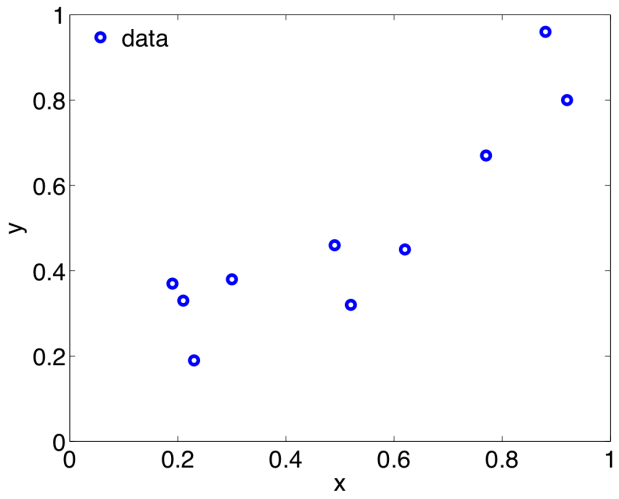
2 Least-Squares

2.1 Simple least-squares

Consider the following dataset. We have a bunch of inputs x_i and corresponding outputs y_i. The particular values in this dataset are

x	y
0.23	0.19
0.88	0.96
0.21	0.33
0.92	0.80
0.49	0.46
0.62	0.45
0.77	0.67
0.52	0.32
0.30	0.38
0.19	0.37



Now, suppose we'd like to fit a line to this data, of the form

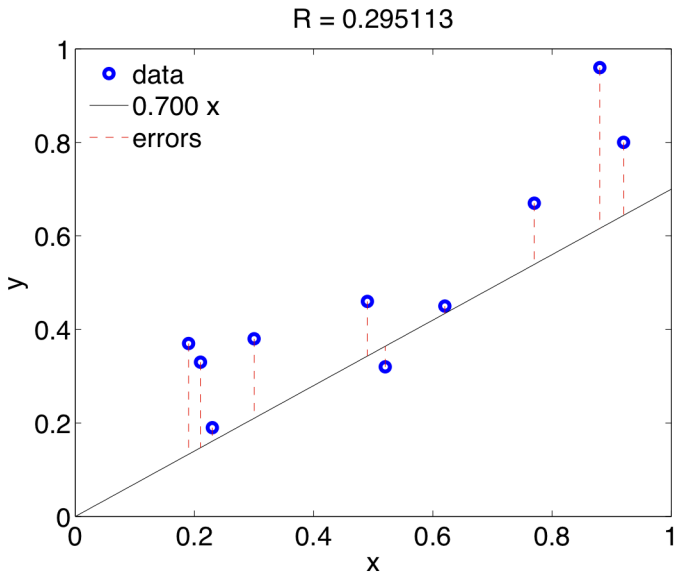
$$y = ax.$$

A standard way to fit such a line is by least-squares, where we measure the sum of the difference of each data point to the line with

$$R = \sum_i (ax_i - y_i)^2.$$

Exercise 8. If we use $a = 0.7$, what is the error R?

This can be visualized as:



Notice that we want to minimize the sum of the square of the length of the error lines. There are various justifications for this, but it is worth noting that one could alternatively minimize the length of the lines themselves, the cube of them, etc. The square is by far the most common, however, leads to simpler algorithms, and will be our focus here.

Now, how can we find the best a?

Theorem 9. *For $R = \sum_i (ax_i - y_i)^2$, the minimum value of R is obtained for*

$$a = \frac{\sum_i x_i y_i}{\sum_i x_i x_i}. \tag{2.1}$$

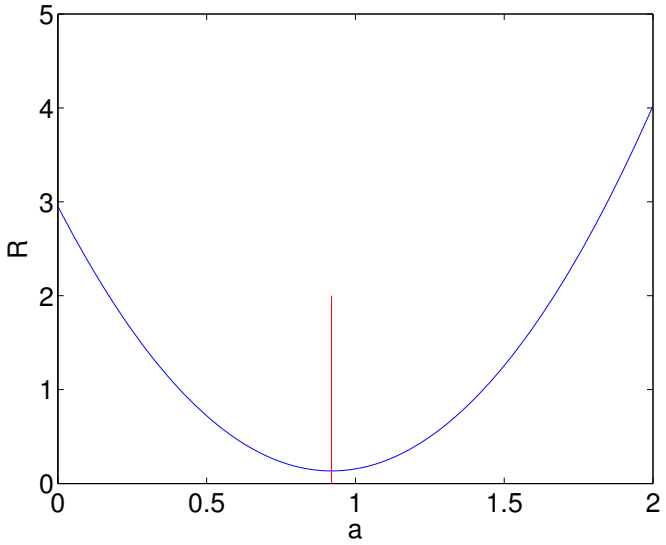
Proof. We can simply calculate the derivative of R and set it to zero.

$$\begin{aligned} \frac{dR}{da} &= 2\sum_i x_i(ax_i - y_i) \\ a\sum_i x_i x_i &= \sum_i x_i y_i \\ a &= \frac{\sum_i x_i y_i}{\sum_i x_i x_i} \end{aligned}$$

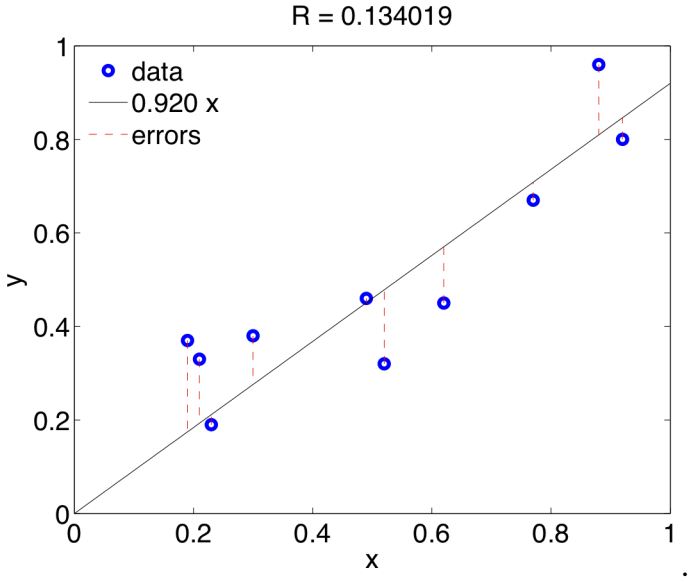
□

Exercise 10. Write a program that takes a bunch of values (x_i, y_i), along with a as input, and returns R. Next write a program to compute a from Eq. 2.1. Test that, on the above dataset, this program correctly returns $a = .920$. Compute the values of R for a range of a and see that this is the right value.

We can plot R for a range of a and see that this is the true minimum.



We can plot the data with the curve for this optimal a as



2.2 Least-squares with a bias term

It is common to add a "bias" term and, rather than fitting the linear function

$$y = ax,$$

fit the affine function

$$y = ax + b.$$

As before, we can use the basic calculus technique of setting derivatives to zero in order to find the best a and b.

Theorem 11. *For $R = \sum_i (ax_i + b - y_i)^2$, the minimum value of R is obtained for*

$$\begin{aligned} b &= \frac{\bar{y} - \bar{x}\bar{xy}/\bar{x^2}}{1 - \bar{x}^2/\bar{x^2}} \\ a &= \frac{1}{\bar{x}}(\bar{y} - b), \end{aligned}$$

where

$$\begin{aligned} \bar{y} &= \frac{1}{N}\sum_i y_i \\ \bar{x} &= \frac{1}{N}\sum_i x_i \\ \bar{x^2} &= \frac{1}{N}\sum_i x_i^2 \\ \bar{xy} &= \frac{1}{N}\sum_i x_i y_i. \end{aligned}$$

Proof. Start with the derivatives

$$\begin{aligned} \frac{dR}{da} &= 2\sum_i x_i(ax_i + b - y_i) \\ \frac{dR}{db} &= 2\sum_i (ax_i + b - y_i) \\ a\sum_i x_i^2 + b\sum_i x_i &= \sum_i x_i y_i \\ a\sum_i x_i + b\sum_i 1 &= \sum_i y_i \end{aligned}$$

Now, making use of the mean notation, we can write the linear system

$$\begin{aligned} a\bar{x^2} + b\bar{x} &= \bar{xy} \\ a\bar{x} + b &= \bar{y} \end{aligned}$$

To eliminate a, we can subtract $\bar{x}/\bar{x^2}$ times the first equation from the second, yielding

$$b - b\bar{x}\frac{\bar{x}}{\bar{x^2}} = \bar{y} - \frac{\bar{x}}{\bar{x^2}}\bar{xy}$$

Which we can solve for

$$b = \frac{\bar{y} - \bar{x}\bar{xy}/\bar{x^2}}{1 - \bar{x}^2/\bar{x^2}}.$$

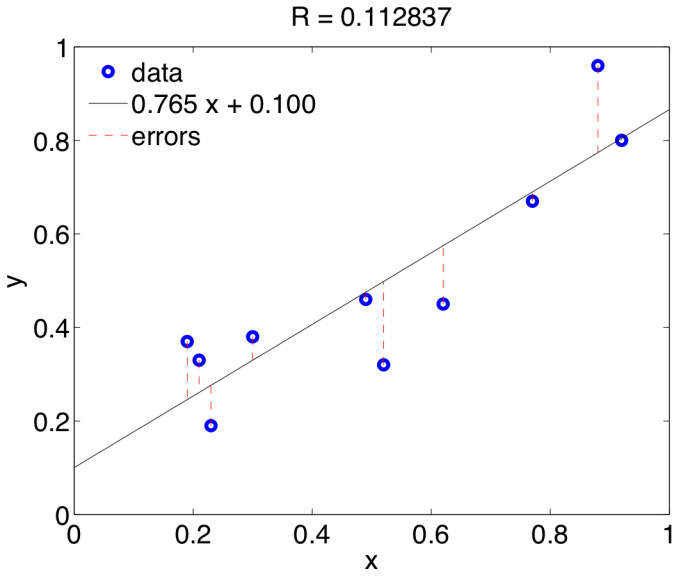
Once we have computed this, we can recover a from

$$a = \frac{1}{\bar{x}}(\bar{y} - b).$$

□

Not the prettiest theorem or proof in the world, but it works. Adding a bias term slightly improves the quality of the fit.

Exercise 12. Write a program that takes a bunch of values (x_i, y_i), and finds the optimal a and b for an affine fit $y = ax + b$. Verify that on the example dataset, you correctly find $a = .765$ and $b = .100$.



2.3 More powerful least squares.

Now, suppose we want to fit a quadratic

$$y = dx^2 + ax + b,$$

or, something crazy like

$$y = a\sin(x) + be^x + d\sqrt{x}.$$

It would be an option to derive an algorithm along the lines of the previous theorem, but one tends to shudder at the thought given how messy it was just to fit an affine function.

Instead, we will define an abstraction. Suppose that the input $\mathbf{z}$ is a *vector*, and we would like to make predictions by

$$y = \mathbf{w}^T\mathbf{z} = \sum_j \mathbf{w}(j)\mathbf{z}(j).$$

Theorem 13. *For $R = \sum_i (\mathbf{w}^T\mathbf{z}_i - y_i)^2$, the minimum value of R is obtained for*

$$\mathbf{w} = \big(\sum_i \mathbf{z}_i\mathbf{z}_i^T\big)^{-1}\big(\sum_i \mathbf{z}_i y_i\big).$$

Proof.

$$\begin{aligned}
\frac{dR}{d\mathbf{w}} = 0 &= 2\sum_i \mathbf{z}_i(\mathbf{w}^T\mathbf{z}_i - y_i) \\
\sum_i \mathbf{z}_i(\mathbf{w}^T\mathbf{z}_i) &= \sum_i \mathbf{z}_i y_i \\
\sum_i \mathbf{z}_i(\mathbf{z}_i^T\mathbf{w}) &= \sum_i \mathbf{z}_i y_i \\
\big(\sum_i \mathbf{z}_i\mathbf{z}_i^T\big)\mathbf{w} &= \big(\sum_i \mathbf{z}_i y_i\big)
\end{aligned}$$

□

To actually implement this, we would use something like the following

Algorithm (Least-Squares)

- Input $\{\mathbf{z}_i\}$, $\{y_i\}$
- $\mathbf{s} \leftarrow \sum_i \mathbf{z}_i y_i$
- $M \leftarrow \sum_i \mathbf{z}_i\mathbf{z}_i^T$

- Solve $M\mathbf{w} = \mathbf{s}$ for $\mathbf{w}$ using Gaussian Elimination
- Output $\mathbf{w}$.

Exercise 14. Implement the above algorithm for solving a least-squares system. Use your previous algorithm for solving linear systems. Check that, on the dataset

$\mathbf{z}(1)$	0.1	0.2	0.3	0.4
$\mathbf{z}(2)$	0.5	0.6	0.7	0.7
y	0.9	1.0	1.1	1.2

it correctly returns $\mathbf{w} = (-0.159, 1.739)$. Have your function also return the residual error R. Check that this is $R = 0.00927$. For testing, you might find it useful to check that

$$M = \begin{bmatrix} .3 & .66 \\ .66 & 1.59 \end{bmatrix},$$

$$\mathbf{s} = \begin{bmatrix} 1.1 \\ 2.66 \end{bmatrix}.$$

Note that this dataset means $\mathbf{z}_1 = (.1, .5)$, $\mathbf{z}_2 = (.2, .6)$, etc.

2.4 Basis expansion

Suppose that we can fit vector-valued least-squares systems of the form

$$y = \mathbf{w}^T \mathbf{z}.$$

Now the question is: what is the connection between all the measured data x and the input vector $\mathbf{z}$?

First, take our old scalar dataset. Suppose we would like to fit an equation of the form

$$y = ax + b.$$

How could we do this?

An idea would be to use what is called a basis expansion. Rather than fitting to the dataset $\{(x_i, y_i)\}$, fit to $\{(\mathbf{z}_i, y_i)\}$, where

$$\mathbf{z}_i = \begin{bmatrix} x_i \\ 1 \end{bmatrix}. \tag{2.2}$$

That is, we simply take the input dataset, and replace each input *scalar* by an input *vector* of length two, consisting of the original scalar plus a constant of one.

Then, we would have that

$$y = \mathbf{w}^T \mathbf{z}_i = w_1 x_i + w_2,$$

which is of exactly the same form as

$$y = ax + b.$$

For example, if we started with the dataset

x	1	2	5
y	2	7	9

then, after basis expansion, we would have the dataset

$\mathbf{z}(1)$	1	2	5
$\mathbf{z}(2)$	1	1	1
y	2	7	9

Exercise 15. Make a basis expansion for the original dataset of the form in Eq. 2.2. Plug it into your least-squares solver from the previous exercise. Check that you correctly find $\mathbf{w} = (.765, .100)$ and the residual error $R = 0.1128$.

Now, we have room for personal creativity. There is nothing that prevents us from fitting more advanced functions. For example, suppose we want to fit a function of the form

$$y = ax + b + cx^2.$$

What basis expansion should we use? The answer is

$$\mathbf{z}_i = \begin{bmatrix} x_i \\ 1 \\ x_i^2 \end{bmatrix}.$$

This is called a **quadratic basis expansion**.

Exercise 16. Make a quadratic basis expansion for the original dataset. Plug it into your least-squares solver from the previous exercise. Check that you correctly find $\mathbf{w} = (-.721, .406, 1.3$ and the residual error $R = 0.0632$.

If we are in a strange mood, we could even fit a function of the form

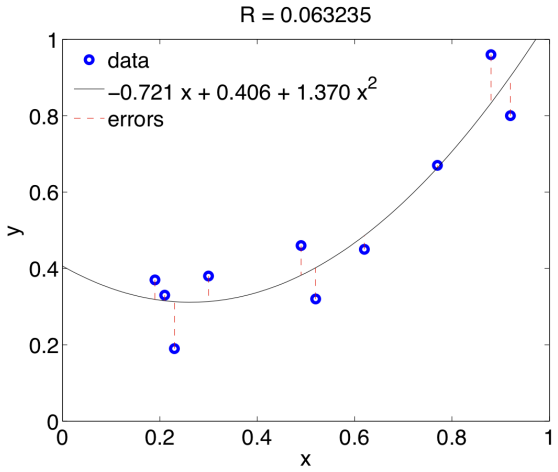
$$y = a\sin(2x) + b\log(x) + c\sqrt{x}$$

by simply using the basis expansion

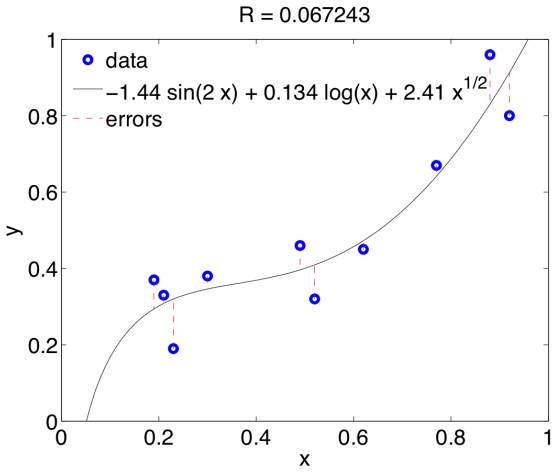
$$\mathbf{z}_i = \begin{bmatrix} \sin(2x_i) \\ \log(x_i) \\ \sqrt{x_i} \end{bmatrix}.$$

Exercise 17. Implement this last basis expansion. Check that you correctly find $\mathbf{w} = (-1.44, 0.134, 2.41)$ and the residual error $R = 0.0672$.

The output of the above two exercises will look like:



Exercise 16



Exercise 17

2.4.1 Matrix notation

Often, when working with a dataset of a bunch of vectors $\mathbf{z}_1$, $\mathbf{z}_2$, ... $\mathbf{z}_M$, it is convenient to put them together into a single matrix $\mathbf{Z}$ by simply placing the vectors next to each other.

$$\mathbf{Z} = \begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \cdots & \mathbf{z}_M \end{bmatrix}. \tag{2.3}$$

It is not hard to see that

Theorem 18. *As defined in Eq. 2.3,*

$$\mathbf{Z}\mathbf{y} = \sum_i \mathbf{z}_i y_i$$

A similar, but somewhat less obvious result is that

Theorem 19. *As defined in Eq. 2.3,*

$$\mathbf{Z}\mathbf{Z}^T = \sum_i \mathbf{z}_i \mathbf{z}_i^T.$$

All this means that we can write $\mathbf{w}$ in the more compact notation

$$(\mathbf{Z}\mathbf{Z}^T)\mathbf{w} = \mathbf{Z}\mathbf{y}.$$

Exercise 20. Write a program that takes inputs $\mathbf{Z}$ and $\mathbf{y}$ and finds the least-squares fit. (Use your previous routine to solve the linear system.) Check that, on an input of

$$\mathbf{Z} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 3 & 4 & 5 \end{bmatrix}, \ \mathbf{y} = (1, 2, 4, 5)^T,$$

it correctly returns

$$\mathbf{w} = (-1.9, 1.4)^T.$$

2.4.2 Linear or nonlinear?

Notice that, with a few of the above examples, we are actually fitting the output y using (sometimes highly nonlinear) functions with respect to the input x.

However, once the data are given, the function is linear with respect to the unknown coefficients / weights $\mathbf{w}$, and the optimal value of $\mathbf{w}$ is obtained with a simple least-square. This type of techniques is known as *linear regression*.

More formally, linear regression refers to the regression function's dependence on the regression coefficients $\mathbf{w}$, not on input data x. The crucial point here is that it is easy to do "nonlinear stuff" using linear regression with:

$$y = \mathbf{w}^T \mathbf{z}.$$

where

$$\mathbf{z} = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \cdot \\ \cdot \\ \cdot \\ f_s(x) \end{bmatrix}.$$

and $f_i(x)$ can be of various forms as shown in previous examples, and can be thought as *features* extracted from the given data x.

In summary, clever hand-engineered features and linear regression (or linear classification) captures a huge percentage of how machine learning is done in the real world. The advantages of linear methods like this are:

- Simplicity
- Reliability
- Interpretability
- Speed

There are also some disadvantages:

- Limited modeling power. Imposing a simple linear form $y = \mathbf{w}^T \mathbf{z}$ is a huge assumption and some datasets simply don't have such a relationship between inputs and outputs.
- You need to find good features. The previous issue can be reduced by coming up with good features, but this process is not automated. More fancy machine learning

methods try to adapt to the structure of the function without the user specifying it. Some, methods such as neural networks, can be understood as finding the features at the same time they fit $\mathbf{w}$. There is recent work on discovering such features in an unsupervised way.