Machine Learning
the Fundamentals

William J. Deuschle

Harvard College
Cambridge, Massachusetts
April, 2019

Contents

Chapter 2

Regression

A major component of machine learning, the one that most people associate with ML, is dedicated to making predictions about some target given some inputs, such as predicting how much money an individual will earn in their lifetime given their demographic information. In this chapter, we're going to focus on the case where our prediction is a continuous, real number. When the target is a real number, we call this prediction procedure **regression**.

2.1 Defining the Problem

In order to understand regression, let's start in a more natural place: what types of problems are we trying to solve? What exactly does it mean to make a prediction that is a *continuous, real number*? To build some intuition, here are a few examples of problems that regression could be used for:

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.

Hopefully you are starting to see the pattern emerging here. Given some inputs, we need to produce a prediction for a continuous output. That is exactly the purpose of **regression**. Notice that regression isn't any one technique in particular! It's just a class of methods that helps us achieve our overall goal of predicting a continuous output.

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

2.2 Solution Options

Now that you understand the type of problems we are trying to solve with regression, we can start to think at a high level of the different ways we might arrive at a solution. Here is a short, nonexhaustive list of regression techniques with brief explanations:

2.2.1 K-Nearest-Neighbors

K-Nearest-Neighbors is an extremely intuitive, non-parametric technique for regression or classification. It works as follows in the regression case:

1. Identify the K points in our data set that are closest to the new data point. 'Closest' is some measure of distance, usually Euclidean.
2. Average the value of interest of those K data points.
3. Return that averaged value of interest: it is the prediction for our new data point.

⋆ A *non-parametric* model simply means we don't make any assumptions about the form of our data. We only need to use the data itself to make predictions.

2.2.2 Neural Networks

A neural network works by scaling and combining input variables many times. Furthermore, at each step of scaling and combining inputs, we typically apply some form of *nonlinearity* over our data values. The proof is beyond the scope of this textbook, but neural networks are known to be *universal function approximators*. This means that given enough scaling and combining steps, we can approximate any function to an arbitrarily high degree of accuracy using a neural network.

2.2.3 Random Forests

Random forests are what's known as an *ensemble method*. This means we average the results of many smaller regressors known as *decision trees* to produce a prediction. These decision trees individually operate by partitioning our original data set with respect to the value of predictive interest using a subset of the features present in that data set. Each decision tree individually may not be a reliable regressor; by combining many of them we achieve a model with better performance.

2.2.4 Gradient Boosted Trees

Gradient boosted trees are another technique built on decision trees. Assume we start with a set of decision trees that together achieve a certain level of performance. Then, we iteratively add new trees to improve performance on the hardest examples in our data set, and reweight our new set of decision trees to produce the best level of performance possible.

2.2.5 Turning to Linear Regression

You've likely never even heard of some of these preceding techniques - that's okay. The point is that we have a number of existing methods that take different approaches to solving regression problems. Each of these will have their own strengths and weaknesses that contribute to a decision about whether or not you might use them for a given regression problem. We obviously do not cover these methods in great detail here; what's more important is the understanding that there are a variety of ways to go about solving regression problems. From here on out, we will take a deeper dive into linear regression. There are several reasons for which we are exploring this specific technique in greater detail. The two main reasons are that it has been around for the longest and thus is very well understood, and it also will introduce many concepts and terms that will be utilized extensively in other machine learning topics.

2.3 Introduction to 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 2.3.1 (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 corresponds to the intercept b. It accounts for data that has a non-zero mean.

Let's illustrate how linear regression works using an example, considering the case of 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:

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

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** 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 **non-probabilistic**. Note that there also exist probabilistic interpretations of linear regression which we will discuss later in the chapter.

Domain	***Training***	***Probabilistic***
Continuous	Supervised	No

2.4 Basic Setup

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:

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

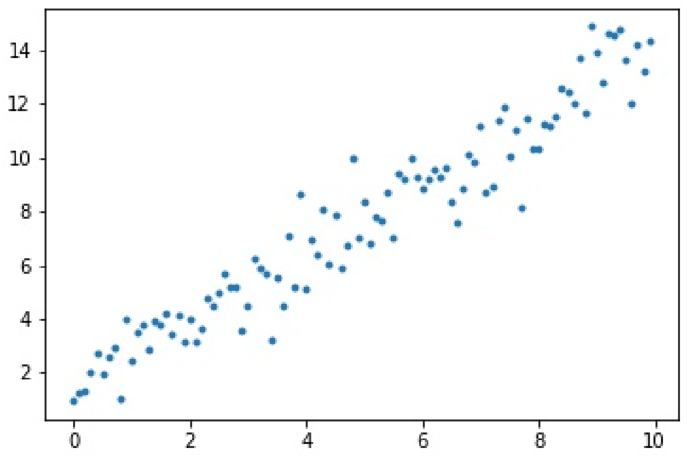

Figure 2.1: Data set with clear trend.

2.4.1 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. A simple way to make our bias term easier to handle 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:

$$f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T\mathbf{x} = w_0x_0 + w_1x_1 + ... + w_Dx_D \tag{2.3}$$

This is more compact, easier to reason about, and makes properties of linear algebra nicer for the calculations we will be performing.

2.4.2 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,

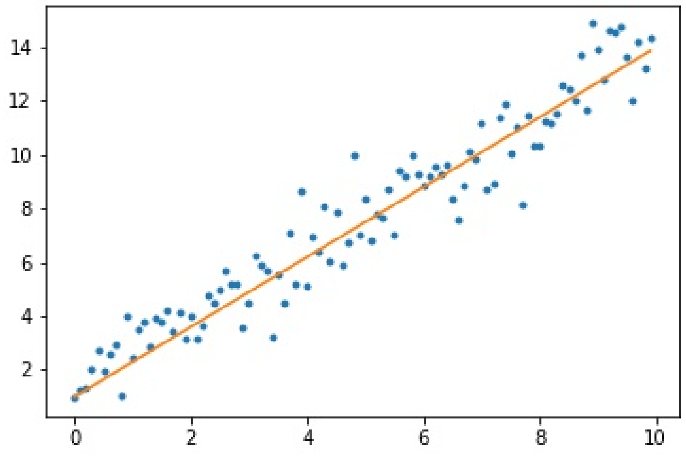

Figure 2.2: Data set with clear trend, best fitting line included.

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 hyperplane that best predicts the data.

$\star$ 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.

2.5 Finding the Best Fitting Line

TODO... we want some sort of intro and sectioning here

2.5.1 Objective Functions and Loss

Now that we've defined our model as a weighted combination of our input variables, we need some way to choose our value of $\mathbf{w}$. To do this, we need an **objective function**.

Definition 2.5.1 (Objective Function): A function that measures the 'goodness' of a model. We can optimize this function to identify the best possible model for our data.

As the definition explains, the purpose of an objective function is to measure how good a specific model is. We can therefore optimize this function to find a good model. Note that in the case of linear regression, our 'model' is just a setting of our parameters $\mathbf{w}$.

An objective function will sometimes be referred to as *loss*. Loss actually measures how bad a model is, and then our goal is to minimize it. It is common to think in terms of loss when discussing linear regression, and we incur loss when the hyperplane we fit is far away from our data.

So how do we compute the loss for a specific setting of $\mathbf{w}$? To do this, we often use **residuals**.

Definition 2.5.2 (Residual): The residual is the difference between the target (y) and predicted ($y(\mathbf{x}, \mathbf{w})$) value that a model produces:

$$\text{residual} = \text{target} - \text{prediction} = y - f(\mathbf{x}, \mathbf{w}) = \boxed{y - \mathbf{w}^T \mathbf{x}}$$

Commonly, loss is a function of the residuals produced by a model. For example, you can imagine taking the absolute value of all of the residuals and adding those up to produce a measurement of loss. This is sometimes referred to as *L1 Loss.* Or, you might square all of the residuals and then add those up to produce loss, which is called *L2 loss* or *least squares loss.* You might also use some combination of L1 and L2 loss. For the most part, these are the two most common forms of loss you will see when discussing linear regression.

When minimized, these distinct measurements of loss will produce solutions for $\mathbf{w}$ that have different properties. For example, L2 loss is not robust to outliers due to the fact that we are squaring residuals. Furthermore, L2 loss will produce only a single solution while L1 loss can potentially have many equivalent solutions. Finally, L1 loss produces unstable solutions, meaning that for small changes in our data set, we may see large changes in our solution $\mathbf{w}$.

Loss is a concept that we will come back to very frequently in the context of supervised machine learning methods. Before exploring exactly how we use loss to fit a line, let's consider least squares loss in greater depth.

2.5.2 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 or L2 loss**. Least squares, as it is often abbreviated, says that the loss for a given data point is the square of the difference between the target and predicted values:

$$\mathcal{L}_n(\mathbf{w}) = (y - \mathbf{w}^T \mathbf{x}_n)^2 \tag{2.4}$$

⋆ The notation $\mathcal{L}_n(\mathbf{w})$ is used to indicate the loss incurred by a model $\mathbf{w}$ for a single data point $(\mathbf{x}_n, \text{y})$. $\mathcal{L}(\mathbf{w})$ indicates the loss incurred for an entire data set by the model $\mathbf{w}$. Be aware that this notation is sometimes inconsistent between different sources.

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. In contrast, L1 loss is not continuously differentiable over the entirety of its domain.

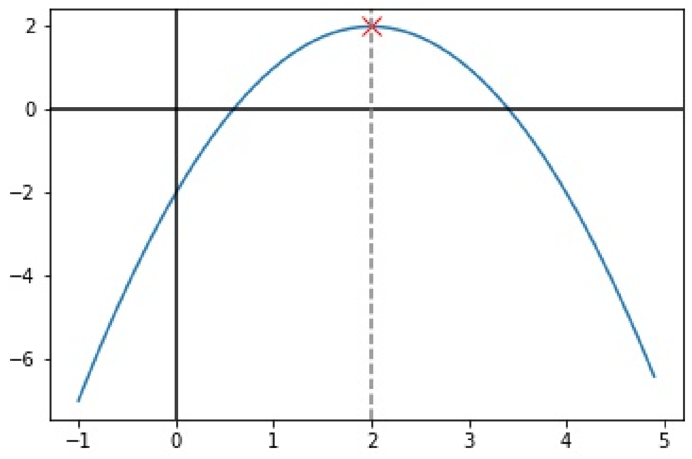

Figure 2.3: Quadratic function with clear optimum at $x = 2$, where the derivative of the function is 0.

2.5.3 Linear Regression as Projection

TODO: add pictures to this section? change because we no longer talk about the pseudo inverse? describe with projections directly?

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 $\mathbf{Y}$.

2.6 Linear Regression Algorithms

Now that we have our least squares loss function, we can finally begin to fit a line to our data. We will walk through the derivation of how this is done, in the process revealing the algorithm used for linear regression.

2.6.1 Optimal Weights via Matrix Differentiation

TODO: pickup here... First, we can define the loss incurred by parameters $\mathbf{w}$ over our entire data set $\mathbf{X}$ as follows:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(y_n - \mathbf{w}^T\mathbf{x}_n)^2 \tag{2.5}$$

⋆ 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.

Derivation 2.6.1 (Least Squares Optimal Weights Derivation): We find the optimal weights $\mathbf{w}^*$ as follows:

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

$$\nabla\mathcal{L}(\mathbf{w}) = \sum_{n=1}^{N}(y_n - \mathbf{w}^T\mathbf{x}_n)\mathbf{x}_n^T$$

Setting this gradient to 0, we have:

$$0 = \sum_{n=1}^{N} y_n\mathbf{x}_n^T - \mathbf{w}^T\sum_{n=1}^{N}\mathbf{x}_n\mathbf{x}_n^T \tag{2.6}$$

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 $(\mathbf{w}^T\sum_{n=1}^{N}\mathbf{x}_n\mathbf{x}_n^T)^T = \mathbf{X}^T\mathbf{X}\mathbf{w}$ and $(\sum_{n=1}^{N} y_n\mathbf{x}_n^T)^T = \mathbf{X}^T\mathbf{Y}$. Rewriting:

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

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

$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \tag{2.7}$$

The quantity $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ in Derivation 2.6.1 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.

2.6.2 Bayesian Solution: Maximum Likelihood Estimation

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:

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

Which can be written equivalently as:

$$p(y_i|\mathbf{x}_i, \mathbf{w}, \beta) = \mathcal{N}(\mathbf{w}^T\mathbf{x}_i, \beta^{-1}) \tag{2.8}$$

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

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

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 $\mathbf{w}$:

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

Using the density function of a univariate Gaussian:

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

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 $\mathbf{w}$:

$$\frac{\partial \ln p(\mathbf{Y}|\mathbf{X}, \mathbf{w}, \beta)}{\partial \mathbf{w}} = \beta \sum_{n=1}^{N}(y_n - \mathbf{w}^T\mathbf{x}_n)\mathbf{x}_n^T$$

$$0 = \beta \sum_{n=1}^{N}(y_n - \mathbf{w}^T\mathbf{x}_n)\mathbf{x}_n^T$$

$$0 = \sum_{n=1}^{N} y_n \mathbf{x}_n^T - \mathbf{w}^T \sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^T$$

Notice that this is exactly the same form as Equation 2.6. Solving for $\mathbf{w}$ as before:

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

Notice that our final solution is exactly the same form as the solution in Equation 2.7, 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**.

2.7 Model Flexibility

Occasionally, linear regression will fail to recover a good solution for a data set. While this may be because our data doesn't actually have predictive power, it might also just indicate that our data is provided in a format unsuitable for linear regression. This section explores that problem, in particular focusing on how we can manipulate the flexibility of our models to make them perform better.

2.7.1 Basis Functions

There are some situations where linear regression is not the best choice of model for our input data $\mathbf{X}$. 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 2.7.1 (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:

$$\phi(\mathbf{x}) = (x^{(1)}, {x^{(1)}}^2, x^{(2)}, \sin(x^{(2)}))'$$

Using a basis function is so common that we will sometimes describe our input data points as $\boldsymbol{\phi} = (\phi^{(1)}, \phi^{(2)}, ..., \phi^{(D)})'$.

⋆ The notation $\mathbf{x} = (x^{(1)}, x^{(2)})'$ is a way to describe the dimensions of a single data point $\mathbf{x}$. The term $\mathbf{x}^{(1)}$ is the first dimension of a data point $\mathbf{x}$, while $\mathbf{x}_1$ is the first data point in a data set.

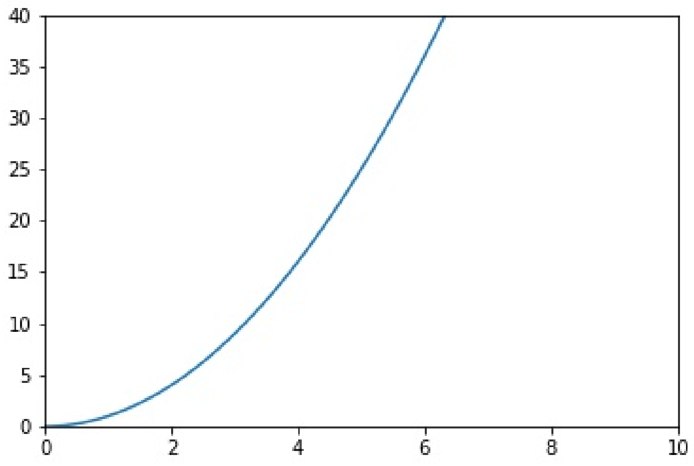

Figure 2.4: Data with no basis function applied.

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 2.4 for linear regression in terms of the basis function applied to our input data:

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

To motivate why we might need basis functions for performing linear regression, let's consider this graph of 1-dimensional inputs **X** along with their target outputs **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. TODO COMBINE THESE IMAGES INTO ONE

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, as the green line demonstrates. TODO ADD GREEN LINE

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, in the chapter on Neural Networks, we will discuss methods for discovering the best basis functions for our data automatically.

2.7.2 Regularization

When we introduced the idea of basis functions above, you might have wondered why we didn't just try adding many basis transformations to our input data to find a good transformation. For

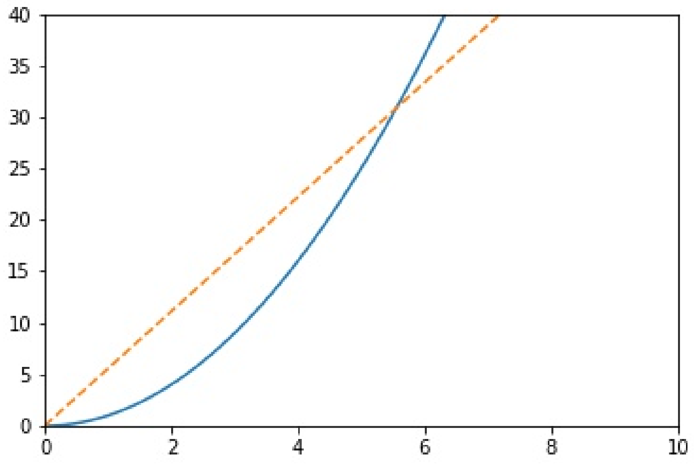

Figure 2.5: Data with no basis function applied, attempt to fit a line.

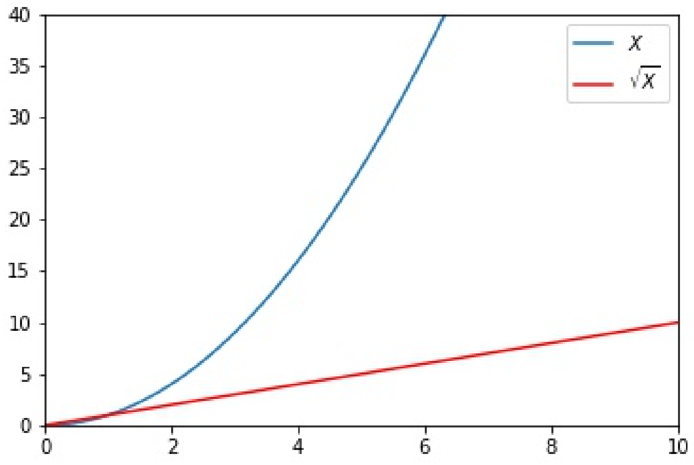

Figure 2.6: Data with square root basis function applied.

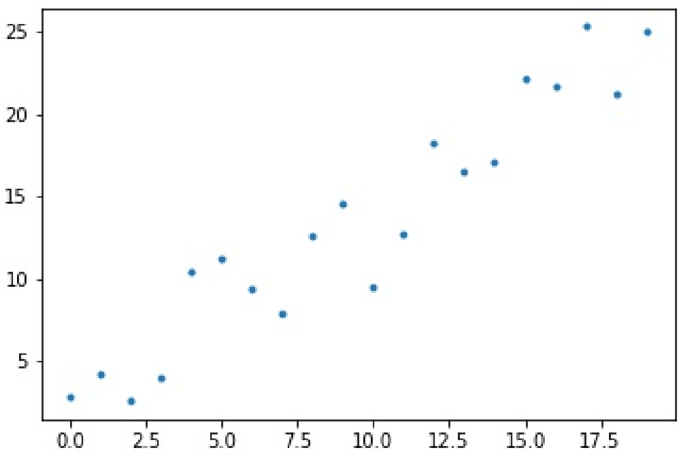

Figure 2.7: Data set with a clear trend and Gaussian noise.

example, we might use this large basis function on a D-dimensional data point $\mathbf{z}$:

$$\phi(\mathbf{z}) = (z^{(1)}, {z^{(1)}}^2, ..., {z^{(1)}}^{100}, z^{(2)}, {z^{(2)}}^2, ..., {z^{(2)}}^{100}, ..., z^{(D)}, {z^{(D)}}^2, ..., {z^{(D)}}^{100})'$$

where you can see that we expand the dimensions of the data point to be 100 times its original size.

Let's say we have an input data point $\mathbf{x}$ that 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 no loss! But is this a desirable outcome? The answer is no, and we'll provide a visual example to illustrate that.

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. TODO move images next to each other

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, we see that right now we're only penalizing predictions that are not correct in training. However, what we ultimately care about is doing well on new data points, not just our training set. This leads us to the idea of **generalization**.

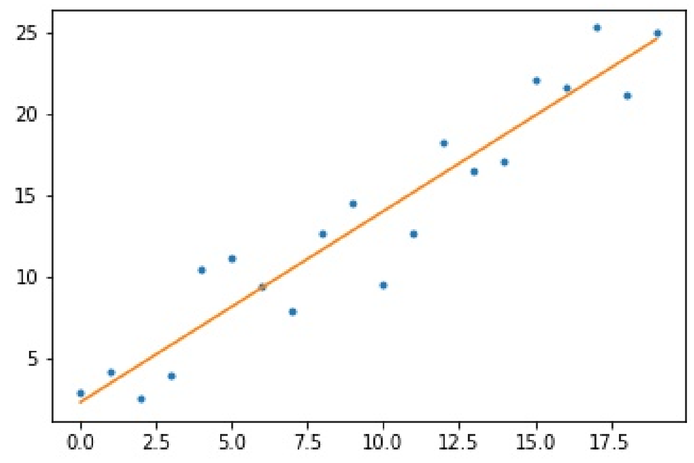

Figure 2.8: Natural fit for this data set.

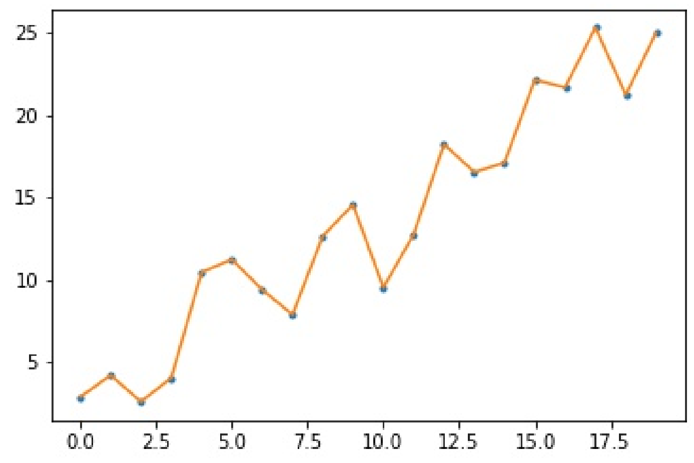

Figure 2.9: Unnatural fit for this data set.

Definition 2.7.2 (Generalization): Generalization is the ability of a model to perform well (said to *generalize to*) new data points outside of the training set.

A convoluted line that matches the noise of our training set exactly isn't going to generalize well to new data points that don't look exactly like those found in our training set. If wish to avoid 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 many complex weight values that produce a messy regression line. 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 an example of what's known as **regularization**, and it's one that we will see come up often in different machine learning methods.

Definition 2.7.3 (Regularization): Applying penalties to parameters of a model.

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:

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

The effect of λ is to penalize large weight parameters. The larger λ is, the more we will favor 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**. While L2 norm regularization is very common, it is just one example of many ways we can perform regularization.

To build some intuition about the effect of this regularization parameter, examine Figure 2.10. Notice how larger values of λ produce less complex lines, which is the result of applying more regularization. This is very nice for the problem we started with - needing 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.

2.7.3 Generalizing Regularization

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

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

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

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(y_n - \mathbf{w}^T\boldsymbol{\phi}_n)^2 + \frac{\lambda}{2}|\mathbf{w}|^h$$

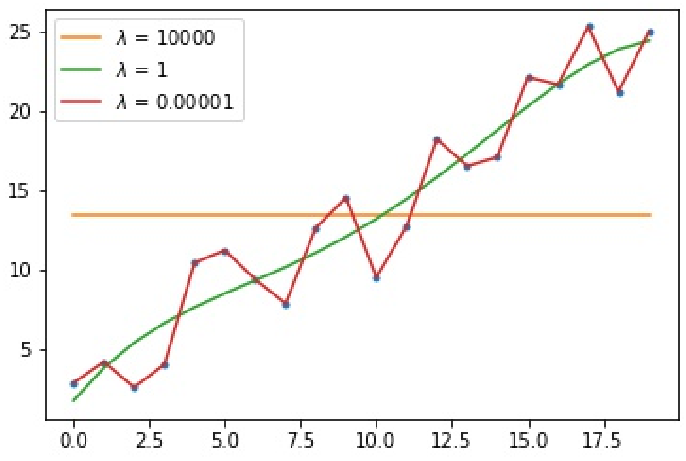

Figure 2.10: Effect of different regularization parameter values on final regression solution.

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 2.11.

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.

2.7.4 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 our weight parameters as adding a prior distribution over $\mathbf{w}$. Note that this is a different conception of regularization than we saw in the previous section. In the Bayesian framework, we are averaging over different models specified by values of $\mathbf{w}$. Therefore, in this context regularization entails weighting models with smaller values

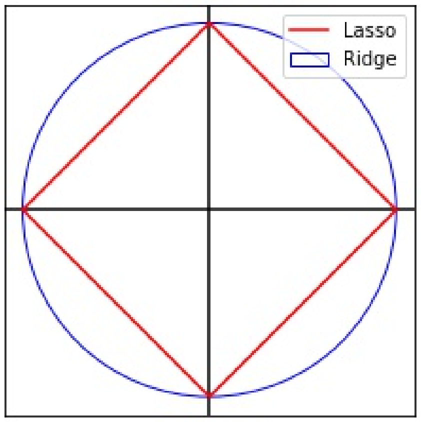

Figure 2.11: Form of the ridge (blue) and lasso (red) regression functions.

of $\mathbf{w}$ more heavily.

Derivation 2.7.1 (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 2.8, 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:

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

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

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

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

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

dividing by a positive constant β:

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

Notice that maximizing the posterior probability is equivalent to minimizing the sum of squared errors $(y_n - \mathbf{w}^T\mathbf{x}_n)^2$ 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 and then maximizing the resulting posterior distribution is equivalent to adding a regularization term where $\lambda = \frac{\boldsymbol{S}_0}{\beta}$

2.8 Choosing Between Models

2.8.1 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 2.8.1 (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 TODO bunch of graphs with different wavy lines for different inputs). 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 our model heavily favors a specific form regardless of the training data, so our target outputs don't fluctuate between distinct training sets.

Obviously a good solution will fall somewhere in between these two extremes of high variance and high bias. Indeed, we have techniques like regularization to help us balance the two extremes (improving generalization), and we have other techniques like *cross-validation* that help us determine when we have found a good balance (measuring generalization).

⋆ 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 mediate between these opposing forces of error in our models, 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 2.8.1 (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 a random variable of $(\mathbf{x}, y)$ pairs sample from a distribution F, 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 $\bar{y}$, which is the true conditional mean given by $\bar{y} = \text{E}_{y|\mathbf{x}}[y]$, 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:

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

And note that:

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

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:

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

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 our prediction mean $\bar{f}(\cdot) = \mathrm{E}_D[f(\mathbf{x})]$, which is the expectation of our model function taken with respect to our random data set.

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

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, since it is the square of the average difference between our prediction and the true target value. The second one, $\mathrm{E}[(\bar{f}(\mathbf{x})-f(\mathbf{x}))^2]$, is the *variance* of our model, since it is the expected squared difference between our model and its average value. Thus:

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

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

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

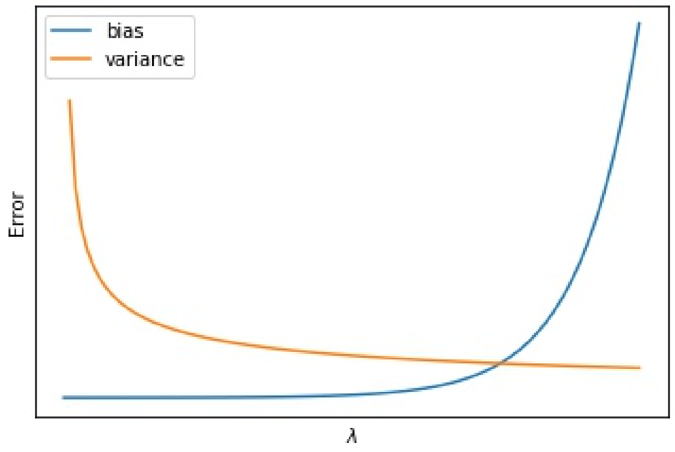

Figure 2.12: Bias and variance both contribute to the overall error of our model.

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 2.12.

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**.

Definition 2.8.2 (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 2.8.3 (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.

2.8.2 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 very general technique for finding the sweet spot of our regularization parameter, other hyperparameters, or even for choosing among entirely different models is known as **cross-validation**. TODO pickup here

Definition 2.8.4 (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 $\mathbf{K}$ 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.

2.8.3 Making a Model Choice

Now that we're aware of overfitting, underfitting, and how those concepts relate to the bias-variance tradeoff, we still need to come back to the question of how we actually select a model. Intuitively, we are trying to find the middle ground between bias and variance: picking a model that fits our data but that is also general enough to perform well on yet unseen data. Furthermore, there is no such thing as the 'right' model choice. Instead, there are only model options that are either better or worse than others. To that end, it can be best to rely on the techniques presented above, specifically cross-validation, to make your model selection. Then, although you will not be able to make any sort of guarantee about your selection being the 'best' of all possible models, you can at least have confidence your model achieved the best generalizability that could be proven through cross-validation.

2.8.4 Bayesian Model Averaging

We can also handle model selection using a Bayesian approach. This means we account for our uncertainty about the true model by averaging over the possible candidate models, weighting each model by our prior certainty that it is the one producing our data. If we have M models indexed by $m = 1, ..., M$, we can write the likelihood of observing our data set $\mathbf{X}$ as follows:

$$p(\mathbf{X}) = \sum_{m=1}^{M} p(\mathbf{X}|m)p(m)$$

where $p(m)$ is our prior certainty for a given model and $p(\mathbf{X}|m)$ is the likelihood of our data set given that model. The elegance of this approach is that we don't have to pick any particular model, instead choosing to marginalize out our uncertainty.

2.9 Linear Regression Extras

With most of linear regression under our belt at this point, it's useful to drill down on a few concepts to come to a deeper understanding of how we can use them in the context of linear regression and beyond. TODO rename this portion now

2.9.1 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 2.7.4, 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 2.9.1 (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

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

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:

$$\begin{aligned} p(y^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) &= \int_{\mathbf{w}} p(y^*|\mathbf{x}^*, \mathbf{w}) p(\mathbf{w}|\mathbf{X}, \mathbf{Y}) d\mathbf{w} \\ &= \mathcal{N}(y^*|\mathbf{w}^T\mathbf{x}^*, \beta^{-1}) \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}_N, \boldsymbol{S}_N^{-1}) d\mathbf{w} \end{aligned} \tag{2.14}$$

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.

2.10 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.

2.11 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.