

Assignment 1: Linear Regression

Machine Learning

Fall 2019

🔗 Learning Objectives

- Gain some familiarity with some of the key ideas in machine learning.
- Review of mathematical concepts we will be using in the beginning part of this course.
- Familiarize yourself with computational tools for machine learning.
- Learn linear regression using a “top-down” approach.

1 Six Big Ideas in Machine Learning

Before diving into the specifics of our first machine learning algorithm, let's examine some important ideas in machine learning. Why six big ideas? If Rob Martello has taught us anything, it's that all good things come in sixes.

Idea 1: Correlations for the Win?

ML algorithms learn to exploit correlations in data in order to make predictions. For instance, if one was using an ML algorithm to recognize whether someone was smiling in an image, the algorithm might learn that bright pixels around the mouth region are correlated with smiles (e.g., they indicate that a person's teeth are showing). This correlation would likely be useful for determining whether a new image of a face was a smile or not. Now suppose you take this model and apply it to a new dataset. You may find that faces that are angry are mistakenly marked as smiling! Why? In the case of angry facial expressions the teeth may also be showing. Of course you would expect the learning algorithm to be smart enough to realize that just using the presence of teeth is not enough to conclusively determine whether someone is smiling. Whether or not this actually happens is a function of the training data given to the ML algorithm. If, for instance, the training set was scraped from profile pictures from a dating website, the training set may not contain pictures of angry faces. Unfortunately, while exploiting correlations is one of the most powerful aspects of ML systems, it is also one of the most potentially problematic.

- *Example 1: Reinforcing Hiring Biases* You may have heard that [Amazon scrapped a secret AI recruiting tool that showed bias against women](#). More specifically, the tool performed automatic keyword analysis of job applications to predict whether or not the applicant was worth forwarding on to a human for further evaluation. Early in the development of this system researchers discovered that the model the system had learned placed a negative weight on words such as “women's” as well as the names of some women's colleges. While there is of course no causal link between these words

appearing in a job application and the suitability of the candidate for the job, there was a *correlation* in the training set between the presence of words such as “women’s” and the candidates not being invited for interviews.

Why might such a correlation exist in the training set? There are many different possible explanations for this correlation, ranging from overt or unconscious bias in the applicant evaluators whose judgments helped form the training data to systemic discrimination that denies women equal access to educational opportunities in STEM. The important thing to take away from this is not *why* there was a correlation, but that the existence of the correlation in the training data caused the model to utilize the correlation in order to evaluate new data. Amazon realized that this was very bad and decided to take steps to address the problem (they say they never used the system to make actual job-screening decisions). Despite efforts to prevent the algorithm from exploiting such correlations, the group determined that they couldn’t fully guarantee that the algorithm had not found another way to achieve the same discriminatory outcome and terminated the project.

- *Example 2: Adversarial Machine Learning*

A second example of an ML algorithm exploiting correlations in training data in unexpected ways can be found in computer vision methods for object detection. Identifying salient visual objects such as road signs and pedestrians is an important building block for applications such as autonomous cars. A popular algorithm for this task, [YOLO](#) (You Only Look Once)¹, can identify and localize objects in images with surprising accuracy. For instance, in the image below YOLO identified the stop sign in the image successfully.

¹ [Cool video of YOLO version 3](#), [TED talk from the head YOLO researcher](#)

While this all seems great, there is a catch. It is very difficult to understand *how* YOLO is making these predictions. That is, what is it about this image that causes the YOLO algorithm to be able to tell that it is a stop sign? Perhaps it is the white text on the red background. Perhaps it is the word “STOP.” In fact, the network that makes this prediction is so complex, that it is impossible for us to say definitively exactly how it makes its decision. What we do know is that the model exploits correlations in the training data between input features (pixels) and outputs (object locations) in potentially unpredictable ways.

The complexity of the model makes it vulnerable to bad actors (or adversaries). Researchers at University of Michigan used a form of ML known as *adversarial machine learning* to [create a specially crafted sticker that could be attached to a stop sign that would make it invisible to the YOLO model](#) (that is YOLO would not identify it as a stop sign). Clearly, this has major implications for the safety of using a model such as this in an application like a self-driving car. An example of the attack is shown in Figure ??.

Idea 2: There’s No Such Thing as a Free Lunch

“[All models are wrong, but some useful.](#)”

— George Box

At the beginning of this document we have a reminder of the basic supervised machine learning setup. A one sentence statement of the setup is that we try to generalize from a set of training data to construct a function \hat{f}^* that best predicts the corresponding output data for unseen input data (e.g., predicting the facial expression of a face that was not in the training set based on a training set of sample faces). In the previous big idea, we discussed how machine learning could go wrong when there are correlations in the data that seem useful to the ML algorithm but are ultimately counterproductive to how we'd like the system to make decisions. It turns out that even before you choose the training data for your algorithm, you must provide an [inductive bias](#) to constrain the space of possible models you might fit. Examples of common inductive biases include the following (the previously linked article has some more).

1. The prediction function \hat{f}^* should change smoothly as you vary the input \mathbf{x} .
2. The prediction function has a particular form (e.g., linear).
3. The prediction function is sparse (it ignores the majority of the inputs).

In fact, there are a whole class of theorems called [No-Free-Lunch \(NFL\) theorems](#) that state that without inductive biases (such as the ones stated above), learning from data is essentially impossible. This connects us back to the quote from George Box. While the inductive bias we encode into our model will never fully represent reality, having this bias is necessary to allow the model to do the useful work of making predictions. What's important for us as machine learning scientists and practitioners is to be explicit about the biases we are introducing when settling on a particular model so that we can best evaluate our results and predict the limitations of our systems.

Idea 3: It's All About How You Frame the Problem

Using Machine Learning algorithms can be a bit disorienting for someone used to the typical engineering workflow. A cartoon picture of the engineering workflow is that you are given a problem (perhaps it is initially difficult to solve or ambiguous), you might reframe the problem to make it easier to solve, and then you work to devise a solution to the reframed problem. In machine learning, the last step is replaced by providing examples of how you'd like your system to work (i.e., input / output pairs), and then the creation of the actual system is automated by the ML algorithm! Your job as an ML practitioner is to reframe the original problem (both by specifying the form of the model and giving appropriate training data) so that the ML algorithm can compute a solution. If you've done the reframing properly, the solution to the reframed problem will also be a good solution to the original problem.

As an example of when a solution to the reframed problem would not be desirable, consider the use of a machine learning algorithm to teach a virtual character to walk in a simulated environment. You might reframe this problem for the ML algorithm as tasking it with computing a controller for the virtual character that moves the character's center of mass forward as fast as possible. The ML algorithm can now search over a vast space of possible control strategies to learn the one that most quickly propels the

center of mass. However, it doesn't necessarily follow that this controller will result in the character walking using a normal bipedal gait.

The notion that the solution an algorithm finds might be unpredictable to the designer is known as "emergence." Some cool examples of this played out in actual experiments in evolving virtual creatures, which are summarized in the paper [The Surprising Creativity of Digital Evolution](#). For instance, a virtual character learned that falling down, see picture above, and getting up was more efficient for locomotion than constantly hopping (which is what the designer had intended the system to learn).

For more examples of this sort of thing, consider checking out [Karl Sims: Evolved Virtual Creatures](#) or the short article [When AI Surprises Us](#). This also connects back to the age-old debate over whether [falling with style can be considered flying](#).

Idea 4: ML Systems Can Learn Intermediate Representations

In the next few weeks we'll learn about artificial neural networks (ANNs). ANNs are biologically inspired algorithms since their functioning, at an abstract level, is modeled on the functioning of biological neurons (e.g., in the brain).

ANNs accept input patterns at an array of virtual neurons called the input layer (see Figure ??). The neurons in the input layer are connected to other neurons via virtual [axons](#) that control to what extent a particular input neuron activates a downstream neuron. The second set of neurons, called the "hidden layer" (shown in blue in the middle of the figure), is responsible for computing intermediate, hidden representations of the input data. This process continues as activations propagate through the network until activations are generated at the output layer (shown in green on the right of the figure). These outputs could correspond to any salient properties of the input (e.g., if the input is an image, the output might encode the objects in the image).

What's amazing about ANNs is that there are learning algorithms for setting the connection strengths between these virtual neurons (the black arrows in Figure ??) based on training data (input / output pairs). These learning algorithms tune the connections strengths (also called "weights") such that for the provided training data the network produces the appropriate training outputs (e.g., if you show the network a training set of images of cats or dogs, over time the network will adjust its weights so that the output is "cat" when the network is presented an image of a cat and "dog" if presented an image of a dog). The algorithms used to tune the network weights are only concerned with reproducing the output patterns, the network is free to choose how it represents information within the network (i.e., at the hidden layer).

What's super amazing is that we can actually examine the internal representations of a neural network to understand how it's performing the computation from input to output. For instance, Figure ??² shows a visualization of the internal representations learned by a network trained to best compress a training set of images (these sorts of networks are called "auto-encoders"). The receptive fields of each of the hidden units in the network and can be understood as specifying how each input pixel activates a particular hidden unit (gray corresponds to no activation, black to negative activation, and white to positive activation). It's remarkable that these receptive fields have coherent

² From [Sparse coding of sensory inputs](#)

structure: they are localized in space, tuned to particular orientations, and tuned to features at a particular scale. You can think of these as oriented edge detectors that the network learned completely on its own (it was never told to extract edges from the images in the training set).

What's super-duper amazing is that if we compare the receptive fields learned by the artificial neural network to the [simple cells](#) in the primary visual cortex of a cat, there are a number of striking similarities. Just as in the ANN, the biological neural network responds to edges at particular orientations and scales. The scientists Hubel and Wiesel performed the pioneering work in neuroscience to establish the properties of receptive fields in the primary visual cortex. Consider watching [a video of their experiment](#) that eventually garnered a Nobel prize (note that in the video the static sound corresponds to the measurement of spikes in activity of an individual neuron in the brain of an anesthetized cat).³ The implication of the similarity between the receptive fields of the neurons in the cat brain and the virtual neurons in the ANN is that they are similar because they are fundamentally solving the same problem (i.e., efficiently representing visual information). In this light, that they should find similar solutions to this problem is not as surprising as it may first seem.

³ There are a variety of opinions on the [ethics of performing research on animals](#)

Idea 5: Machine Learning Zoomed Out

Historically, most ML courses have been laser-focused on learning about learning algorithms (e.g., neural networks, support vector machines, decision trees, etc.). In some courses there would be a little bit of emphasis on machine learning applications, which have always been strongly tied to the research in ML algorithms and theory. The focus on ML algorithms also reflected the positioning of these courses within Computer Science curricula, which approached the field more from a liberal arts perspective rather than an engineering one.

A number of recent trends have made the almost sole focus on learning algorithms insufficient for those who want to either use ML in their careers or go into ML as a field.

1. The explosion of data has made the skills necessary for collecting, wrangling, exploring, and cleaning data very relevant.
2. Improvements in the accuracy of ML algorithms coupled with the ability to deploy ML systems to a wide variety of devices (e.g., mobile phones) means that it is increasingly important to consider how ML systems will behave in real-world, highly complex settings.

The first point ties into a set of skills sometimes grouped under "Data Science." While we will have a comparatively lesser focus on this skillset than in our dedicated Data Science course, we will be learning some of these skills. The second point corresponds to ML systems as embedded in larger and more complex contexts. As you've seen from some of the examples earlier in this document, unexpected things can happen when ML algorithms meet messy and/or biased real world data (take for example the automated job applicant evaluator). In light of this, again, we think that the traditional focus on ML

algorithms is not adequate for a modern class on ML. Here are two figures to further illustrate this point.

In the figure above, the box labeled *ML Code* is the actual learning algorithm. But in modern systems, this is but a small fraction of all of the tools needed to deploy a real world ML system. This is not to say that we will be spending a lot of time learning about each of these other boxes (we will learn about some of them), but it helps to have a sense of the software ecosystem in which your ML model would be deployed.

In addition to understanding how ML code is situated within larger software ecosystems, it is even more important to realize the [socio-technical context](#) in which an ML system is deployed. The figure above shows a socio-technical analysis of a technology. The figure highlights the need to consider contextual factors such as user impacts, culture, and regulations when analyzing technologies.

Using the tools of socio-technical systems analysis is becoming increasingly popular for analyzing machine learning systems. We'll be digging into some of these resources later in the course, but here are two papers in this spirit.

- [Reframing AI Discourse](#)
- [Fairness and Abstraction in Sociotechnical Systems](#)

Idea 6: It's Not All Doom and Gloom


While we'll be talking a lot about how ML can go wrong, unleashing unexpected consequences, we'll also be talking about the positive things that ML can do. Here are just a couple of resources that discuss such systems (not to say that these systems don't have the potential for things to go wrong!). We'll leave this list deliberately short to give you a chance to find your own example in the exercise below.

- [AI for social good: 7 inspiring examples](#)
- [Machine Learning for Web Accessibility](#)
- [19 Times Data Analysis Empowered Students and Schools](#)
- This one is kind of cheating. Austin Vesiliza put together [a list of links to AI for social good projects](#) that you might use for inspiration.

Exercise 1 (60 minutes)

Now, we want to hear from you!

- (a) Choose one of the big ideas above and write a short response to it. Your response could incorporate something surprising you read, a thought-provoking question, your personal experience, an additional resource that builds upon or shifts the discussion. We hope that this reflection will help scaffold class discussions and get you thinking about your interests in the big space that is ML. Also, you have license from us to customize the structure of your response as you see fit. As a rough guide, you should aim for a response of a 1-2 para-

The  icon means that you will be submitting your response to this exercise with your assignment. By default responses to exercises do not need to be submitted.

graphs.

☆ Solution

There's no one right answer here!

- (b) Idea 6 talks about the idea of ML for positive impact. What is one example of an ML application (real or imagined) that you think would have the largest (or most unambiguously) positive impact on the world? Why? Alternatively, what is an example of an ML application (real or imagined) that no matter how carefully the designers approach it, should just not exist due to the harm it would cause the world? Why?

☆ Solution

There's no one right answer here!

2 Mathematical Background

⚠ Notice

For the purposes of this class, we will be consistent with the notation we use. Of course, when we link to other resources, they may use other notation. If notation is different in a way that causes confusion, we will try to point out pitfalls you should watch out for. Please use this link to access our guide to [our notation conventions](#).

🔗 External Resource(s) (60 minutes)

In order to engage with this assignment, you'll want to make sure you are familiar with the following concepts (links to resources embedded below):

- Vector-vector multiplication
 - Section 2.1 of [Zico Kolter's Linear Algebra Review and Reference](#)
- Matrix-vector multiplication
 - Section 2.2 of [Zico Kolter's Linear Algebra Review and Reference](#)
 - The first bits of the Khan academy video on [Linear Transformations](#)
- Partial derivatives and gradients
 - Khan Academy videos on partial derivatives: [intro](#), [graphical understanding](#), and [formal definition](#)

– [Khan Academy video on Gradient](#)

Exercise 2 (20 minutes)

Here are some diagnostic problems to test some basic understanding of the concepts above.

- (a) Suppose $f(x, y) = 2x \sin y + y^2 x^3$. Calculate $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial y}$, and ∇f .

☆ Solution

$$\frac{\partial f}{\partial x} = 2 \sin y + 3y^2 x^2 \quad (1)$$

$$\frac{\partial f}{\partial y} = 2x \cos y + 2yx^3 \quad (2)$$

$$\nabla f = \begin{bmatrix} 2 \sin y + 3y^2 x^2 \\ 2x \cos y + 2yx^3 \end{bmatrix} \quad (3)$$

- (b) Suppose $\mathbf{x} = \begin{bmatrix} 3 \\ -1 \\ 4 \end{bmatrix}$ and $\mathbf{y} = \begin{bmatrix} 2 \\ 7 \\ 4 \end{bmatrix}$. Calculate $\mathbf{x} \cdot \mathbf{y}$, $\mathbf{x}^\top \mathbf{y}$, and \mathbf{xy}^\top .

☆ Solution

$$\mathbf{x} \cdot \mathbf{y} = 3 \times 2 + (-1) \times 7 + 4 \times 4 = 15 \quad (4)$$

$$\mathbf{x}^\top \mathbf{y} = \mathbf{x} \cdot \mathbf{y} = 15 \quad (5)$$

$$\mathbf{xy}^\top = \begin{bmatrix} 3 \times 2 & 3 \times 7 & 3 \times 4 \\ -1 \times 2 & -1 \times 7 & -1 \times 4 \\ 4 \times 2 & 4 \times 7 & 4 \times 4 \end{bmatrix} \quad (6)$$

$$= \begin{bmatrix} 6 & 21 & 12 \\ -2 & -7 & -4 \\ 8 & 28 & 16 \end{bmatrix} \quad (7)$$

- (c) Let $\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \dots & \mathbf{a}_n \end{bmatrix} = \begin{bmatrix} \mathbf{row}_1^\top \\ \mathbf{row}_2^\top \\ \vdots \\ \mathbf{row}_m^\top \end{bmatrix}$

(that is, the matrix \mathbf{A} can either be thought of as consisting of the columns $\mathbf{a}_1, \dots, \mathbf{a}_n$ or the rows $\mathbf{row}_1^\top, \dots, \mathbf{row}_m^\top$). Let \mathbf{v} be an arbitrary n -dimensional vector.

Compute \mathbf{Av} in terms of $\mathbf{a}_1, \dots, \mathbf{a}_n$.

☆ Solution

$$\mathbf{A}\mathbf{v} = \mathbf{v}_1\mathbf{a}_1 + \mathbf{v}_2\mathbf{a}_2 + \dots + \mathbf{v}_n\mathbf{a}_n \quad (8)$$

Compute $\mathbf{A}\mathbf{v}$ in terms of the rows of $\mathbf{row}_1, \dots, \mathbf{row}_m$.

☆ Solution

$$\mathbf{A}\mathbf{v} = \begin{bmatrix} \mathbf{v} \cdot \mathbf{row}_1 \\ \mathbf{v} \cdot \mathbf{row}_2 \\ \vdots \\ \mathbf{v} \cdot \mathbf{row}_m \end{bmatrix} \quad (9)$$

3 Supervised Learning Problem Setup

Suppose you are given a training set of data points, $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)$ where each \mathbf{x}_i represents an element of an input space (e.g., a d -dimensional feature vector) and each y_i represents an element of an output space (e.g., a scalar target value). In the supervised learning setting, your goal is to determine a function \hat{f} that maps from the input space to the output space. For example, if we provide an input \mathbf{x} to \hat{f} it would generate the predicted output $\hat{y} = \hat{f}(\mathbf{x})$.

We typically also assume that there is some loss function, ℓ , that determines the amount of loss that a particular prediction \hat{y}_i incurs due to a mismatch with the actual output y_i . We can define the best possible model, \hat{f}^* as the one that minimizes these losses over the training set. This notion can be expressed with the following equation (note: that $\arg \min$ in the equation below just means the value that minimizes the expression inside of the $\arg \min$, e.g., $\arg \min_x (x - 2)^2 = 2$, whereas $\min_x (x - 2)^2 = 0$).

$$\hat{f}^* = \arg \min_f \sum_{i=1}^n \ell(\hat{f}(\mathbf{x}_i), y_i) \quad (10)$$

4 Linear Regression from the Top-Down

Motivation: Why Learn About Linear Regression?

Before we jump into the *what* of linear regression, let's spend a little bit of time talking about the *why* of linear regression. As you'll soon see, linear regression is among the simplest (perhaps *the* simplest) machine learning algorithm. It has many limitations, which you'll also see, but also a of ton strengths. **First, it is a great place to start when learning about machine learning** since the algorithm can be understood and implemented using a relatively small number of mathematical ideas (you'll be reviewing

these ideas later in this assignment). In terms of the algorithm itself, it has the following very nice properties.

- **Transparent:** it's pretty easy to examine the model and understand how it arrives at its predictions.
- **Computational tractable:** models can be trained efficiently on datasets with large numbers of features and data points.
- **Easy to implement:** linear regression can be implemented using a number of different algorithms (e.g., gradient descent, closed-form solution). Even if the algorithm is not built into your favorite numerical computation library, the algorithm can be implemented in only a couple of lines of code.

For linear regression our input data, \mathbf{x}_i , are d -dimensional vectors (each entry of these vectors can be thought of as a feature), our output data, y_i , are scalars, and our prediction functions, \hat{f} , are all of the form $\hat{f}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} = \mathbf{w}^\top \mathbf{x} = \sum_{i=1}^d w_i x_i$ for some vector of weights \mathbf{w} (you could think of \hat{f} as also taking \mathbf{w} as an input, e.g., writing $\hat{f}(\mathbf{x}, \mathbf{w})$. Most of the time we'll leave \mathbf{w} as an implicit input: writing $\hat{f}(\mathbf{x})$).

In the function, \hat{f} , the elements of the vector \mathbf{w} represent weights that multiply various dimensions of the input. For instance, if an element of \mathbf{w} is high, that means that as the corresponding element of \mathbf{x} increases, the prediction that \hat{f} generates would also increase (you may want to mentally think through other cases, e.g., what would happen if the element of \mathbf{x} decreases, or what would happen if the entry of \mathbf{w} was large and negative). The products of the weights and the features are then summed to arrive at an overall prediction.

Given this model, we can now define our very first machine learning algorithm: **ordinary least squares** (OLS)! In the ordinary least squares algorithm, we use our training set to select the \mathbf{w} that minimizes the sum of squared differences between the model's predictions and the training outputs. Thinking back to the supervised learning problem setup, this corresponds to choosing $\ell(y, \hat{y}) = (y - \hat{y})^2$. Therefore, the OLS algorithm will use the training data to select the optimal value of \mathbf{w} (called \mathbf{w}^*), which minimizes the sum of squared differences between the model's predictions and the training outputs.

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \sum_{i=1}^n \ell(\hat{f}(\mathbf{x}_i, \mathbf{w}), y_i) \quad (11)$$

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \sum_{i=1}^n (\hat{f}(\mathbf{x}_i, \mathbf{w}) - y_i)^2 \quad (12)$$

$$= \arg \min_{\mathbf{w}} \sum_{i=1}^n (\mathbf{w}^\top \mathbf{x}_i - y_i)^2 \quad (13)$$

⚠ Notice

Digesting mathematical equations like this can be daunting, but your understanding will be increased by unpacking them carefully. Make sure you understand what was substituted and why in each of these lines. Make sure you understand what each symbol represents. If you are confused, ask for help (e.g., post on NB).

While we haven't talked at all about how to find \mathbf{w}^* , that will be the focus of the next assignment, once we have \mathbf{w}^* we can predict a value for a new input point, \mathbf{x} , by predicting the corresponding (unknown) output, y , as $\hat{y} = \mathbf{w}^* \cdot \mathbf{x}$. In this way, we have used the training data to learn how to make predictions about unseen data, which is the hallmark of supervised machine learning!

Exercise 3

Draw a scatter plot in 2D (the x-axis is the independent variable and the y-axis is the dependent variable). Fill in a few data points and draw a potential line of best fit. On the plot identify the vertical differences between the data points and the line (these differences are the residuals). Draw a second potential line of best fit and mark the residuals. From the point of view of ordinary least-squares, which of these lines is better?

☆ Solution

The red line (line 1) would be better since the residuals are generally smaller. Line 2 also has several large residuals, which when squared will cause a large penalty for line 2.

5 *Getting a Feel for Linear Regression*

In this class we'll be learning about algorithms using both a top-down and a bottom-up approach. By bottom-up we mean applying various mathematical rules to derive a solution to a problem and only then trying to understand how to apply it and how it well it might work for various problems. By top-down we mean starting by applying the algorithm to various problems and through these applications gaining a sense of the algorithm's properties. We'll start our investigation of linear regression using a **top-down approach**.

5.1 *Linear Regression with One Input Variable: Line of Best Fit*

If any of what we've said so far sounds familiar, it is likely because you have seen the idea of a line of best fit in some previous class. To understand more intuitively what the OLS algorithm is doing, we want you to investigate its behavior when there is a single input variable (i.e., you are computing a line of best fit).

🔗 External Resource(s)

Use the [line of best fit online app](#) to create some datasets, guess the line of best fit, and then compare the results to the OLS solution (line of best fit).

Exercise 4

- (a) Examine the role that outliers play in determining the line of best fit. Does OLS seem sensitive or insensitive to the presence of outliers in the data?

☆ Solution

OLS is very sensitive to outliers. A single outlier can change the slope of the line of best fit dramatically. Here is an example of this phenomenon.

- (b) Were there any times when the line of best fit didn't seem to really be "best" (e.g., it didn't seem to capture the trends in the data)?

☆ Solution

This could happen for many reasons. If the dataset is piecewise linear (e.g., composed of multiple line segments), if it has some other non-linear form (e.g., if it is quadratic), or if there are outliers.

5.2 Linear Regression with Multiple Input Variables: Explorations in Python

🔗 External Resource(s) (90 minutes)

Work through the [Assignment 1 Companion Notebook](#) to get some practice with numpy and explore linear regression using a top-down approach. You can place your answers directly in the Jupyter notebook so that you have them for your records.