Machine Learning
the Fundamentals

William J. Deuschle

Harvard College
Cambridge, Massachusetts
April, 2019

Contents

Chapter 2

Linear Regression

A major component of machine learning, the one that most people associate with ML, is dedicated to making predictions under uncertain conditions. Given some input data, we would like to produce a target output. In this chapter, we're going to focus on the case where our prediction is a continuous, real number. This type of problem is known as **regression**.

2.1 Introduction and Motivation

It's important to emphasize that linear regression is just one example of the larger set of techniques known as regression.

Definition 2.1.1 (Regression): A class of problems that seeks to make predictions about unknown target variables given observed input variables.

2.1.1 Examples of Regression

We can imagine many situations where regression is useful:

1. Predicting a person's height given the height of their parents.

2. Predicting the amount of time someone will take to pay back a loan given their credit history.

3. Predicting what time a package will arrive given current weather and traffic conditions.

2.1.2 Linear Regression

In this chapter, we're specifically going to focus on **linear regression**, which means that our goal is to find some linear combination of the $x_1, ..., x_D$ input values that predict our target y.

Definition 2.1.2 (Linear Regression): Suppose we have an input $\mathbf{x} \in \mathbb{R}^D$ and a continuous target $y \in \mathbb{R}$. Linear regression determines weights $w_i \in \mathbb{R}$ that combine the values of x_i to produce y:

$$y = w_0 + w_1 x_1 + ... + w_D x_D \tag{2.1}$$

⋆ Notice w_0 in the expression above, which doesn't have a corresponding x_0 value. This is known as the *bias* term. If you consider the definition of a line $y = mx + b$, the bias term is comparable to the intercept b. It can account for a general trend in our data, such as if all of our target y values are greater than 50.

All of these follow a similar formula: a data input $\mathbf{x}$ gets transformed into prediction y. For example, consider 10 year old Sam. She is curious about how tall she will be when she grows up. She has a data set of parents' heights and the final heights of their children. The inputs $\mathbf{x}$ are:

$$x_1 = \text{height of mother (cm)}$$
$$x_2 = \text{height of father (cm)}$$

Using linear regression, she determines the weights $\mathbf{w}$ to be:

$$\mathbf{w} = [34, 0.39, 0.33]$$

Sam's mother is 165 cm tall and her father is 185 cm tall. Using the results of the linear regression solution, Sam solves for her expected height:

$$\text{Sam's height} = 34 + 0.39(165) + 0.33(185) = \textbf{159.4 cm}$$

ML Framework Cube: Linear Regression

Let's inspect the categories linear regression falls into for our ML framework cube. First, as we've already stated, linear regression deals with a **continuous** input and output domain. Second, our goal is to make predictions on future data points, and to construct something capable of making those predictions we first need a labeled data set of inputs and outputs. This makes linear regression a **supervised** technique. Third and finally, linear regression is a special case when it comes to being **probabilistic or non-probabilistic**. Depending on our interpretation, it can be either one! We will explain how this works later in the chapter.

Domain	***Training***	***Probabilistic***
Continuous	Supervised	Yes / No

2.2 Technical

The most basic form of linear regression is a simple weighted combination of the input variables $\mathbf{x}$, which you will often see written as:

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + ... + w_D x_D \tag{2.2}$$

2.2.1 Merging of Bias

We're going to introduce a common notational trick here for making the bias term, w_0, easier to handle. At the moment w_0 is unwieldly because it is not being multiplied by an x_i value. Without the bias term, we could easily express $y(\mathbf{x}, \mathbf{w})$ as:

$$y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x} = w_0 x_0 + w_1 x_1 + ... + w_D x_D \tag{2.3}$$

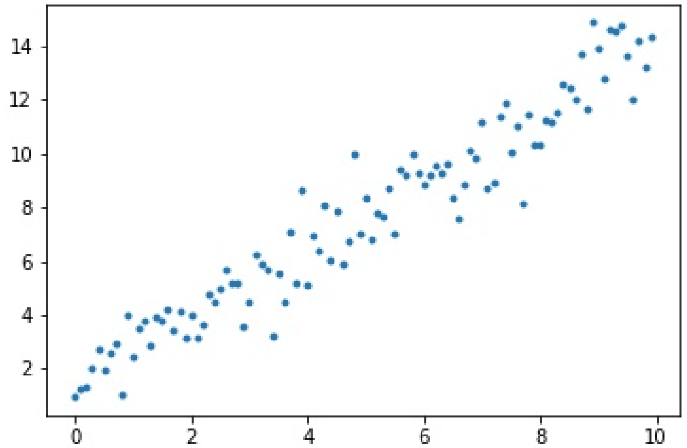

Figure 2.1: Data set with clear trend.

This is more compact, easier to reason about, and makes properties of linear algebra nicer for the calculations we will be performing. An easy way to keep our bias term and still get this nice form of $\mathbf{w}^T\mathbf{x}$ is to simply introduce another variable, x_0, that is always 1 for every data point. For example, considering the case of Sam's height from above, we have the height of her parents, $\mathbf{x}$:

$$\mathbf{x} = (165, 185)$$

We now add a 1 in the first position of the data point to make it:

$$\mathbf{x'} = (1, 165, 185)$$

We do this for every point in our data set. This bias trick lets us write:

$$y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T\mathbf{x} = w_0x_0 + w_1x_1 + ... + w_Dx_D \tag{2.4}$$

as desired.

2.2.2 Visualization of Linear Regression

Let's try to build some intuition about how linear regression works. Our algorithm is given a collection of data points: inputs $\mathbf{x}$ and corresponding targets $\mathbf{y}$. Our goal is to find the best set of weights $\mathbf{w}$ such that given a new data point $\mathbf{x}$, we can accurately predict the true target value y. This is visualizable in the simple case where $\mathbf{x}$ is a 1-dimensional input variable, as in Figure 2.1.

Our eyes are naturally able to detect a very clear trend in this data. If we were given a new $\mathbf{x}^*$ data point, how would be predict its target value y? We would first fit a line to our data, as in Figure 2.2, and then find where on that line the new $\mathbf{x}^*$ value sits.

That is the entirety of linear regression! It fits the 'best' line to our data, and then uses that line to make predictions. In 1-D input space, this manifests itself as the simple problem seen above, where we need only find a single bias term w_0 (which acts as the intercept of the line) and single weight w_1 (which acts as the slope of the line). However, the same principle applies to higher dimensional data as well. We're always fitting the line (meaning calculate the weights $\mathbf{w}$) that give us maximal predictive power.

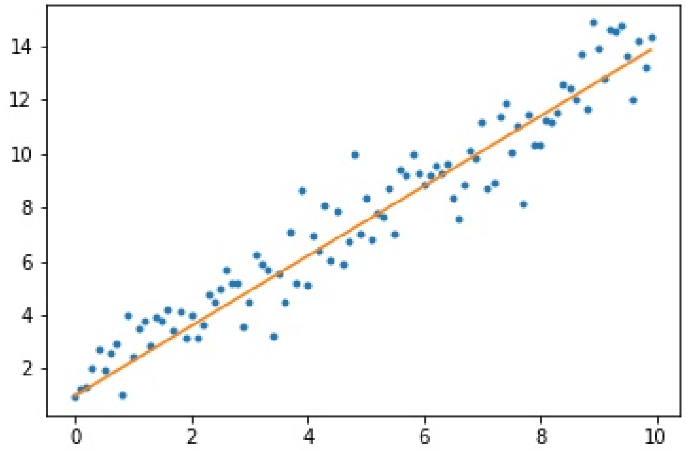

Figure 2.2: Data set with clear trend, best fitting line included.

⋆ Although our input data points $\mathbf{x}$ can take on multiple dimensions, our output data y is always a 1-dimensional real number when dealing with regression problems.

Now that we have some intuition for what linear regression is, a natural question arises: how do we find the optimal values for $\mathbf{w}$? That is the remaining focus of this chapter.

2.2.3 Finding the Best Fitting Line: Loss

We can inspect a set of data as presented in Figure 2.1, and are quite good at fitting a line by hand. When we fit that line, whether we think about it or not, what we're trying to do is make the line as close to all of the data points as possible. We can formalize this notion by introducing the concept of **loss**, and use it to our advantage in developing a method to automatically fit a line to our data.

Definition 2.2.1 (Loss): Loss is the error incurred for a given prediction. It can be thought of as a measurement of difference between the target (t) and predicted ($y(\mathbf{x}, \mathbf{w})$) values:

$$\mathcal{L}(\mathbf{w}) = \text{target} - \text{prediction} = t - y(\mathbf{x}, \mathbf{w}) = \boxed{t - \mathbf{w}^T\mathbf{x}}$$

Notice that loss is denoted $\mathcal{L}(\mathbf{w})$, and it is a function of our parameters $\mathbf{w}$, which are used to generate our prediction. Loss can be computed in different manners, and the choice of what loss function to use has important implications that we will discuss.

Loss is a concept that we will come back to very frequently in the context of supervised machine learning methods.

2.2.4 Least Squares Loss

As we mentioned above, there are different methods for computing loss. One of the most commonly used measurements is known as **Least Squares Loss**. Least squares, as it is often abbreviated,

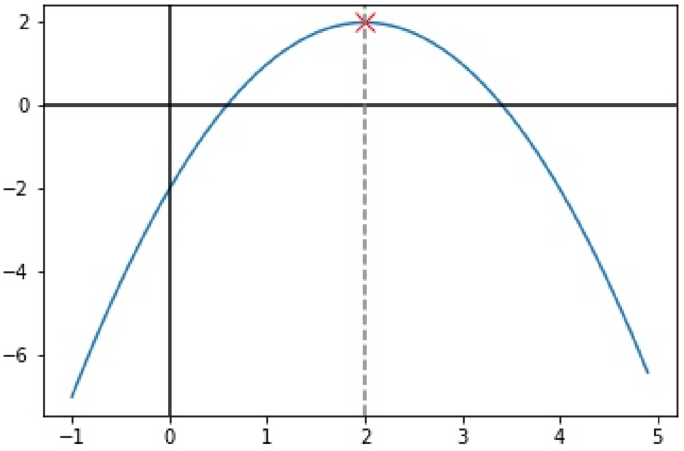

Figure 2.3: Quadratic function with clear optimum at $x = 2$, where the derivative of the function is 0.

says that the loss for a given data point is the square of the difference between the target and predicted values:

$$\mathcal{L}(\mathbf{w}) = (t - \mathbf{w}^T\mathbf{x})^2 \tag{2.5}$$

There is a satisfying statistical interpretation for using this loss function which we will explain later in this chapter, but for now it will suffice to discuss some of the properties of this loss function that make it desirable.

First, notice that it will always take on positive values. This is convenient because we can focus exclusively on minimizing our loss, and it also allows us to combine the loss incurred from different data points without worrying about them cancelling out.

A more subtle but enormously important property of this loss function is that we know a lot about how to efficiently optimize quadratic functions. This is not a textbook about optimization, but some quick and dirty intuition that we will take advantage of throughout this book is that we can easily and reliably take the derivative of quadratic functions because they are continuously differentiable. We also know that optima of a quadrative function will be located at points where the derivative of the function is equal to 0, as seen in Figure 2.3.

If we compare this to an alternative loss function, such as an absolute value loss:

$$\mathcal{L}(\mathbf{w}) = |t - \mathbf{w}^T\mathbf{x}| \tag{2.6}$$

We know that this function is not continuously differentiable around its optima, and as a result, it is much less convenient to work with as compared to least squares loss.

** add something about aversion to outliers using least squares loss? ** ** add a visualization of what the loss actually looks like for a toy data set (small little red lines from predicted to true) **

2.2.5 Optimal Weights via Matrix Differentiation

Now that we have our least squares loss function, we can finally begin to fit a line to our data. First, we can define the loss incurred by parameters $\mathbf{w}$ over our entire data set $\mathbf{X}$ as follows:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(t - \mathbf{w}^T\mathbf{x})^2 \tag{2.7}$$

⋆ Note that we added a constant $\frac{1}{2}$ to the beginning of our loss expression. This scales the loss, which will not change our final result for the optimal parameters. It has the benefit of making our calculations cleaner once we've taken the gradient of the loss.

We now want to solve for the values of $\mathbf{w}$ that minimize this expression, since a small loss implies that we are doing a good job of predicting values that are close to their true target values.

Derivation 2.2.1 (Least Squares Derivation): We find the optimal weights $\mathbf{w}^*$ as follows:

Start by taking the gradient of the loss with respect to our parameter $\mathbf{w}$:

$$\nabla\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{n}(t - \mathbf{w}^T\mathbf{x})\mathbf{x}^T$$

Setting this gradient to 0, we have:

$$0 = \sum_{i=1}^{n} t\mathbf{x}^T - \mathbf{w}^T\sum_{i=1}^{n}\mathbf{x}\mathbf{x}^T \tag{2.8}$$

At this point, it is convenient to rewrite these summations as matrix operations in terms of $\mathbf{w}$ instead of $\mathbf{w}^T$. Note that $(\mathbf{w}^T\sum_{i=1}^{n}\mathbf{x}\mathbf{x}^T)^T = \mathbf{X}^T\mathbf{X}\mathbf{w}$ and $(\sum_{i=1}^{n} t\mathbf{x}^T)^T = \mathbf{X}^T\mathbf{Y}$. Rewriting:

$$0 = \mathbf{X}^T\mathbf{Y} - \mathbf{X}^T\mathbf{X}\mathbf{w}$$

Solving for $\mathbf{w}^*$:

$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \tag{2.9}$$

Note that the quantity $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ in Derivation 2.2.1 has a special name: the ***Moore-Penrose pseudo inverse***. You can think of it as a generalization of a matrix inversion operation to a non-square matrix.

2.2.6 Linear Regression as Projection

Another common interpretation of linear regression is that of a projection of our targets, $\mathbf{Y}$, onto the column space of our inputs $\mathbf{X}$. This can be useful for building intuition.

We showed above that the quantity $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ can be thought of as the pseudoinverse for our inputs $\mathbf{X}$. Let's now consider the case where $\mathbf{X}$ is square and the pseudoinverse is equal to the true inverse: $\mathbf{X}^{-1} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$. We have for our optimal $\mathbf{w}^*$:

$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

which becomes

$$\mathbf{w}^* = \mathbf{X}^{-1}\mathbf{Y}$$

We can recover our target values $\mathbf{Y}$ by multiplying either side by $\mathbf{X}$:

$$\begin{aligned}\mathbf{X}\mathbf{w}^* &= \mathbf{X}\mathbf{X}^{-1}\mathbf{Y} \\ \mathbf{X}\mathbf{w}^* &= \mathbf{Y}\end{aligned}$$

We were able to recover our targets $\mathbf{Y}$ exactly because $\mathbf{X}$ is an invertible tranformation. However, in the general case where $\mathbf{X}$ is not invertible and we have to use the approximate pseudoinverse $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$, we instead recover $\hat{\mathbf{Y}}$:

$$\begin{aligned}\mathbf{X}\mathbf{w}^* &= \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \\ \mathbf{X}\mathbf{w}^* &= \hat{\mathbf{Y}}\end{aligned}$$

where $\hat{\mathbf{Y}}$ can be thought of as the closest projection of $\mathbf{Y}$ into the column space of $\mathbf{X}$. Furthermore, this motivates the intuition that $\mathbf{w}^*$ is the set of coefficients that best transforms our input space $\mathbf{X}$ into our target values $\mathbf{Y}$.

2.2.7 Basis Functions

There are some situations where our input data $\mathbf{X}$ is not the best form of our data for performing linear regression. Because linear regression only scales and combines input variables, it is unable to apply more complex transformations to our data such as a *sin* or square root function. In those situations where we need to transform our input variable somehow prior to performing linear regression (which is known as moving our data into a new *basis*), we apply what is known as a **basis function**.

Definition 2.2.2 (Basis Function): Typically denoted by the symbol $\phi(\cdot)$, a basis function is a transformation applied to an input data point $\mathbf{x}$ to move our data into a different *input basis*, which is another phrase for *input domain.*

For example, consider our original data point:

$$\mathbf{x} = (x_1, x_2)'$$

We may choose our basis function $\phi(\mathbf{x})$ such that our transformed data point in its new basis is:

$$\mathbf{x} = (x_1, x_1^2, x_2, \sin(x_2))'$$

Using a basis function is so common that we will sometimes describe our input data points as $\boldsymbol{\phi} = (\phi_1, \phi_2, ..., \phi_D)'$.

Basis functions are very general - they could specify that we just keep our input data the same. As a result, it's common to rewrite the least squares loss function from Equation 2.5 for linear regression in terms of the basis function applied to our input data:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(t - \mathbf{w}^T\boldsymbol{\phi})^2 \tag{2.10}$$

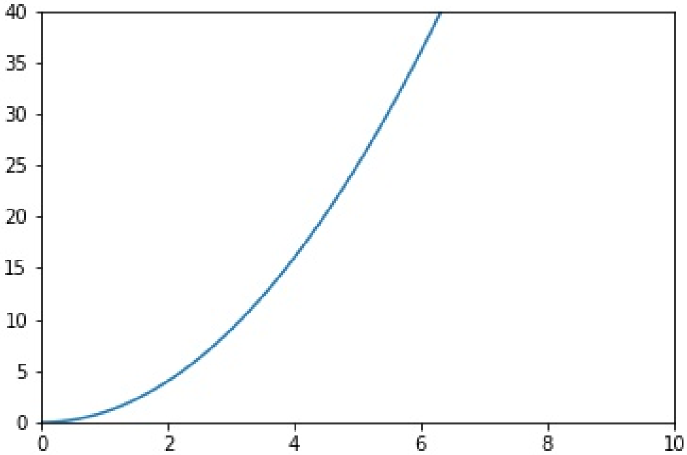

Figure 2.4: Data with no basis function applied.

To motivate why we might need basis functions for performing linear regression, let's consider this graph of 1-dimensional inputs $\mathbf{X}$ along with their target outputs $\mathbf{Y}$, presented in Figure 2.4.

As we can see, we're not going to be able to fit a good line to this data. The best we can hope to do is something like that of Figure 2.5.

However, if we just apply a simple basis function to our data, in this case the square root function, $\phi(\mathbf{x}) = (\sqrt{x_1})'$, we then have the red line in Figure 2.6.

We now see that we can fit a very good line to our data, thanks to basis functions. Still, the logical question remains: how can I choose the appropriate basis function? This toy example had a very obviously good basis function, but in general with high dimensional, messy input data, how do we choose the basis function we need?

The answer is that this is not an easy problem to solve. Often, you may have some domain specific knowledge that tells you to try a certain basis, such as if you're working with chemical data and know that an important equation involves a certain function of one of your inputs. However, more often than not we won't have this expert knowledge either. Later on, we will discuss methods for discovering the best basis functions for our data automatically.

2.2.8 Regularization

Now that we've introduced the idea of basis functions, you might wonder why we don't just try adding many basis transformations to our input data to find a good transformation. For example, we might use this large basis function:

$$\phi(\mathbf{x}) = (x_1, x_1^2, ..., x_1^{100}, x_2, x_2^2, ..., x_2^{100}, x_D, x_D^2, ..., x_D^{100})'$$

Let's say our input data point $\mathbf{x}$ is 1-dimensional, and we apply the basis function described above, so that after the transformation each data point is represented by 100 values. Say we have 100 data points on which to perform linear regression, and because our transformed input space has 100 values, we have 100 parameters to fit. In this case, with one parameter per data point, it's possible for us to fit our regression line perfectly to our data so that we have 0 loss! But is this a desirable outcome? The answer is no, and we'll provide a visual example to illustrate that.

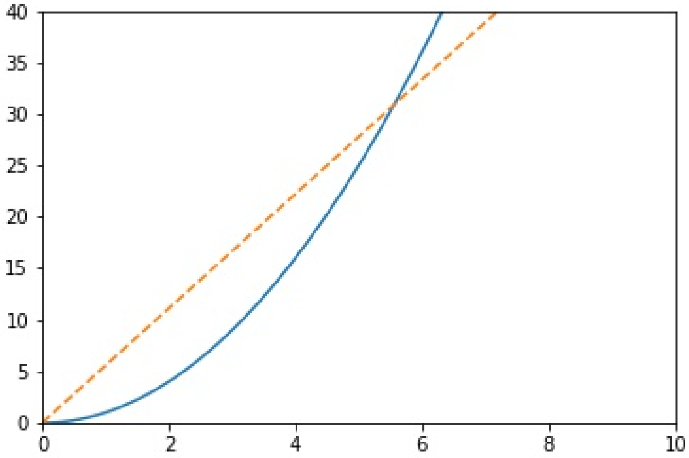

Figure 2.5: Data with no basis function applied, attempt to fit a line.

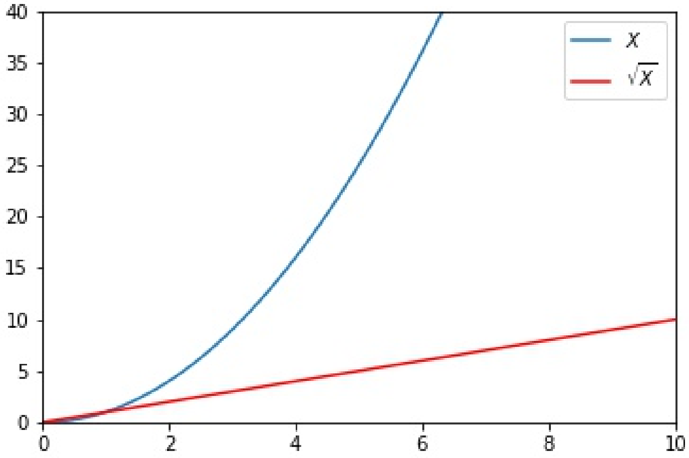

Figure 2.6: Data with square root basis function applied.

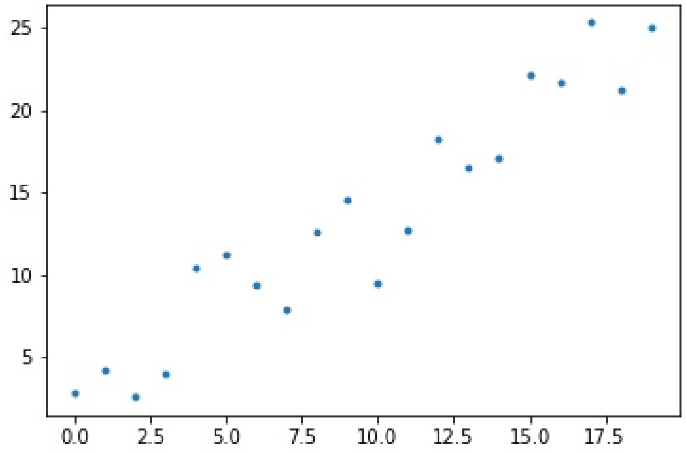

Figure 2.7: Data set with a clear trend and Gaussian noise.

Imagine Figure 2.7 is our data set. There is a very clear trend in this data, and you would likely draw a line that looks something like that of Figure 2.8 to fit it.

However, imagine now we performed a large basis transformation like the one described above. If we do that, it's possible for us to fit our line perfectly, threading every data point, like that in Figure 2.9.

Let's see how both of these would perform on new data points. With our first regression line, if we have a new data point $\mathbf{x} = (10)'$, we would predict a target value of **14.1**, which most people would agree is a pretty good measurement. However, with the second regression line, we would predict a value of **9.5**, which most people would agree does not describe the general trend in the data. So how can we handle this problem elegantly?

Examining our loss function in Equation 2.10, we see that right now we're only penalizing predictions that are not correct. Intuitively, if we wanted to avoid the problem of recovering a convoluted line as our solution, we should also penalize the total size of our weights $\mathbf{w}$. The effect of this is to discourage lots of complex weight values that produce a messy regression line. Instead, by penalizing large weights, we favor simple regression lines like the one in Figure 2.8 that take advantage of only the most important basis functions.

The concept that we are introducing, penalizing large weights, is what is known as **regularization**, and it's one that we will see come up often in different machine learning methods.

Definition 2.2.3 (Regularization): The idea of adding a loss for large weight values, which encourages simple solutions that take advantage of the most important information from our data.

There is obviously a tradeoff between how aggressively we regularize our weights and how tightly our solution fits to our data, and we will formalize this tradeoff in the next section. However, for now, we will simply introduce a regularization parameter λ to our least squares loss function:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(t - \mathbf{w}^T\boldsymbol{\phi})^2 + \frac{\lambda}{2}\mathbf{w}^2 \tag{2.11}$$

The effect of λ is to penalize large weight parameters. The larger λ is, the more we will favor

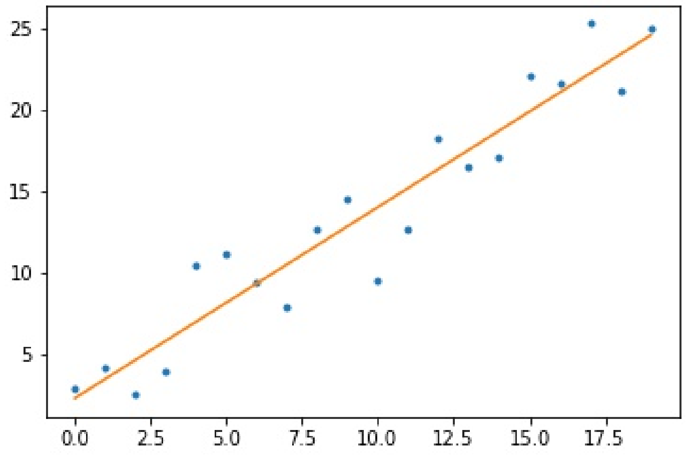

Figure 2.8: Natural fit for this data set.

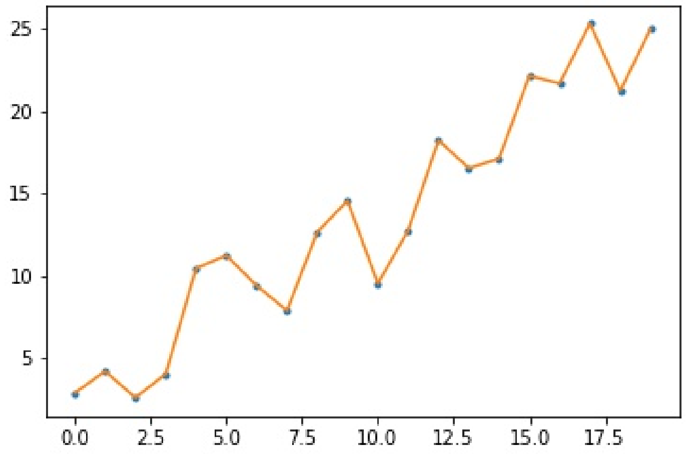

Figure 2.9: Unnatural fit for this data set.

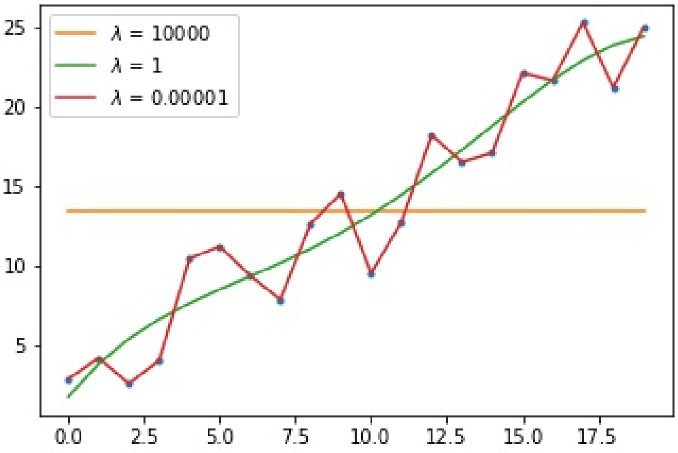

Figure 2.10: Effect of different regularization parameter values on final regression solution.

simple solutions. In the limit $\lim_{\lambda\to\infty}\mathcal{L}(\mathbf{w})$, we will drive all weights to 0, while with a nonexistant $\lambda = 0$ we will apply no regularization at all. Notice that we're squaring our weight parameters - this is known as *L2 norm regularization* or **ridge regression**. We will discuss the different types of regularization later on, but for now just know that L2 norm regularization is very common.

To build some intuition about this regularization parameter, let's see how different values of it will produce different regression lines:

Notice how larger values of λ produce less complex lines, which is the effect of applying more regularization. This is very nice for the problem we started with - which was wanting a way to choose which basis functions we wanted to use. With regularization, we can select many basis functions, and then allow regularization to 'prune' the ones that aren't meaningful (by driving their weight parameters to 0). While this doesn't mean that we should use as many basis transformations as possible (there will be computational overhead for doing this), it does allow us to create a much more flexible linear regression model without creating a convoluted regression line.

2.2.9 Bias-Variance Tradeoff and Decomposition

Now that you know about regularization, you might have some intuition for why we need to find a balance between complex and simple regression solutions. A complex solution, while it might fit all of our training data, may not generalize well to future data points. On the other hand, a line that is too simple might not vary enough to provide good predictions at all. This phenomenon is not unique to linear regression- it's actually a very fundamental concept in machine learning that's known as the **bias-variance tradeoff**.

Definition 2.2.4 (Bias-Variance Tradeoff): When construction machine learning models, we have a choice somewhere on a spectrum between two extremes: fitting exactly to our training data or not varying in response to our training data at all. The first extreme, fitting all of our training data, is a situation of high *variance*, because our output changes heavily in reponse to our input data (see the red line in Figure 2.10). At the other extreme, a solution that doesn't change in response to our training data at all, is a situation of high *bias*. This means that our target output doesn't fluctuate at all as a result of our training data.

Obviously a good solution will fall somewhere in between these two extremes of total variance and total bias. Indeed, we have techniques like regularization that help us balance the two extremes, and we have additional techniques like *cross-validation* that help us determine when we have found a good balance between the two.

⋆ In case you are not familiar with the terms *bias* and *variance*, we provide their statistical definitions here:

$$\text{bias}(\theta) = \text{E}[\theta] - \theta$$

$$\text{variance}(\theta) = \text{E}[(\theta - \text{E}[\theta])^2]$$

Before we discuss how to effectively balance these two outcomes, we will first show that the bias-variance tradeoff is not only conceptual but also has probabilistic underpinnings. Specifically, any loss that we incur over our training set using a given model can be described in terms of bias and variance, as we will demonstrate now.

Derivation 2.2.2 (Bias-Variance Decomposition): Let's begin by asserting that we have a model $f(\cdot)$ that makes a prediction of our target y given input data point $\mathbf{x}$. We wish to break down the squared error of f into terms involving bias and variance.

Start with the expected squared error (MSE), where the expectation is taken with respect to both our data set $\mathbf{D}$, which is composed of ($\mathbf{x}$, y) pairs, and our conditional distribution $y|\mathbf{x}$:

$$MSE = \text{E}[(y - f(\mathbf{x}))^2]$$

For reasons that will become clear in a few steps, add and subtract our target mean, $\bar{y}$, inside of the squared term:

$$MSE = \text{E}[(y - \bar{y} + \bar{y} - f(\mathbf{x}))^2]$$

Group together the first two terms and the last two terms:

$$MSE = \text{E}[((y - \bar{y}) + (\bar{y} - f(\mathbf{x})))^2]$$

Expanding this expression and using linearity of expectation:

$$MSE = \text{E}[(y - \bar{y})^2] + \text{E}[(\bar{y} - f(\mathbf{x}))^2] + 2\text{E}[(y - \bar{y})(\bar{y} - f(\mathbf{x}))] \tag{2.12}$$

Let's examine the last term, $2\text{E}[(y - \bar{y})(\bar{y} - f(\mathbf{x}))]$. Notice that $(\bar{y} - f(\mathbf{x}))$ does not depend on the conditional distribution $y|\mathbf{x}$ at all. Thus, we are able to move one of those expecations in, which makes this term:

$$2\text{E}[(y - \bar{y})(\bar{y} - f(\mathbf{x}))] = 2\text{E}_D[\text{E}_{y|\mathbf{x}}[(y - \bar{y})](\bar{y} - f(\mathbf{x}))]$$

And note that:

$$\text{E}_{y|\mathbf{x}}[(y - \bar{y})] = 0$$

Which eliminates this last term entirely:

$$2\mathrm{E}[(y-\bar{y})(\bar{y}-f(\mathbf{x}))] = 2\mathrm{E}_D[0 \cdot (\bar{y}-f(\mathbf{x}))] = 0$$

We can now write Equation 2.12 as:

$$MSE = \mathrm{E}[(y-\bar{y})^2] + \mathrm{E}[(\bar{y}-f(\mathbf{x}))^2] \tag{2.13}$$

We now have two terms contributing to our squared error. We will ignore the first term $\mathrm{E}[(y-\bar{y})^2]$, as this is unidentifiable *noise* in our data set. In other words, our data will randomly deviate from the mean in ways we cannot predict. On the other hand, we can work with the second term $\mathrm{E}[(\bar{y}-f(\mathbf{x}))^2]$ as it involves our model function $f(\cdot)$

As before, for reasons that will become clear in a few steps, let's add and subtract the mean of our prediction, $\bar{f}(\cdot)$, inside this expression:

$$\mathrm{E}[(\bar{y}-f(\mathbf{x}))^2] = \mathrm{E}[(\bar{y}-\bar{f}(\mathbf{x})+\bar{f}(\mathbf{x})-f(\mathbf{x}))^2]$$

Expanding this squared term, we have:

$$\mathrm{E}[(\bar{y}-f(\mathbf{x}))^2] = (\bar{y}-\bar{f}(\mathbf{x}))^2 + \mathrm{E}[(\bar{f}(\mathbf{x})-f(\mathbf{x}))^2] + 2\mathrm{E}[(\bar{y}-\bar{f}(\mathbf{x}))(\bar{f}(\mathbf{x})-f(\mathbf{x}))]$$

As before, the third term here is 0:

$$2\mathrm{E}[(\bar{y}-\bar{f}(\mathbf{x}))(\bar{f}(\mathbf{x})-f(\mathbf{x}))] = 2(\bar{y}-\bar{f}(\mathbf{x}))\mathrm{E}[(\bar{f}(\mathbf{x})-f(\mathbf{x}))] = 2(\bar{y}-\bar{f}(\mathbf{x}))(0) = 0$$

Leaving us with these two terms:

$$\mathrm{E}[(\bar{y}-f(\mathbf{x}))^2] = (\bar{y}-\bar{f}(\mathbf{x}))^2 + \mathrm{E}[(\bar{f}(\mathbf{x})-f(\mathbf{x}))^2]$$

Notice the form of these two terms. The first one, $(\bar{y}-\bar{f}(\mathbf{x}))^2$, is the squared *bias* of our model. The second one, $\mathrm{E}[(\bar{f}(\mathbf{x})-f(\mathbf{x}))^2]$, is the *variance* of our model:

$$\mathrm{E}[(y-f(\mathbf{x}))^2] = \mathit{bias}(f(\mathbf{x}))^2 + \mathit{variance}(f(\mathbf{x}))$$

Thus, our total squared error, plugging in to Equation 2.13 can be written as:

$$\boxed{MSE = \mathit{noise}(\mathbf{x}) + \mathit{bias}(f(\mathbf{x}))^2 + \mathit{variance}(f(\mathbf{x}))}$$

The key takeaway of the bias-variance decomposition is that the controllable error in our model is given by the squared bias and variance. Holding our error constant, to decrease bias requires increasing the variance in our model, and vice-versa. In general, a graph of the source of error in our model might look something like Figure 2.11.

For a moment, consider what happens on the far left side of this graph. Our variance is very high, and our bias is very low. In effect, we're fitting perfectly to all of the data in our data set. This is exactly why we introduced the idea of regularization from before - we're fitting a very convoluted line that is able to pass through all of our data but which doesn't generalize well to new data points. There is a name for this: **overfitting**.

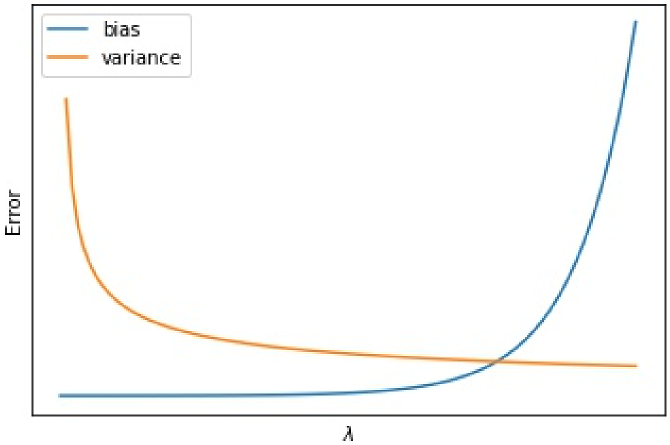

Figure 2.11: Bias and variance both contribute to the overall error of our model.

Definition 2.2.5 (Overfitting): A phenomenon where we construct a convoluted model that is able to predict every point in our data set perfectly but which doesn't generalize well to new data points.

The opposite idea, **underfitting**, is what happens at the far right of the graph: we have high bias and aren't responding to the variation in our data set at all.

Definition 2.2.6 (Underfitting): A phenomenon where we construct a model that doesn't respond to variation in our data.

So you can hopefully now see that the bias-variance tradeoff is important to managing the problem of overfitting and underfitting. Too much variance in our model and we'll overfit to our data set. Too much bias and we won't account for the trends in our data set at all.

In general, we would like to find a sweet spot of moderate bias and variance that produces minimal error. In the next section, we will explore how we find this sweet spot.

2.2.10 Cross-Validation

We've seen that in choosing a model, we incur error that can be described in terms of bias and variance. We've also seen that we can regulate the source of error through regularization, where heavier regularization increases the bias of our model. A natural question then is how do we know how much regularization to apply to achieve a good balance of bias and variance?

Another way to look at this is that we've traded the question of finding the optimal number of basis functions for finding the optimal value of the regularization parameter λ, which is often an easier problem in most contexts.

One solution to finding the sweet spot for our regularization parameter, or any hyperparemter for that matter, is known as **cross-validation**.

Definition 2.2.7 (Cross-Validation): A subsampling procedure used over a data set to tune hyperparameters and avoid over-fitting. Some portion of a data set (10-20% is common) is set aside, and training is performed on the remaining, larger portion of data. When training is complete, the smaller portion of data left out of training is used for testing. The larger portion of data is sometimes referred to as the *training set*, and the smaller portion is sometimes referred to as the *validation set.*

Cross-validation is often performed more than once for a given setting of hyperparameters to avoid a skewed set of validation data being selected by chance. In **K-Folds cross-validation**, you perform cross-validation **K** times, allocating $\frac{1}{\mathbf{K}}$ of your data for the validation set at each iteration.

Let's tie this back into finding a good regularization parameter. For a given value of λ, we will incur a certain amount of error in our model. We can measure this error using cross-validation, where we train our model on the training set and compute the final error using the validation set. To find the optimal value for λ, we perform cross-validation using different values of λ, eventually settling on the value that produces the lowest final error. This will effectively trade off bias and variance, finding the value of λ that minimizes the total error.

You might wonder why we need to perform cross-validation at all - why can't we train on the entire data set and then compute the error over the entire data set as well?

The answer is again overfitting. If we train over the entire data set and then validate our results on the exact same data set, we are likely to choose a regularization parameter that encourages our model to conform to the exact variation in our data set instead of finding the generalizable trends. By training on one set of data, and then validating on a completely different set of data, we force our model to find good generalizations in our data set. This ultimately allows us to pick the regularization term λ that finds the sweet spot between bias and variance, overfitting and underfitting.

2.2.11 Bayesian Linear Regression

We've thus far been discussing linear regression exclusively in terms of a loss function that helps us fit a set of weights to our data. In particular, we have been working with least squares, which has nice properties that make it a reasonable loss function.

In a very satisfying fashion, least squares also has a statistical foundation. In fact, you can recover the least squares loss function purely from a statistical derivation that we present here.

Consider our data set $\mathbf{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, $\mathbf{x}_i \in \mathbb{R}^m$, $y \in \mathbb{R}$. Let's imagine that our data was generated according to the following process:

$$y_i \sim \mathcal{N}(\mathbf{w}^T\phi(\mathbf{x}_i), \beta^{-1})$$

Which can be written equivalently as:

$$p(y_i|\mathbf{x}_i, \mathbf{w}, \beta) = \mathcal{N}(\mathbf{w}^T\phi(\mathbf{x}_i), \beta^{-1}) \tag{2.14}$$

The interpretation of this is that our target value y is generated according to a linear combination of our inputs $\mathbf{x}$, but there is also some noise in the data generating process described by the variance parameter β^{-1}. It's an acknowledgement that some noise exists naturally in our data.

⋆ It's common to write variance as an inverse term, such as β^{-1}. The parameter β is then known as the *precision*, which is sometimes easier to work with than the variance.

As before, we now ask the question: how do we solve for the optimal weights $\mathbf{w}$? One approach we can take is to maximize the probability of observing our target data $\mathbf{Y}$. This technique is known as *maximum likelihood estimation*.

Derivation 2.2.3 (Maximum Likelihood Estimation for Bayesian Linear Regression): The likelihood of our data set is given by:

$$p(\mathbf{Y}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}(\mathbf{w}^T \phi(\mathbf{x}_i), \beta^{-1})$$

We then take the logarithm of the likelihood, and since the logarithm is a strictly increasing, continuous function, this will not change our optimal weights $\mathbf{w}$:

$$\ln p(\mathbf{Y}|\mathbf{X}, \mathbf{w}, \beta) = \sum_{i=1}^{N} \ln \mathcal{N}(\mathbf{w}^T \phi(\mathbf{x}_i), \beta^{-1})$$

Using the density function of a univariate Gaussian:

$$\begin{aligned} \ln p(\mathbf{Y}|\mathbf{X}, \mathbf{w}, \beta) &= \sum_{i=1}^{N} \ln \frac{1}{\sqrt{2\pi\beta^{-1}}} e^{-(y_i - \mathbf{w}^T \phi(\mathbf{x}_i))^2 / 2\beta^{-1}} \\ &= \frac{N}{2} \ln \beta - \frac{N}{2} \ln (2\pi) - \frac{\beta}{2} \sum_{i=1}^{N} (y_i - \mathbf{w}^T \phi(\mathbf{x}_i))^2 \end{aligned}$$

Notice that this is a quadratic function in $\mathbf{w}$, which means that we can solve for it by taking the derivative with respect to $\mathbf{w}$, setting that expression to 0, and solving for $\mathbf{w}$:

$$\frac{\partial \ln p(\mathbf{Y}|\mathbf{X}, \mathbf{w}, \beta)}{\partial \mathbf{w}} = \beta \sum_{i=1}^{N} (y_i - \mathbf{w}^T \phi(\mathbf{x}_i)) \phi(\mathbf{x}_i)^T$$

$$0 = \beta \sum_{i=1}^{N} (y_i - \mathbf{w}^T \phi(\mathbf{x}_i)) \phi(\mathbf{x}_i)^T$$

$$0 = \sum_{i=1}^{N} y_i \phi(\mathbf{x}_i)^T - \mathbf{w}^T \sum_{i=1}^{N} \phi(\mathbf{x}_i) \phi(\mathbf{x}_i)^T$$

Notice that this is exactly the same form as Equation 2.8. Solving for $\mathbf{w}$ as before:

$$\mathbf{w}^* = (\phi(\mathbf{X})^T \phi(\mathbf{X}))^{-1} \phi(\mathbf{X})^T \mathbf{Y}$$

Note that you'll often see $\mathbf{X} = \mathbf{\Phi}$, which is known as the *design matrix*, giving us:

$$\mathbf{w}^* = (\mathbf{\Phi}^T \mathbf{\Phi})^{-1} \mathbf{\Phi}^T \mathbf{Y} \tag{2.15}$$

Notice that our final solution is exactly the same form as the solution in Equation 2.9, which we solved for by minimizing the least squares loss! **The takeaway here is that minimizing**

a least squares loss function is equivalent to maximizing the probability under the assumption of a linear model with Gaussian noise.

2.2.12 Regularization Interpretation

We've thus far only discussed one form of regularization: ridge regression. Remember that under ridge regression, the loss function takes the form:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(t - \mathbf{w}^T\boldsymbol{\phi})^2 + \frac{\lambda}{2}\mathbf{w}^2$$

Where the $\frac{\lambda}{2}\mathbf{w}^2$ term is for the regularization. We can generalize our type of regularization by writing it as:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(t - \mathbf{w}^T\boldsymbol{\phi})^2 + \frac{\lambda}{2}\left|\mathbf{w}\right|^h$$

where h determines the type of regularization we are using and thus the form of the optimal solution that we recover. The three most commonly used forms of regularization are lasso, ridge, and elastic net.

Ridge Regression

This is the case of $h = 2$, which we've already discussed, but what type of solutions does it tend to uncover? Ridge regression prevents any individual weight from growing too large, providing us with solutions that are generally moderate.

Lasso Regression

Lasso regression is the case of $h = 1$. Unlike ridge regression, lasso regression will drive some parameters $\mathbf{w}_i$ to zero if they aren't informative for our final solution. Thus, lasso regression is good if you wish to recover a sparse solution that will allow you to throw out some of your basis functions.

You can see the forms of ridge and lasso regression in Figure 2.12.

Elastic Net

Elastic net is a middle ground between ridge and lasso regression, which it achieves by using a linear combination of the previous two regularization terms. Depending on how heavily each regularization term is weighted, this can produce results on a spectrum between lasso and ridge regression.

2.2.13 Bayesian Regularization

We've seen regularization in the context of loss functions, where the goal is to penalize large weight values. How does the concept of regularization apply to Bayesian linear regression?

The answer is that we can interpret regularizing out weight parameters as adding a prior distribution over $\mathbf{w}$.

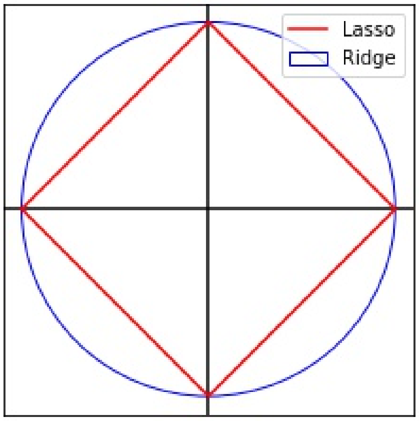

Figure 2.12: Form of the ridge (blue) and lasso (red) regression functions.

Derivation 2.2.4 (Bayesian Regularization Derivation): Because we wish to shrink our weight values toward 0 (which is exactly what regularization does), we will select a Normal prior with mean 0 and variance $\boldsymbol{S}_0^{-1}$:

$$\mathbf{w} \sim \mathcal{N}(0, \boldsymbol{S}_0^{-1}\mathbf{I})$$

Remember that the distribution over our observed data is Normal as well from Equation 2.14, written here in terms of our entire data set:

$$p(\mathbf{Y}|\mathbf{X}, \mathbf{w}, \beta) = \mathcal{N}(\mathbf{X}\mathbf{w}, \beta^{-1}\mathbf{I})$$

We want to combine the likelihood and the prior to recover the posterior distribution of $\mathbf{w}$, which follows directly from Bayes' Theorem:

$$\underbrace{p(\mathbf{w}|\mathbf{X}, \mathbf{Y}, \beta)}_{\text{posterior}} \propto \underbrace{p(\mathbf{Y}|\mathbf{X}, \mathbf{w}, \beta)}_{\text{likelihood}} \underbrace{p(\mathbf{w})}_{\text{prior}}$$

We now wish to find the value of $\mathbf{w}$ that maximizes the posterior distribution. We can maximize the log of the posterior with respect to $\mathbf{w}$, which simplifies the problem slightly:

$$\ln p(\mathbf{w}|\mathbf{X}, \mathbf{Y}, \beta) \propto \ln p(\mathbf{Y}|\mathbf{X}, \mathbf{w}, \beta) + \ln p(\mathbf{w})$$

Let's handle $\ln p(\mathbf{Y}|\mathbf{X}, \mathbf{w}, \beta)$ first:

$$\begin{aligned}
\ln p(\mathbf{Y}|\mathbf{X}, \mathbf{w}, \beta) &= \ln \prod_{i=1}^{N} \mathcal{N}(y_i|\mathbf{w}^T\mathbf{x}_i, \beta^{-1}) \\
&= \ln \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\beta^{-1}}} \exp\left\{ -\frac{\beta}{2}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \right\} \\
&= \mathbf{C} - \frac{\beta}{2}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \frac{1}{\sqrt{2\pi\beta^{-1}}}
\end{aligned}$$

where $\mathbf{C}$ collects the constant terms that don't depend on $\mathbf{w}$. Let's now handle $\ln p(\mathbf{w})$:

$$\begin{aligned}\ln p(\mathbf{w}) &= \ln \mathcal{N}(0, \boldsymbol{S}_0^{-1}\mathbf{I}) \\ &= \ln \frac{1}{(|2\pi \boldsymbol{S}_0^{-1}\mathbf{I}|)^{\frac{1}{2}}} \exp\left\{-\frac{\boldsymbol{S}_0}{2}\mathbf{w}^T\mathbf{w}\right\} \\ &= \mathbf{C} - \frac{\boldsymbol{S}_0}{2}\mathbf{w}^T\mathbf{w}\end{aligned}$$

combining the terms for $\ln p(\mathbf{Y}|\mathbf{X}, \mathbf{w}, \beta)$ and $\ln p(\mathbf{w})$:

$$\ln p(\mathbf{w}|\mathbf{X}, \mathbf{Y}, \beta) = -\frac{\beta}{2}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 - \frac{\boldsymbol{S}_0}{2}\mathbf{w}^T\mathbf{w}$$

dividing by a positive constant β:

$$\ln p(\mathbf{w}|\mathbf{X}, \mathbf{Y}, \beta) = -\frac{1}{2}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 - \frac{\boldsymbol{S}_0}{\beta}\frac{1}{2}\mathbf{w}^T\mathbf{w}$$

Notice that maximizing the posterior probability is equivalent to minimizing the sum of squared errors $(y_i - \mathbf{w}^T\mathbf{x}_i)^2$ and the regularization term $\mathbf{w}^T\mathbf{w}$.

The interpretation of this is that adding a prior over the distribution of our weight parameters w is equivalent to adding a regularization term where $\lambda = \frac{\boldsymbol{S}_0}{\beta}$

2.2.14 Predictive Distribution

Remaining in the setting of Bayesian Linear Regression, we may wish to get a distribution over our weights $\mathbf{w}$ instead of a point estimator for it using maximum likelihood. As we saw in Section 2.2.13, we can introduce a prior distribution over $\mathbf{w}$, then together with our observed data, we can produce a posterior distribution over $\mathbf{w}$ as desired.

Derivation 2.2.5 (Posterior Predictive Derivation): For the sake of simplicity and ease of use, we will select our prior over $\mathbf{w}$ to be a Normal distribution with mean $\boldsymbol{\mu}_0$ and variance $\boldsymbol{S}_0^{-1}$:

$$p(\mathbf{w}) = \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{S}_0^{-1})$$

Remembering that the observed data is normally distributed, and accounting for Normal-Normal conjugacy, our posterior distribution will be Normal as well:

$$p(\mathbf{w}|\mathbf{X}, \mathbf{Y}, \beta) = \mathcal{N}(\boldsymbol{\mu}_N, \boldsymbol{S}_N^{-1})$$

where

$$\begin{aligned}\boldsymbol{S}_N &= (\boldsymbol{S}_0^{-1} + \beta\mathbf{X}^T\mathbf{X})^{-1} \\ \boldsymbol{\mu}_N &= \boldsymbol{S}_N(\boldsymbol{S}_0^{-1}\boldsymbol{\mu}_0 + \beta\mathbf{X}\mathbf{Y})\end{aligned}$$

We now have a posterior distribution over $\mathbf{w}$. However, usually this distribution is not what we care about. We're actually interested in making a point prediction for the target y^* given a new input $\mathbf{x}^*$. How do we go from a posterior distribution over $\mathbf{w}$ to this prediction?

The answer is using what's known as the **posterior predictive** over y^* given by:

$$\begin{aligned} p(y^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) &= \int_{\mathbf{w}} p(y^*|\mathbf{x}^*, \mathbf{w})p(\mathbf{w}|\mathbf{X}, \mathbf{Y})d\mathbf{w} \\ &= \mathcal{N}(y^*|\mathbf{w}^T\mathbf{x}^*, \beta^{-1})\mathcal{N}(\mathbf{w}|\boldsymbol{\mu}_N, \boldsymbol{S}_N^{-1})d\mathbf{w} \end{aligned} \tag{2.16}$$

The idea here is to average the probability of y^* over all the possible setting of $\mathbf{w}$, weighting the probabilities by how likely each setting of $\mathbf{w}$ is according to its posterior distribution.

2.3 Conclusion

In this chapter, we looked at a specific tool for handling regression problems known as linear regression. We've seen linear regression described in terms of loss functions, geometric projections, and probabilistic expressions, which reflects the deep body of knowledge that we have around this very common technique.

We've also discussed many concepts in this chapter that will prove useful in other areas of machine learning, particularly for other supervised techniques: loss functions, regularization, bias and variance, over and underfitting, posterior distributions, maximum likelihood estimation, and cross-validation among others. Spending time to develop an understanding of these concepts now will pay off going forward.

It may or may not be obvious at this point that we are missing a technique for a very large class of problems: those where the solution is not just a continuous, real number. How do we handle situations where we need to make a choice between different discrete options? This is the question we will turn to in the next chapter.

2.4 Practice Problems

1. Bias and variance of an estimator.

2. Ridge regression derivation in matrix form.

3. Completing the square to derive the form of the posterior predictive.

4. Deriving lasso regularization using Lagrange multipliers.

5. MLE solution for variance.

6. Conceptual question: data set size and impact on bias/variance.

7. Conceptual question: using cross-validation.

8. Conceptual question: model generalization.