Monday: Linear Regression

Agenda

5 min: Overview

120 min: Suggested Readings

120 min: Exercises

Specific Learning Outcomes

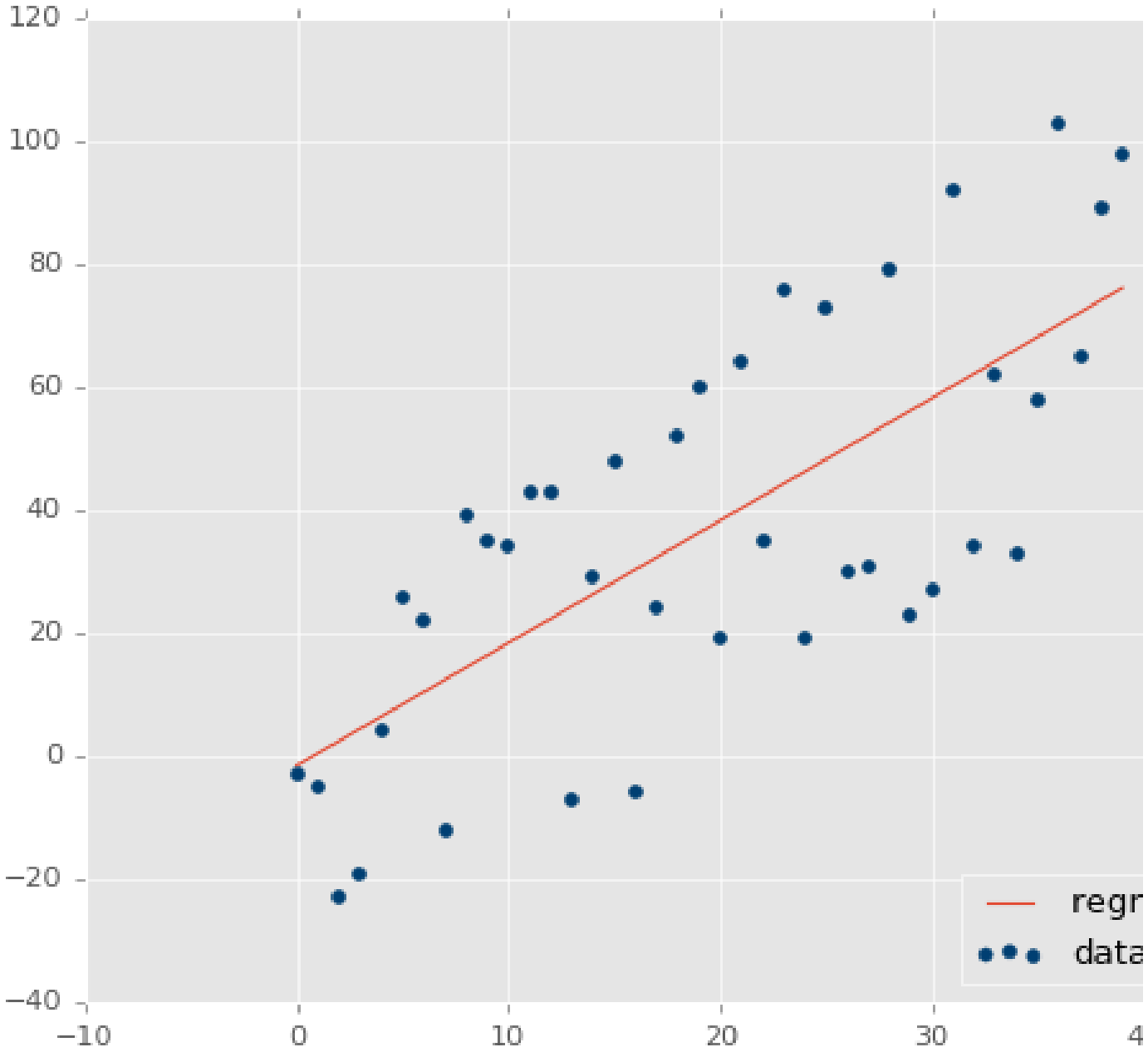
- I can understand when to use regressions and the pre-requisites for linear regressions in particular.
- I can perform simple linear regressions, as well as multivariate linear regressions.
- I can develop an intuition for gradient descent and are aware of the most common versions of gradient descent algorithms.
- I can understand the concept of the learning rate and the challenges of rates that are too high / too low.

Main Resources

Linear regression allows us to predict some dependent variable - *y* - based on independent variable *x* by finding an optimal linear relationship between them. Remember from high school the formula for a line equation:

y = *m*x + *b*

Linear regression algorithms will seek to find optimal values for ***m*** (the slope of the line)and ***b***(The intercept, also known as the bias). This line equation should then allow us to compute a value for y, our dependent variable, so long as we have some input x.



A few things to note when using linear regression:

- This is best for predicting **continuous** variables: Something like price, duration, etc. We will unveil other algorithms later that deal with discrete variables.
- As you can probably tell from the graph above, linear regression models can be heavily biased by a

few **outliers** in the data. Your data cleaning will be critical to get the most out of linear regression.

Multiple independent variables?

We will typically leverage **multiple** independent variables to make our predictions, but the logic remains. instead of the simple line equation above, if we have **n independent variables,** we will end up with an equation that looks like:

$$y = b + m_1 x_1 + m_2 x_2 + m_3 x_3 + ... + m_n x_n$$

Mathematically this is sound: With only one independent variable, this is the equation of a line, with 2, this becomes the equation of a 2d plane in 3 dimensions, etc.

A linear regression with multiple variables is called **a multivariate linear regression.** One particularly interesting result in these scenarios is studying the various coefficients. Effectively in an optimized model, what coefficient m_1 tells us is that if all **other independent variables** remain constant**, increasing x_1 by 1 will increase y by m_1.** This can help you get a sense of how much each independent variable "contributes" to the value of the dependent variable.

Where is the machine learning in this?

How do we come up with an optimal line equation for our model? what is the best value for **m** and **b?** A first step to optimizing **m** and **b** is to create an **error function** (often

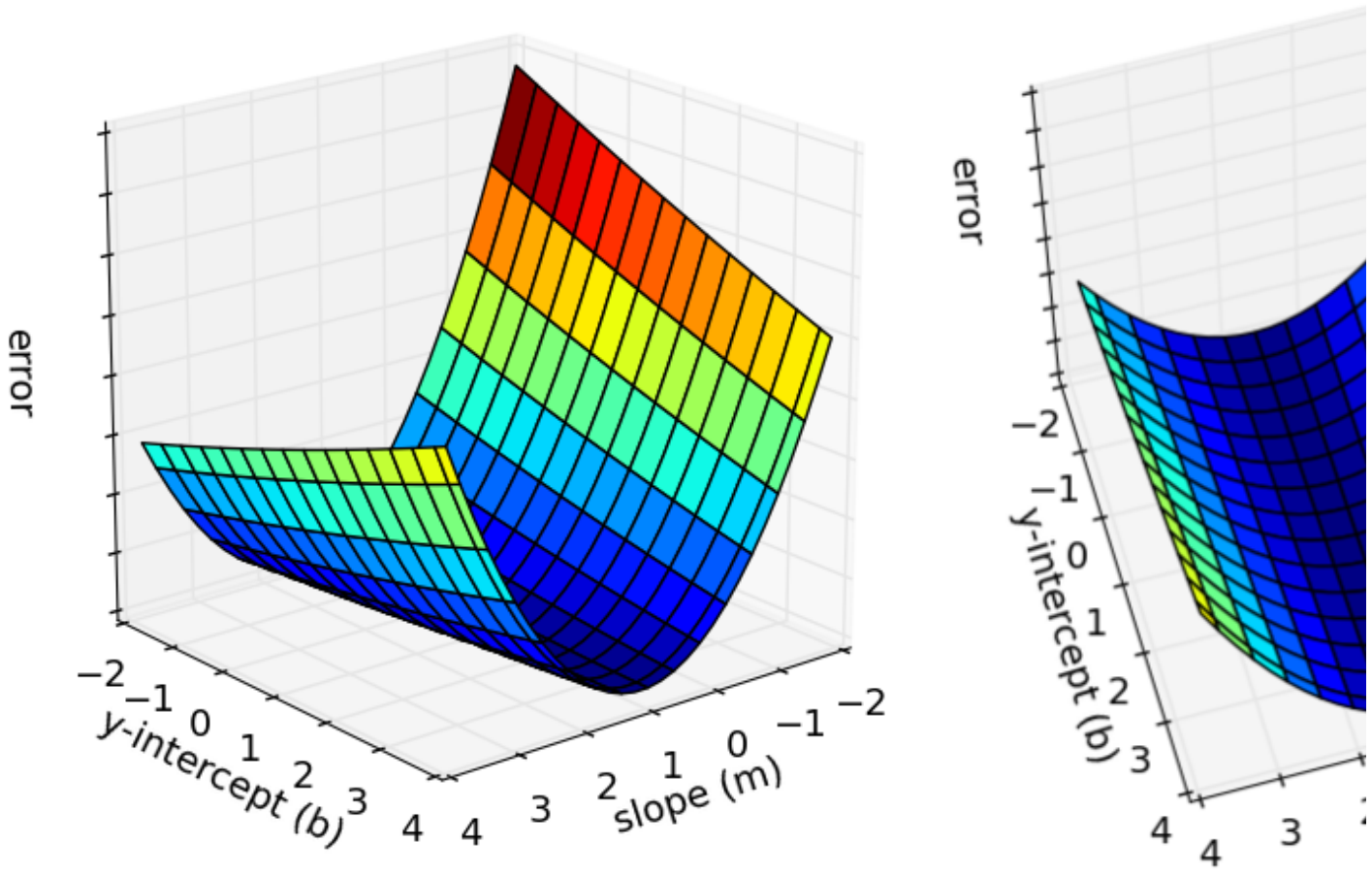
referred to as cost function as well) to assess how good our line y = mx + b is.

If you look at the graph above, our regression line does not pass through most of the points, suggesting it's fairly error-prone. To compute the error function we can perform the following steps on each data point in our data set

1. Compute the difference between your data for the dependent variable, and the prediction from the regression line
2. Square that value, we want all the differences to be positive
3. Add up all the values you've obtained
4. Divide by the number of data points.

In summary, your error function is the average value of $\mathbf{(y_i - (mx_i + b))^2}$ for all your data points. We want to tweak the values of **m** and **b** in order to minimize this function.

This is where the **gradient descent algorithm** comes into play. Imagine plotting the possible values of **m** and **b** against the error function's value. You'd end up with a shape like this.



In the example above, the shape takes the form of a valley of sorts, we basically want to find the point at the bottom of that valley.

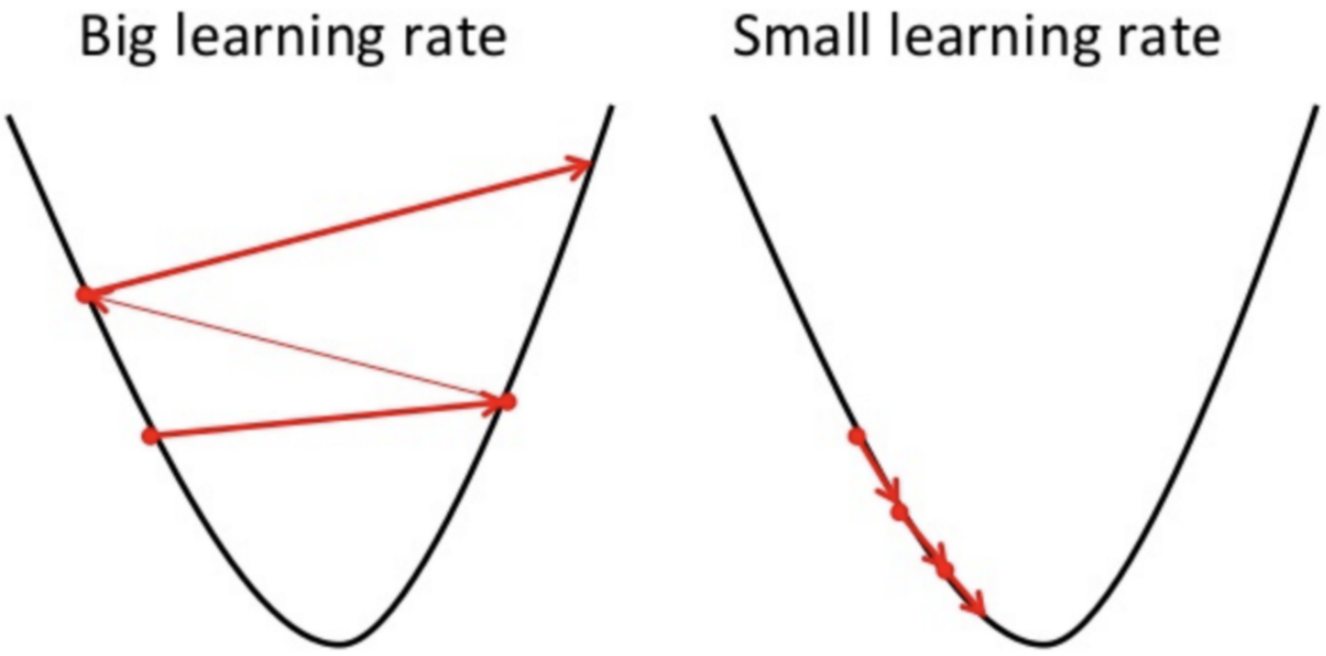
Imagine if you were blindfolded on a hill, and asked to come down of it. How would you approach that challenge? Well, you could feel the ground for the **slope** of the hill, and get a sense for where it's sloping downwards, then follow that route.

This is effectively what gradient descent algorithms do. They pick an initial value for **m** and **b,** then compute the error. From then on, they compute the **slope (or gradient)** at that point,

then move towards new values for **m** and **b** that are "downhill" from where they started.

How much **m** and **b** change from an iteration to the next is based on **the learning rate.** This is a very important parameter, which is worth considering carefully:

- A low learning rate will lead to a very slow model to train. You will be waiting for a long time for the optimal regression line.
- A high learning rate risks missing the optimal point altogether, as exemplified in the picture below.



Gradient descent is a crucial algorithm to be familiar with. Make sure to take time on it today. We have found this **link (https://developers.google.com/machine-learning/crash-**

course/fitter/graph) particularly useful in building up intuition for it.

The readings below will show you implementations of the algorithm, as well as define the most common variants of gradient descent:

- **Batch gradient descent**
- **Stochastic gradient descent**
- **Mini-batch gradient descent**

Exercises

- Linear Regression Exercise I. [Link (https://colab.research.google.com/drive/1lKt0hW3aWrKNwB0xxOQGjjWC8h3t_G8Q?usp=sharing)]
- Linear Regression Exercise II. [Link (https://drive.google.com/open?id=1uilwlUkAEe5QPiikYZRiB8VlN1FN-Twl)]

Suggested Readings

- Linear Regression Basics. [Link (https://towardsdatascience.com/a-beginners-guide-to-linear-regression-in-python-with-scikit-learn-83a8f7ae2b4f)]
- Intro to Linear Regressions and Gradient Descent. [Link (https://spin.atomicobject.com/2014/06/24/gradient-descent-linear-regression/)]

- Gradient Descent in-depth. [Link (https://www.freecodecamp.org/news/understanding-gradient-descent-the-most-popular-ml-algorithm-a66c0d97307f/)]

"Machine learning will automate jobs that most people thought could only be done by people." ~ Dave Waters