4.8 Application: Linear Regression

In the past few sections, we have learned about one application of mathematical proof to computer science: arguing why a given function implementation is *correct*, i.e., satisfies the function's specification for all allowed inputs. Our examples of `sum_to_n`, `divides`, and `is_prime` have all had one feature in common: we started with a simple, naive translation of a mathematical definition/specification, and then saw how to use theorems to justify the correctness of an alternate (and faster) implementation. However, there are plenty of complex problems that do not have an "obvious" solution obtained by translating a mathematical definition or expression. We'll study one such problem—encrypting and decrypting data—in detail later on in the course, but we wanted to end this chapter on a brief introduction to another such problem: approximating relationships between two quantitative variables.

What is linear regression?

One of the fundamental kinds of study performed in the sciences and social sciences is to identify and analyze patterns in data, with the goal of finding relationships between different quantities. For example, a climate scientist might collect data from around the world to investigate the relationship between carbon dioxide levels and temperature; an economist might investigate the relationship between education and employment in various industries; a health researcher might investigate the levels of various molecular compounds present in the human body.

The particular kind of investigation we'll look at in this section is when the data collected involv two numeric variables which by convention we'll call x and y.[1] Formally, we can represent our dataset as a *set of* (x, y) *tuples*: we let $n \in \mathbb{Z}^+$, and define our dataset to be $S = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$.

[1] We'll keep things fairly abstract for this section, but you may find it helpful to think concretely using the example of x representing carbon dioxide levels and y representing temperatures.

The goal of a **regression analysis** of this dataset is to find a mathematical model to represent the relationship between our x and y variables, and to evaluate how accurate this model is. As you might imagine, for any given pair of variables there could be all sorts of mathematical relationships between them, and have probably heard terms like "linear", "exponential", and "sinusoidal" used to describe these relationships.

The simplest form of regression analysis is **simple linear regression**, in which the goal is to describe the relationship between two variables as a linear function: $y = a + bx$, where a and b are some real numbers.[2] But of course, the question is: given a dataset S, how do evaluate which constants a and b give "good" linear models of the data, and given this evaluation, how to we find the *best* a and b values for our dataset?

[2] You may have also learned of the term "line of best fit": this refers to the linear function that results from performing a linear regression, using the calculations we'll describe in this section.

Before answering these questions, let's reframe them in the language of Python to check our understanding. Our goal is to implement the following function:

```python
def simple_linear_regression(data: list[tuple[float, float]]) -> tuple[float, float]:
    """Return a tuple (a, b) that defines the best linear relationship
    between the two variables given in the dataset.
    """
```

You probably have lots of questions after reading that function specification—good! Let's keep going.

Residuals

No mathematical model of a dataset can perfectly match that dataset perfectly. However, we can evaluate how closely a model matches the dataset by looking at how close each data point is to the model. We can make this precise using the following definition.

Definition. Let $n \in \mathbb{Z}^+$, $S = \{(x_1, y_1), \dots, (x_n, y_n)\}$ be a two-variable dataset, and $y = f(x)$ be a model for the dataset (where $f : \mathbb{R} \to \mathbb{R}$).

We define the **residual of the model for** (x_i, y_i) to be $y_i - f(x_i)$, i.e., the difference between y_i, the actual observed value, and $f(x_i)$, the value that the model predicts given x_i.

For example, if we have a linear model $y = a + bx_i$, then the residual of the model for (x_i, y_i) is the expression $y_i - a - bx_i$.

Now having defined these residuals, we can define precisely how we'll evaluate our models, with the intuition that "the smaller the residuals, the better".

Definition. Let $n \in \mathbb{Z}^+$, $S = \{(x_1, y_1), \dots, (x_n, y_n)\}$ be a two-variable dataset, and $y = f(x)$ be a model for the dataset (where $f : \mathbb{R} \to \mathbb{R}$).

We define the **sum of squares error** of the model to equal the sum of the squares of the residuals for each data point: $\sum_{i=1}^{n} \left(y_i - f(x_i)\right)^2$. So for example, if f is a linear model $y = a + bx$, then the sum of squares error is $\sum_{i=1}^{n} \left(y_i - a - bx_i\right)^2$.

Let's take a pause here, because the definitions of residual and sum of squares error are both easily translated into Python functions. We'll do that here for practice:

```python
def calculate_residual(observation: tuple[float, float], a: float, b: float) -> float:
    """Return the residual for the given observation (x_i, y_i) under the linear model y = a +
    """
    x_i = observation[0]
    y_i = observation[1]
    return y_i - (a + b * x_i)


def calculate_sos_error(data: list[tuple[float, float]], a: float, b: float) -> float:
    """Return the sum of squares error for the given data and the linear model y = a + b * x.
    """
    return sum([calculate_residual(observation, a, b) for observation in data])
```

Computing the "line of best fit"

With these definitions in place, we are ready to answer the first question we posed above: we evaluate the "goodness" of a model by calculating its sum of squares error. Now we can turn to our second question: given a dataset, how do we find the *best* linear model for the data, i.e., the linear model that minimizes the sum of squares error, which people refer to as **the line of best fit**.

We can formulate this question mathematically as follows: find the values of $a, b \in \mathbb{R}$ that minimize the value of $\sum_{i=1}^{n} \left(y_i - a - bx_i\right)^2$.

But even though this statement is straightforward to understand, we cannot do a direct translation into Python code! Not only are there infinite possibilities for a and b, even if we try restricting ourselves to a possible range (like we did for `divides` and `is_prime`), there will *still* be an infinite number of possible values because we're dealing with real numbers rather than integers.

Mathematics to the rescue! It turns out that not only is there indeed a way of calculating this line of best fit, but also there are explicit formulas that do these calculations. We express this in the following theorem.

Theorem. (*Simple linear regression solution*)

Let $n \in \mathbb{Z}^+$, $S = \{(x_1, y_1), \dots, (x_n, y_n)\}$ be a two-variable dataset. Let $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ be the averages of the x and y values in the dataset, respectively.

The sum of squares error for linear models $y = a + bx$ on this dataset is minimized for the following values of a and b:

$$b = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$
$$a = \bar{y} - b \times \bar{x}$$

Proving this theorem is outside the scope of these course notes,[3] but *using* this theorem is quite straightforward, since these formulas can be translated into Python code. With this theorem in hand, we can be confident in an implementation of our `linear_regression` function that simply translates the formulas (using some local variables to break down the calculations!).

[3] The proof itself involves some techniques from calculus, and is not too bad.

```python
def simple_linear_regression(data: list[tuple[float, float]]) -> tuple[float, float]:
    """Return a tuple (a, b) that defines the best linear relationship between the two variable

    This function returns a pair of floats (a, b) such that the line y = a + bx
    is the linear model that minimizes the sum of squares error for this dataset---
    i.e., that minimizes calculate_sos_error(data, a, b).
    """
    x_mean = average([p[0] for p in data])
    y_mean = average([p[1] for p in data])

    b_numerator = sum((p[0] - x_mean) * (p[1] - y_mean) for p in data)
    b_denominator = sum((p[0] - x_mean) * (p[0] - x_mean) for p in data)
    b = b_numerator / b_denominator
    a = y_mean - b * x_mean

    return (a, b)
```

Pretty neat! We started with a very open-ended problem—finding the "best" linear model for a set of data—and applied some mathematical techniques to write a function that computes this line. While this function is a bit longer than the other functions we've written in this section, it is conceptually pretty simple, relying on arithmetic calculations. But the reason these arithmetic calculations actually lead to a "line of best fit", i.e., the reason this function is *correct*, is once again rooted in mathematics.