although Gauss solved the problem using ascension and declination data observed from the earth [50], here we modify the problem so that the simulated data shown in the left panel of Figure 3.18 simulates Cartesian coordinates of the location of

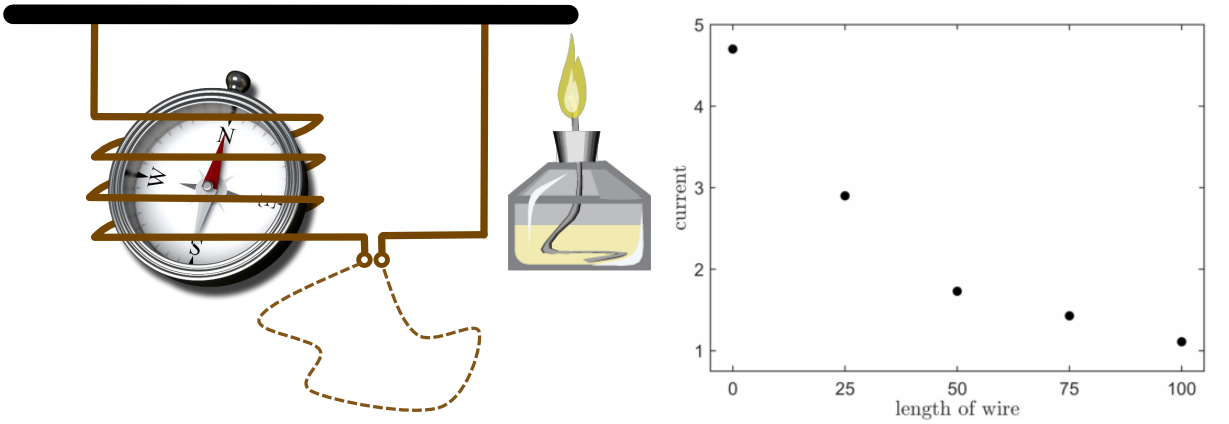

Figure 3.17. *(left panel) Experimental setup for verification of Ohm's law. Black and brown wires are made up of constantan and copper, respectively. (right panel) Current measurements for 5 different lengths of closing wire.*

the asteroid on its orbital plane. With this assumption, and according to Kepler's laws of planetary motion, we need to fit an ellipse to a series of observation points in order to recover the true orbit.

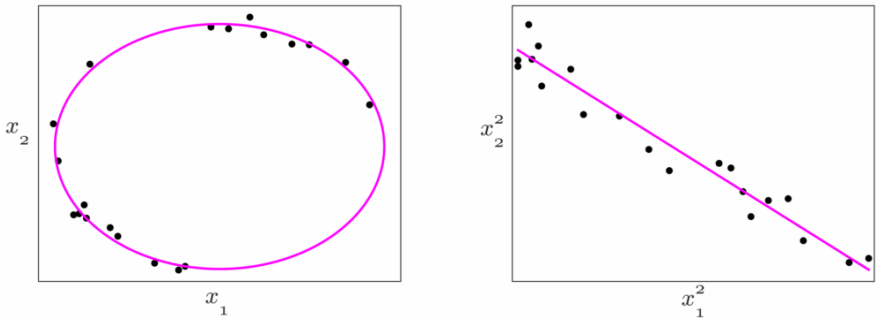

Figure 3.18. *Simulated observation data for the location of the asteroid Pallas on its orbital plane. The ellipsoidal curve fit to the data approximates the true orbit of Pallas. (right panel) Fitting an ellipsoid to the data in the original data space is equivalent to fitting a line to the data in a new space where both dimensions are squared.*

In this instance the data comes in the form of $P = 20$ noisy coordinates $\{(x_{1,p}, x_{2,p})\}_{p=1}^{P}$ taken from an ellipsoid with the standard form of

$$\left(\frac{x_{1,p}}{\nu_1}\right)^2 + \left(\frac{x_{2,p}}{\nu_2}\right)^2 \approx 1 \quad \text{for all } p = 1...P, \tag{3.33}$$

where ν_1 and ν_2 are tunable parameters. By making the substitutions $w_1 = \left(\frac{1}{\nu_1}\right)^2$ and

$w_2 = \left(\frac{1}{\nu_2}\right)^2$ this can be phrased equivalently as a set of approximate linear equations

$$x_{1,p}^2 w_1 + x_{2,p}^2 w_2 \approx 1 \quad \text{for all } p = 1...P. \tag{3.34}$$

a) Reformulate the equations shown above using vector notation as

$$\mathbf{f}_p^T \mathbf{w} \approx y_p \quad \text{for all } p = 1...P \tag{3.35}$$

by determining the appropriate $\mathbf{f}_p$ and y_p where $\mathbf{w} = \begin{bmatrix} w_1 & w_2 \end{bmatrix}^T$.

b) Formulate and solve the associated Least Squares cost function to recover the proper weights $\mathbf{w}$ and plot the ellipse with the data shown in the left panel of Figure 3.18 located in the file *asteroid_ data.csv.*

Section 3.3 exercises

Exercise 3.10. Logistic regression as a linear system

In this exercise you will explore particular circumstances that allow one to transform the nonlinear system of equations in (3.24) into a system which is linear in the parameters b and w. In order to do this recall that a function f has an *inverse* at t if another function f^{-1} exists such that $f^{-1}(f(t)) = t$. For example, the exponential function $f(t) = e^t$ has the inverse $f^{-1}(t) = \log(t)$ for every t since we always have $f^{-1}(f(t)) = \log(e^t) = t$.

a) Show that the logistic sigmoid has an inverse for each t where $0 < t < 1$ of the form $\sigma^{-1}(t) = \log\left(\frac{t}{1-t}\right)$ and check that indeed $\sigma^{-1}(\sigma(t)) = t$ for all such values of t.

b) Suppose that for a given dataset $\{(x_p, y_p)\}_{p=1}^P$ that $0 < y_p < 1$ for all p. Apply the sigmoid inverse to the system shown in equation (3.24) to derive the equivalent set of linear equations

$$b + x_p w \approx \log\left(\frac{y_p}{1 - y_p}\right) \quad p = 1, ..., P. \tag{3.36}$$

Since the equations in (3.36) are now linear in both b and w we may solve for these parameters by simply checking the first order condition for optimality.

c) Using the dataset *bacteria_ data.csv* solve the Least Squares cost function based on the linear system of equations from part **b)** and plot the data, along with the logistic sigmoid fit to the data as shown in Figure 3.19

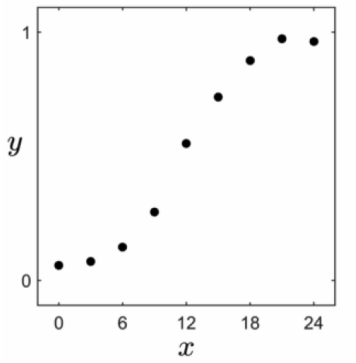

Figure 3.19. *The normalized cell concentration of Lactobacillus delbrueckii in a constrained laboratory environment over the period of 24 hours. Data in this Figure is taken from* [37].

Exercise 3.11. Code up gradient descent for logistic regression

In this exercise you will reproduce the gradient descent paths shown in Figure 3.11.

a) Verify that the gradient descent step shown in equation (3.27) is correct.
Note that the this gradient can be written more compactly by denoting $\sigma_p^{k-1} = \sigma\left(\tilde{\mathbf{x}}_p^T \tilde{\mathbf{w}}^{k-1}\right)$, $r_p^{k-1} = 2\left(\sigma_p^{k-1} - y_p\right)\sigma_p^{k-1}\left(1 - \sigma_p^{k-1}\right)$ for all $p = 1, ..., P$, and $\mathbf{r}^{k-1} = \begin{bmatrix} r_1^{k-1} & r_2^{k-1} & \cdots & r_P^{k-1} \end{bmatrix}$, and stacking the columns vectors $\widetilde{\mathbf{x}}_p$ column-wise into the matrix $\widetilde{\mathbf{X}}$. Then the gradient can be written as $\nabla g\left(\tilde{\mathbf{w}}^{k-1}\right) = \widetilde{\mathbf{X}}\mathbf{r}^{k-1}$. For programming languages like Python and MATLAB/OCTAVE that have especially efficient implementations of matrix/vector operations this can be much more efficient than explicitly summing over the P points as in equation (3.27).

b) The surface in this Figure was generated via the wrapper *nonconvex_logistic_growth* with the dataset *bacteria_data.csv*, and inside the wrapper you must complete a short gradient descent function to produce the descent paths called

$$[\text{in, out}] = \text{grad\_descent}\left(\widetilde{\mathbf{X}}, \mathbf{y}, \tilde{\mathbf{w}}^0\right) \tag{3.37}$$

where 'in' and 'out' contain the gradient steps $\tilde{\mathbf{w}}^k = \tilde{\mathbf{w}}^{k-1} - \alpha_k \nabla g\left(\tilde{\mathbf{w}}^{k-1}\right)$ taken and corresponding objective value $g\left(\tilde{\mathbf{w}}^k\right)$ respectively, $\widetilde{\mathbf{X}}$ is the input data matrix, $\mathbf{y}$ the output values, and $\tilde{\mathbf{w}}^0$ the initial point.
Almost all of this function has already been constructed for you. For example, the step length is fixed at $\alpha_k = 10^{-2}$ for all iterations, etc, and you must only enter the gradient of the associated cost function. Pressing 'run' in the editor will run gradient descent and will reproduce Figure 3.11.

Exercise 3.12. A general sinusoid model nonlinear in its parameters

Recall the periodic sinusoidal regression discussed in Example 3.2. There we chose a model $b + \sin(2\pi x_p w) \approx y_p$ that fit the given data which was linear in the weights b and w, and we saw that the corresponding Least Squares cost function was therefore convex. This allowed us to solve for the optimal values for these weights in closed form via the first order system, with complete assurance that they represent a global minimum of the associated Least Squares cost function. In this exercise you will investigate how a simple change to this model leads to a comparably much more challenging optimization problem to solve.

Figure 3.20 shows a set of $P = 75$ data points $\{(x_p, y_p)\}_{p=1}^{P}$ generated via the model

$$w_1 \sin(2\pi x_p w_2) + \epsilon = y_p \quad \text{for all } p = 1...P \tag{3.38}$$

where $\epsilon > 0$ is a small amount of noise. This dataset may be located in the file *extended_sinusoid_data.csv*. Unlike the previous instance here the model is nonlinear in both the weights. Here w_1 controls the amplitude (i.e., stretches the model in the vertical direction) and w_2 controls the frequency (i.e., how quickly the sinusoid completes a single period) of the sinusoidal model.

We can then attempt to recover optimal weights of a representative curve for this data set by minimizing the associated Least Squares cost function with respect to the dataset

$$g(\mathbf{w}) = \sum_{p=1}^{P} \left(w_1 \sin(2\pi x_p w_2) - y_p\right)^2. \tag{3.39}$$

a) Plot the surface of g over the region defined by $-3 \leq w_1, w_2 \leq 3$.

b) Discuss the approach you would take to find the best possible stationary point of this function, along with any potential difficulties you foresee in doing so.

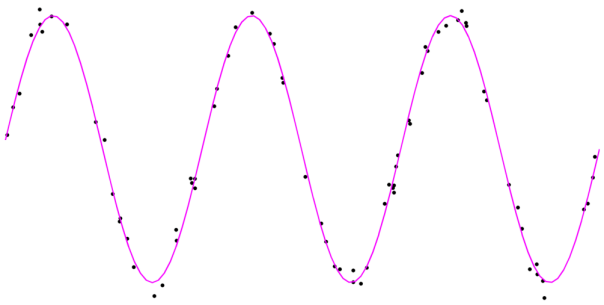

Figure 3.20. *A set of $N = 75$ periodic data points along with the underlying sinusoidal model used to generate the data in magenta.*

Exercise 3.13. Code up gradient descent for ℓ_2 regularized logistic regression

In this exercise you will reproduce Figure 3.13 by coding up gradient descent to minimize the regularized logistic regression Least Squares cost function shown in equation (3.29).

a) Verify that the gradient of the cost function can be written as

$$\nabla g\left(\tilde{\mathbf{w}}\right) = 2\sum_{p=1}^{P}\left(\sigma\left(\tilde{\mathbf{x}}_p^T\tilde{\mathbf{w}}\right) - y_p\right)\sigma\left(\tilde{\mathbf{x}}_p^T\tilde{\mathbf{w}}\right)\left(1 - \sigma\left(\tilde{\mathbf{x}}_p^T\tilde{\mathbf{w}}\right)\right)\tilde{\mathbf{x}}_p + 2\lambda\begin{bmatrix} 0 \\ \mathbf{w} \end{bmatrix}. \tag{3.40}$$

b) The surface in this Figure was generated via the wrapper *l2reg_ nonconvex_ logistic_ growth* with the dataset *bacteria_ data.csv*, and inside the wrapper you must complete a short gradient descent function to produce the descent paths called

$$[\text{in, out}] = \text{grad\_descent}\left(\widetilde{\mathbf{X}}, \mathbf{y}, \tilde{\mathbf{w}}^0\right) \tag{3.41}$$

where 'in' and 'out' contain the gradient steps $\tilde{\mathbf{w}}^k = \tilde{\mathbf{w}}^{k-1} - \alpha_k \nabla g\left(\tilde{\mathbf{w}}^{k-1}\right)$ taken and corresponding objective value $g\left(\tilde{\mathbf{w}}^k\right)$ respectively, $\widetilde{\mathbf{X}}$ is the input data matrix whose p^{th} column is the input data $\tilde{\mathbf{x}}_p$, $\mathbf{y}$ the output values stacked into a column vector, and $\tilde{\mathbf{w}}^0$ the initial point.

Almost all of this function has already been constructed for you. For example, the step length is fixed at $\alpha_k = 10^{-2}$ for all iterations, etc., and you must only enter the gradient of the associated cost function. Pressing 'run' in the editor will run gradient descent and will reproduce Figure 3.13.

Exercise 3.14. The ℓ_2 regularized Newton's method

Recall from Section 2.2.4 that when applied to minimizing non-convex cost functions Newton's method can climb to local maxima (or even diverge) due to the concave shape of the quadratic second order Taylor series approximation at concave points of the cost function (see Figure 2.11). One very common way of dealing with this issue, which we explore formally in this exercise, is to add an ℓ_2 regularizer (centered at the each step) to the quadratic approximation used by Newton's method in order to ensure that it is convex at each step. This is also commonly done when applying Newton's method to convex functions as well since, as we will see, the addition of a regularizer increases the eigenvalues of the Hessian and therefore helps avoid numerical problems associated with solving linear systems with zero (or near-zero) eigenvalues.

a) At the k^{th} iteration of the regularized Newton's method we add an ℓ_2 regularizer centered at $\mathbf{w}^{k-1}$, i.e., $\frac{\lambda}{2}\left\|\mathbf{w}-\mathbf{w}^{k-1}\right\|_2^2$ where $\lambda \geq 0$, to the second order Taylor series approximation in (2.18) giving

$$h\left(\mathbf{w}\right) = g\left(\mathbf{w}^{k-1}\right) + \nabla g\left(\mathbf{w}^{k-1}\right)^T\left(\mathbf{w}-\mathbf{w}^{k-1}\right) + \frac{1}{2}\left(\mathbf{w}-\mathbf{w}^{k-1}\right)^T\nabla^2 g\left(\mathbf{w}^{k-1}\right)\left(\mathbf{w}-\mathbf{w}^{k-1}\right) + \frac{\lambda}{2}\left\|\mathbf{w}-\mathbf{w}^{k-1}\right\|_2^2. \tag{3.42}$$

Show that the first order condition for optimality leads to the following adjusted Newton's system for a stationary point of the above quadratic

$$\left[\nabla^2 g\left(\mathbf{w}^{k-1}\right) + \lambda \mathbf{I}_{N\times N}\right]\mathbf{w} = \left[\nabla^2 g\left(\mathbf{w}^{k-1}\right) + \lambda \mathbf{I}_{N\times N}\right]\mathbf{w}^{k-1} - \nabla g\left(\mathbf{w}^{k-1}\right). \tag{3.43}$$

b) Show that the eigenvalues of $\nabla^2 g\left(\mathbf{w}^{k-1}\right) + \lambda \mathbf{I}_{N\times N}$ in the system above can all be made to be positive by setting λ large enough. What is the smallest value of λ that will make this happen? This is typically the value used in practice. *Hint: see exercise* 2.9.

c) Using the value of λ determined in part **b)** conclude that the ℓ_2 regularized second order Taylor series approximation centered at $\mathbf{w}^{k-1}$ in (3.42) is convex. *Hint: see exercise 2.11.*

For a non-convex function λ, as defined in part **b)** above, is typically adjusted at each step so that it just forces the eigenvalues of $\nabla^2 g\left(\mathbf{w}^{k-1}\right) + \lambda \mathbf{I}_{N\times N}$ to be all positive. In the case of a convex cost since the eigenvalues of $\nabla^2 g\left(\mathbf{w}^{k-1}\right)$ are always nonnegative (via the second order definition of convexity) *any* positive value of λ will force the eigenvalues of $\nabla^2 g\left(\mathbf{w}^{k-1}\right) + \lambda \mathbf{I}_{N\times N}$ to be all positive. Therefore often for convex functions λ is set fixed for all iterations at some small value like $\lambda = 10^{-3}$ or $\lambda = 10^{-4}$.