Uncovering patterns in data

S. Marzen

April 19, 2023

So far, we've talked about modeling biophysical situations in one of two ways:

- The entire model is based on a few physical principles, e.g. revisit equilibrium statistical mechanics' role in understanding shape and forces or revisit the role of chemical reaction kinetics in understanding gene regulation and protein production.
- Part of the model is based on a few physical principles and the remainder is based on some data, e.g. revisit Levy flights from ecology.

What we'll talk about now are models that you make when you literally have no idea what is going on. Crudely speaking, this is what is typical in statistics and in machine learning. I should mention that things are changing. Nowadays, researchers adapt their models (sometimes) so that the model structure respects physical or biophysical assumptions. For instance, I was asked once to model the prevalence of a protein as a function of position and time. There were good reasons to assume a "reaction-diffusion" equation, basically meaning that there was some drift and some diffusive aspect to how the proteins moved. However, we had no clue what the reaction term was or what the diffusion constant was. These we could fit to data. This model is a hybrid between "physics constrains everything" and "we have no idea", as we are constraining the form of the model based on certain biophysical assumptions about how things should act, but not constraining all aspects of it. Only now are these hybrid models of dynamics beginning to take off in the machine learning literature.

I hope that, by now, if you are given some data, you will have the tools to develop a minimal model based on physical principles that describes some aspect of it. But what if you don't? What if the system is just too complicated and you don't know where to start? Never fear. There are a few things you can do regardless. In two of these situations, we'll imagine that we get pairs of input and output, x_i and y_i, and that we are trying to build a model that relates the two. In the last of these situations, we'll imagine that we just have a lot of data points x_i and that we are trying to find structure in the data, somehow.

1 Supervised learning

In this section, we imagine that we have pairs (x_i, y_i). These could be anything. For instance, x_i could be the ligand concentration and y_i could be whether or not the ligand-gated ion channel was open/how many ligands were bound on average over a period of time. In general, we will think of x_i as the input and y_i as the output, although you may want to adapt the interpretation to whatever problem you're dealing with.

Our goal will be to build a model of $p(y_i|x_i)$. If we want to, from that model, we can choose the most likely y_i given the input x_i. The only difference between regression and classification is based on the type of output data. If the output data can take values in $\mathbb{R}^n$, then we have regression; if the output data is categorical, such as "apples" and "oranges", then we have classification. The only material difference between the two is the *metric* that we use to evaluate the model we build and our predictions thereof.

In both cases, we take a Bayesian perspective in which we try to simultaneously maximize the log likelihood and minimize a regularization term, to be explained. It's worth mentioning that maximization of the log likelihood actually connects nicely to an information-theoretic concept that we skimmed over before

called the Kullback-Leibler divergence. The idea here is that there is some true distribution p_{data} from which our data is sampled and that p_{model} is our model of the data. One natural way of viewing model-fitting is to say that we are trying to make p_{data} and p_{model} as close as possible. If we view model-fitting in this way, rather than in the Bayesian way described below, we can choose literally any distance metric under the Sun. There is, however, a good argument for choosing the Kullback-Leibler divergence, despite it being highly sensitive to zero probabilities and not actually a distance metric:

$$D_{KL}[p_{data}||p_{model}] = \sum_x p_{data}(x) \log \frac{p_{data}(x)}{p_{model}(x)}. \tag{1}$$

If we are trying to guess data that comes out of an opaque box, and we have correctly modeled the data distribution as p_{data}, then we will need to ask on average $H[p_{data}]$ questions to understand each sample. But if we have the wrong model of the data, p_{model}, then we need an additional $D_{KL}[p_{data}||p_{model}]$ questions on average. Furthermore, a little math convinces us that

$$D_{KL}[p_{data}||p_{model}] = \sum_x p_{data}(x) \log p_{data}(x) - \sum_x p_{data}(x) \log p_{model}(x). \tag{2}$$

The first term is the negative entropy of the data, which is something that we cannot alter by changing our model. The second term is the cross-entropy or (more famously) the log likelihood. And so, by minimizing a particular "distance" between model and data, we have maximized the log likelihood. In other words, there are multiple roads that lead us to maximizing the log likelihood, though for our purposes we will stick with maximization of the log posterior.

1.1 Regression

Let's start with an example and build up. The simplest example of regression is linear regression. Imagine that x_i is some vector of real numbers and that y_i is a single real number. For instance, maybe x_i are behavioral features and y_i is someone's change in body weight over the course of a month. I'm going to posit– without any real reason why I should posit this– that

$$y_i = w^\top x_i + b + z_i \tag{3}$$

where w is a vector of weights/coefficients, b is a bias term, and z_i is zero-mean Gaussian noise. If someone were to ask us to make a prediction about y_i given only the input x_i, we would guess (for reasons that will be discussed in a second) $w^\top x_i + b$. That is, the elements of w tell us what weight we give to each aspect of the input, and the bias corrects for (usually unimportant) shifts in the average value of the output. Another, completely equivalent way of describing the relationship in Eq. 3 is to write it probabilistically:

$$p_\theta(y_i|x_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - w^\top x_i - b)^2}{2\sigma^2}\right). \tag{4}$$

Essentially, we are making a choice to model the output as a Gaussian given the input with mean $w^\top x_i + b$ and standard deviation σ^2. The θ subscript refers to w, b, and maybe even σ (to come later) as the list of parameters involved in this model.

Brainstorm, with your in-class group, three or four datasets that you could use linear (or nonlinear) regression to understand. Clearly identify what x_i and y_i are. Is a linear model reasonable? What about the assumption of Gaussian noise?

Now, linear regression relies on Eq. 3 being at least somewhat accurate. But let's imagine why things might deviate. It is a general rule in physics that as long as you don't have a large dynamic range on your input, you can Taylor expand pretty much any input-output relationship and find something approximately linear. But what if the dynamic range on your input is large? Or what if your noise isn't Gaussian? The

first of these complications would cause us to go from linear regression to nonlinear regression. The second of these complications leads to a more subtle change in how we want to approach the problem– which objective function we choose to minimize.

To give you a sense of the space of nonlinear regressors, imagine that you had a powerful "function approximator" that could literally nearly approximate any function. Further imagine that you had a powerful set of learning algorithms that could train the parameters underlying this function approximator so that, in finite time, you really do learn a function that matches the data. Surprisingly, these things exist. They are called neural networks. We will not tackle neural networks in this class, but one way of thinking about the utility of neural networks is simply to think of them as universal function approximators– trainable systems that can approximate literally any function to within any desired accuracy. For any of these universal function approximators, you can choose your objective function at will. The art of choosing the structure of the neural network, its learning algorithm, and its objective function are good things to learn and will net you large salaries at Google.

Back to linear regression! We will approach this from a Bayesian perspective, which will naturally become the more typical perspective you'll see on, say, Towards Data Science. What we want to do is choose parameters θ (comprising $w,\ b,\ \sigma$) that maximize the posterior:

$$\theta^* \quad = \quad \arg\max_\theta p(\theta|\{x_i, y_i\}). \tag{5}$$

Basically, we want the parameters we choose to be as reflective of the data as possible. Recalling Bayes' rule,

$$p(\theta|\{x_i, y_i\}) \quad = \quad \frac{p(\{x_i, y_i\}|\theta)p(\theta)}{p(\{x_i, y_i\})}, \tag{6}$$

we find that

$$\begin{aligned} \theta^* &= \arg\max_\theta \frac{p(\{x_i, y_i\}|\theta)p(\theta)}{p(\{x_i, y_i\})} && (7)\\ &= \arg\max_\theta p(\{x_i, y_i\}|\theta)p(\theta) && (8)\\ &= \arg\max_\theta \log p(\{x_i, y_i\}|\theta) + \log p(\theta). && (9) \end{aligned}$$

We're going to assume that each of our data points (x_i, y_i) is completely independent of every other data point, so that

$$\begin{aligned} p(\{x_i, y_i\}|\theta) &= \prod_i p(x_i, y_i|\theta) && (10)\\ \log p(\{x_i, y_i\}|\theta) &= \sum_i \log p(x_i, y_i|\theta) && (11) \end{aligned}$$

and therefore

$$\theta^* \quad = \quad \arg\max_\theta \sum_i \log p(x_i, y_i|\theta) + \log p(\theta). \tag{12}$$

Let's take a quick look at our objective function. It's comprised of two parts: one part that has to do with the data itself, $\sum_i \log p(x_i, y_i|\theta)$, the log likelihood; and one part that has to do with our prior belief about what the parameters look like, $\log p(\theta)$. As we acquire more data, the log likelihood will always overtake the prior, and we will choose parameters that cause maximal match between the model and the data. However, when there isn't enough data, the prior will govern the parameters that we find, and we may end up choosing– for instance– parameters that are close to 0. The choice of prior can be thought of as a *regularization*, the utility of which we will cover in a second when we look at linear tricks to do nonlinear regression.

At this point, the objective function looks intractable, but a little massaging gets it into a decent form. Recall Eq. 4. From that, we can simplify the log likelihood, assuming that our prior over inputs is uniform:

$$\log p(x_i, y_i|\theta) = \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y_i - w^\top x_i - b)^2}{2\sigma^2}\right)\right) \quad (13)$$

$$= -\frac{1}{2}\log(2\pi\sigma^2) - \frac{(y_i - w^\top x_i - b)^2}{2\sigma^2} \quad (14)$$

$$\sum_i \log p(x_i, y_i|\theta) = -\frac{|\mathcal{D}|}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{|\mathcal{D}|}(y_i - w^\top x_i - b)^2 \quad (15)$$

and therefore

$$\theta^* = \arg\max_\theta -\frac{|\mathcal{D}|}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{|\mathcal{D}|}(y_i - w^\top x_i - b)^2 + \log p(w, b, \sigma). \quad (16)$$

The last thing we have to do is choose a prior over parameters. In all honesty, we don't care very much about σ, and so we will focus on w and b. Typically, we end up assuming that these parameters are independent (because we can, though you can do something different), that $p(w, b, \sigma) = \prod_i p(w_i)p(b)p(\sigma)$, where $p(w_i)$ is our prior belief on the i^{th} element of w. Now what do we do?

Again, you have no physical or biophysical model guiding your understanding of what w_i, b, or σ is likely to be. You are essentially out in the Wild Wild West, choosing whatever prior you feel like. There are a few key guiding concepts to choosing a good prior, the main one being: a good prior should prevent *overfitting*. Imagine that you are modeling how x relates to y and that you have decided that the relationship is nonlinear, and that in order to fit a nonlinear function in an easy-to-fit way, you have decided to write

$$y = w_0 + w_1 x + w_2 x^2 + ... + w_9 x^9 + z, \quad (17)$$

with z zero-mean Gaussian noise. This is equivalent to linear regression when $(x, x^2, ..., x^9)$ is treated as the input. So now let's imagine a weird and extreme situation, in which you only have nine data points. You may not know this, but you will be able to find weights w_i that cause the prediction $\hat{y} = w_0 + w_1 x + w_2 x^2 + ... + w_9 x^9$ to exactly match the true answer. That sounds like a great thing, but it might not be. What if your data actually suggest a line with some scatter? Then, if someone asks you to predict y at a large x, your complicated ninth-order polynomial that exactly fits your data will get the answer very, very wrong. In jargon, you can interpolate, but you can't extrapolate.

To avoid this catastrophe, we do something called *regularize*. Basically, we choose priors that prefer weights that are closer to 0. That way, w_9 is unlikely to be large in the previous example, and you're unlikely to be as bad as extrapolating. Within those guidelines– send weights to 0– there is plenty of room for personal choice. Here are three typical priors, one of which does not regularize at all:

- $p(\theta)$ is uniform over some potentially bounded region of space– an uninformed prior that makes no attempt to prefer zero weights;
- $p(\theta)$ is Gaussian with a mean at 0– leading to what's called "ridge regression";
- and $p(\theta) \propto e^{-\lambda|\theta|}$, leading to Lasso regression.

Of the two priors that prefer zero weights, the Gaussian prior is softer. You tend to see weights that are closer to zero, but not equal to zero, while with the harsher prior, weights literally hit 0 when they are not needed. In general, if one chooses priors of the form $e^{-\lambda|\theta|^p}$ where p is a real number, weights will hit 0 when $p \leq 1$ and will prefer 0 but not hit 0 otherwise. All this is to say, the choice of priors can be tricky, and in some ways it is an art.

When we take logs, priors become what are called *regularizers*. From Eq. 16, we deem our "objective function" to be

$$\mathcal{L} = -\frac{|\mathcal{D}|}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{|\mathcal{D}|}(y_i - w^\top x_i - b)^2 + \log p(w, b, \sigma). \tag{18}$$

We are trying to maximize this log posterior. The first two terms correspond to the log likelihood, and the second of these two terms is a constant off of what is known as the mean-squared error, a measure of our model's inability to match the output. The last term is the log of our prior, and it is known as a regularizer:

- the uninformed prior results in no regularizer;
- the Gaussian prior results in a regularizer of $\frac{b^2}{2\lambda} + \sum_i \frac{w_i^2}{2\lambda}$
- the Laplacian prior results in a regularizer of $\frac{|b|}{\lambda} + \sum_i \frac{|w_i|}{\lambda}$.

Our goal is to not only maximize the log likelihood by setting weights so that the model exactly matches the data, but to simultaneously minimize the regularizer by setting the weights to be as small as possible, and this results in a tug-of-war. The one that wins is governed by $\lambda|\mathcal{D}|$, as we shall see– the strength of the prior relative to the amount of data.

With this objective function in hand, there are in theory two ways of approaching linear regression, one analytical and one numerical. Taking an arg max can be difficult, but when the parameter is any real number or vector of real numbers, and when your objective function is so-called "convex" (and we won't go into this), it's easy. You simply take the gradient of the objective with respect to θ, $\nabla_\theta \mathcal{L}$, and set the answer equal to the 0 vector. The two approaches become the following:

- One can numerically move θ up the gradient of $\mathcal{L}$ in whatever steps one feels like. You may remember this from multivariable calculus as gradient ascent. Plenty of software packages implement this approach.
- One can analytically solve for the w, b, and σ that set the gradient to the zero vector. This approach usually doesn't work, but sometimes, one can turn the corresponding equations into iterative solutions.

Usually, one should numerically ascend the gradient of $\mathcal{L}$ using ScikitLearn or the like to choose weights. To choose a prior, standard practice is to divide your data into a training, validation, and test set. The test set is there so that you can accurately calibrate how well your model is doing, e.g. what is the eventual mean-squared error. The training set is there to choose weights for a particular set of "hyperparameters" that have to do with the prior, and the validation set is there so that you can choose which prior you want. The idea is this. We'll pick a set of priors that we might want to choose. For each one of those priors, we use the training set to choose weights, and then measure performance on the validation set. Then, you choose the prior that performs best on the validation set, and measure its final performance on the test set. **Need picture.**

If you want to merely use linear regression, stop here! The above paragraph is all you need. What follows is designed to give you intuition for linear regression with an uninformed prior and ridge regression using as much analytical horsepower as we can muster.

1.1.1 Uninformed prior

In the first case, $p(\theta)$ is uniform. In this limit, Eq. 16 turns into

$$\theta^* = \arg\max_\theta -\frac{|\mathcal{D}|}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{|\mathcal{D}|}(y_i - w^\top x_i - b)^2. \tag{19}$$

The calculus is grungy, and I include it here only if you're interested in the grunge. Solutions are provided after the grunge, and I encourage you to skip to these. We'll start with b:

$$\begin{aligned} 0 &= \frac{\partial \mathcal{L}}{\partial b} && (20)\\ &= \frac{\partial}{\partial b} \sum_{i=1}^{|\mathcal{D}|} (y_i - w^\top x_i - b)^2 && (21)\\ &= \sum_{i=1}^{|\mathcal{D}|} (y_i - w^\top x_i - b) && (22)\\ 0 &= |\mathcal{D}|\langle y \rangle - |\mathcal{D}| w^\top \langle x \rangle - |\mathcal{D}| b && (23)\\ &= \langle y \rangle - w^\top \langle x \rangle - b. && (24) \end{aligned}$$

We'll need to solve for b:

$$b = \langle y \rangle - w^\top \langle x \rangle. \quad (25)$$

Now we'll move to w:

$$\begin{aligned} 0 &= \frac{\partial \mathcal{L}}{\partial w_j} && (26)\\ &= \frac{\partial}{\partial w_j} \sum_{i=1}^{|\mathcal{D}|} (y_i - w^\top x_i - b)^2 && (27)\\ &= \sum_{i=1}^{|\mathcal{D}|} x_{i,j} (y_i - w^\top x_i - b) && (28)\\ &= \langle x_j y \rangle - w^\top \langle x x_j \rangle - b \langle x_j \rangle. && (29) \end{aligned}$$

Let's write this in vector form:

$$\vec{0} = \langle xy \rangle - w^\top \langle x x^\top \rangle - b \langle x \rangle \quad (30)$$

We now have enough information to solve for w and b. Substituting Eq. 25 into Eq. 30 gives

$$\begin{aligned} \vec{0} &= \langle xy \rangle - w^\top \langle x x^\top \rangle - \left(\langle y \rangle - w^\top \langle x \rangle \right) \langle x \rangle && (31)\\ &= \langle xy \rangle - w^\top \langle x x^\top \rangle - \langle x \rangle \langle y \rangle + w^\top \langle x \rangle \langle x \rangle && (32)\\ &= \left(\langle xy \rangle - \langle x \rangle \langle y \rangle \right) - w^\top \left(\langle x x^\top \rangle - \langle x \rangle \langle x^\top \rangle \right) && (33)\\ w^\top \left(\langle x x^\top \rangle - \langle x \rangle \langle x^\top \rangle \right) &= \langle xy \rangle - \langle x \rangle \langle y \rangle && (34)\\ w &= \left(\langle x x^\top \rangle - \langle x \rangle \langle x^\top \rangle \right)^{-1} \left(\langle xy \rangle - \langle x \rangle \langle y \rangle \right). && (35) \end{aligned}$$

We now have a closed-form solution for w. I should add that in practice, these covariance matrices $\langle x x^\top \rangle - \langle x \rangle \langle x^\top \rangle$ can be close to singular, and so it is sometimes way better numerically to do gradient ascent. Even so, the closed form solution can be useful in other ways (e.g., ask me about some of my work on linear recurrent networks) and it also provides some insight about what matters for any particular weight. Namely, a particular weight is larger if x_i and y are more strongly correlated, though the strength of this weight is modulated by how strongly correlated x_i is with x_j. From this closed-form expression for w and Eq. 25, we immediately find

$$b = \langle y \rangle - \left(\langle xy \rangle - \langle x \rangle \langle y \rangle \right)^\top \left(\langle x x^\top \rangle - \langle x \rangle \langle x^\top \rangle \right)^{-1} \langle x \rangle. \quad (36)$$

This nasty expression is hard to interpret, but it breaks up into two terms. The first term corresponds to the mean value of the output y, and the second term corresponds to the mean value of the prediction for y. Any mean difference between the two is shoved into the intercept b, as one might expect. One can also

mathematically add a 1 to the input vector to take the place of an explicit intercept. In such a case, Eq. 35 is all you need.

Finally, let's turn our attention to σ, the size of the noise. The inferred size of the noise should be determined by how wildly inaccurate our predictions are. If we substitute our expressions for b and w into the objective function, we find

$$\begin{aligned}
\mathcal{L} &= -\frac{|\mathcal{D}|}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{|\mathcal{D}|}(y_i - w^\top x_i - b)^2 && (37)\\
&= -\frac{|\mathcal{D}|}{2}\log(2\pi\sigma^2) - \frac{|\mathcal{D}|}{2\sigma^2}\langle(y - w^\top x - b)^2\rangle && (38)\\
&= -\frac{|\mathcal{D}|}{2}\log(2\pi\sigma^2) - \frac{|\mathcal{D}|}{2\sigma^2}\left(\langle y^2\rangle - 2w^\top\langle xy\rangle - 2b\langle y\rangle + b^2 + 2bw^\top\langle x\rangle + w^\top\langle xx^\top\rangle w\right) && (39)\\
&= -\frac{|\mathcal{D}|}{2}\log(2\pi\sigma^2) - \frac{|\mathcal{D}|}{2\sigma^2}\left(\sigma_{yy} - \sigma_{xy}^\top\sigma_{xx}^{-1}\sigma_{xy}\right) && (40)
\end{aligned}$$

where I have skipped some of the gory algebra and where $\sigma_{xy} = \langle xy\rangle - \langle x\rangle\langle y\rangle$, $\sigma_{xx} = \langle xx^\top\rangle - \langle x\rangle\langle x^\top\rangle$, and $\sigma_{yy} = \langle y^2\rangle - \langle y\rangle^2$. The expression $\sigma_{yy} - \sigma_{xy}^\top\sigma_{xx}^{-1}\sigma_{xy}$ is the mean-squared error, or rather, on net how wrong our predictions are. It is the difference of two terms. The first term is σ_{yy}, the variance associated with the outputs. The second term looks complicated, $\sigma_{xy}^\top\sigma_{xx}^{-1}\sigma_{xy}$, and it is how much of the fluctuations in the output are explained by our model. The difference of these two is how much of the fluctuations in the output we fail to explain with our linear model.

With these simplifications, our

$$\begin{aligned}
0 &= \frac{\partial}{\partial\sigma}\mathcal{L} && (41)\\
&= \frac{d}{d\sigma}\left(-\frac{|\mathcal{D}|}{2}\log(2\pi\sigma^2) - \frac{|\mathcal{D}|}{2\sigma^2}\left(\sigma_{yy} - \sigma_{xy}^\top\sigma_{xx}^{-1}\sigma_{xy}\right)\right) && (42)\\
&= \frac{d}{d(\sigma^2)}\left(\frac{1}{2}\log\sigma^2 + \frac{1}{2\sigma^2}(\sigma_{yy} - \sigma_{xy}^\top\sigma_{xx}^{-1}\sigma_{xy})\right) && (43)\\
&= \frac{1}{\sigma^2} - \frac{1}{\sigma^4}(\sigma_{yy} - \sigma_{xy}^\top\sigma_{xx}^{-1}\sigma_{xy}) && (44)\\
&= \sigma^2 - (\sigma_{yy} - \sigma_{xy}^\top\sigma_{xx}^{-1}\sigma_{xy}) && (45)\\
\sigma^2 &= \sigma_{yy} - \sigma_{xy}^\top\sigma_{xx}^{-1}\sigma_{xy} && (46)\\
\sigma &= \sqrt{\sigma_{yy} - \sigma_{xy}^\top\sigma_{xx}^{-1}\sigma_{xy}}. && (47)
\end{aligned}$$

So the estimated noise is exactly the unexplained variance in the outputs.

After all this algebra, what have we learned? We have estimated the weights as being the covariance between inputs and outputs, weighted by the inverse of the covariance of the inputs. We have found out that the estimated noise is exactly the unexplained variance. And it turns out that the amount of variance explained, normalized by the total amount of variance to explain, is known as the (squared) correlation coefficient,

$$\rho^2 = \frac{\sigma_{xy}^\top\sigma_{xx}^{-1}\sigma_{xy}}{\sigma_{yy}}. \quad (48)$$

The larger this squared correlation coefficient, the better our model. But be careful. If we build a model that manages to explain by overfitting, ρ^2 will be artificially large. It pays to divide one's data into a train set and a test set, and to train the weights on the training data but to calculate the squared correlation coefficient on the test set. Otherwise, you are liable to conclude that you have far more explanatory power than you actually do.

1.1.2 Ridge regression

Let's say we want to avoid overfitting, but still want to build our linear model. What we'll do is add a term that sends the weights to 0, if possible:

$$p(w, b, \sigma) = e^{-b^2/\lambda} \prod_i e^{-w_i^2/\lambda}. \tag{49}$$

It is not necessary to preference σ to be close to 0, as higher or lower estimates of the unexplained variance in outputs don't lead to overfitting. The parameter λ can be as small or as large as we choose. If it's large, then b and w_i are only slightly biased towards 0, and we are essentially in the regime of the uninformed prior detailed in the last subsection. However, when λ is small, we are more strongly asking weights to remain near 0. We'll get into how to choose λ in a second.

With this prior in hand, we are poised to modify our objective function as follows:

$$\mathcal{L} = -\frac{|\mathcal{D}|}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{|\mathcal{D}|}(y_i - w^\top x_i - b)^2 - \frac{b^2}{2\lambda} - \sum_i \frac{w_i^2}{2\lambda}. \tag{50}$$

There are factors related to λ that we have dropped, as they have no dependence on σ, b, w. One can actually keep these terms in and use them to choose λ, but we won't make analytic headway that way, so I'll tell you about a simpler technique for choosing the prior. This is essentially the same objective function as before with a few small differences. Let's take a look at what happens to the gradient:

$$\begin{aligned}
0 &= \frac{\partial \mathcal{L}}{\partial b} && (51)\\
&= \frac{\partial}{\partial b}\left(-\frac{|\mathcal{D}|}{2\sigma^2}\langle (y - w^\top x - b)^2\rangle - \frac{b^2}{2\lambda}\right) && (52)\\
&= \frac{|\mathcal{D}|}{\sigma^2}\langle y - w^\top x - b\rangle - \frac{b}{\lambda} && (53)\\
&= \langle y\rangle - w^\top\langle x\rangle - b - \frac{\sigma^2}{\lambda|\mathcal{D}|}b && (54)\\
b &= \frac{1}{1 + \frac{\sigma^2}{\lambda|\mathcal{D}|}}\left(\langle y\rangle - w^\top\langle x\rangle\right). && (55)
\end{aligned}$$

This is nearly the same as the formula we had before for the uninformed prior. The only difference is a prefactor that decreases our estimate of the intercept– in essence, doing exactly what we expected by preferencing the intercept towards 0. The more data $|\mathcal{D}|$ we have, the less the preference; but the stronger our prior $\frac{1}{\lambda}$, the stronger the preference.

Let's look at our new gradient equations for w_j:

$$\begin{aligned}
0 &= \frac{\partial \mathcal{L}}{\partial w_j} && (56)\\
&= -\frac{|\mathcal{D}|}{\sigma^2}\langle x_j(y - w^\top x - b)\rangle - \frac{w_j}{\lambda} && (57)\\
&= \frac{|\mathcal{D}|}{\sigma^2}\left(\langle y x_j\rangle - w^\top\langle x x_j\rangle - b\langle x_j\rangle\right) + \frac{w_j}{\lambda}. && (58)
\end{aligned}$$

In vectorized form, this becomes

$$\vec{0} = \frac{|\mathcal{D}|}{\sigma^2}\left(\langle xy\rangle - w^\top\langle x x^\top\rangle - b\langle x\rangle\right) + \frac{1}{\lambda}w. \tag{59}$$

Notice– there's only a slight change from before– the addition of $\frac{1}{\lambda}w$. This should have more weight precisely when λ is small, or when we place a strong prior on the weights being small.

Now, we combine (as before) the equations for b and w:

$$\vec{0} = \left(\langle xy\rangle - w^\top\langle xx^\top\rangle - \frac{1}{1+\frac{\sigma^2}{\lambda|\mathcal{D}|}}\left(\langle y\rangle - w^\top\langle x\rangle\right)\langle x\rangle\right) + \frac{\sigma^2}{\lambda|\mathcal{D}|}w \tag{60}$$

$$= \frac{\sigma_{xy}+\frac{\sigma^2}{\lambda|\mathcal{D}|}\langle xy\rangle}{1+\frac{\sigma^2}{\lambda|\mathcal{D}|}} - \left(\frac{\sigma_{xx}+\frac{\sigma^2}{\lambda|\mathcal{D}|}\langle xx^\top\rangle}{1+\frac{\sigma^2}{\lambda|\mathcal{D}|}} + \frac{\sigma^2}{\lambda|\mathcal{D}|}I\right)w. \tag{61}$$

This solves as

$$w = \left(\frac{\sigma_{xx}+\frac{\sigma^2}{\lambda|\mathcal{D}|}\langle xx^\top\rangle}{1+\frac{\sigma^2}{\lambda|\mathcal{D}|}} + \frac{\sigma^2}{\lambda|\mathcal{D}|}I\right)^{-1}\left(\frac{\sigma_{xy}+\frac{\sigma^2}{\lambda|\mathcal{D}|}\langle xy\rangle}{1+\frac{\sigma^2}{\lambda|\mathcal{D}|}}\right). \tag{62}$$

This looks more complicated, but the key point here is the parameter $\frac{\sigma^2}{\lambda|\mathcal{D}|}$. When we have a large amount of data compared to the strength of our prior, this parameter is small, and the solution for ridge regression is almost nearly the solution with the uninformed prior. It's only when this is not the case that a very different solution takes hold. When this parameter is large, the optimal weights look almost entirely different:

$$w \approx \left(\frac{\sigma^2}{\lambda|\mathcal{D}|}\right)^{-1}\langle xy\rangle = \frac{\lambda|\mathcal{D}|}{\sigma^2}\langle xy\rangle. \tag{63}$$

Still, the weights are governed by how x correlates with y, but it is interesting that we no longer subtract the mean $\langle x\rangle\langle y\rangle$. Furthermore, since $\langle xy\rangle$ is multiplied by $\lambda|\mathcal{D}|$, these weights can be vanishingly small, as one would expect if our prior is to have small weights. As $|\mathcal{D}|$ increases, we interpolate smoothly between these two extremes.

One thing we have not yet talked about is what σ is. Based on our experience with the uninformed prior, it must be the unexplained variance– but now the unexplained variance is going to be some complicated function of σ. Hence, these are not really closed-form solutions. What should we do? Exactly what the paragraph before the start of these subsections outlined: use ScikitLearn.

1.1.3 Lasso regression

In this case, we suppose that $p(w,b,\sigma) \propto \prod_i e^{-|w_i|/\lambda}e^{-|b|/\lambda}$, and get a slightly different objective function:

$$\mathcal{L} = -\frac{|\mathcal{D}|}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{|\mathcal{D}|}(y_i - w^\top x_i - b)^2 - \frac{|b|}{\lambda} - \sum_i \frac{|w_i|}{\lambda}. \tag{64}$$

Rather than go through the same exercise in getting as close to closed-form solutions as possible, I will reiterate that what one does in practice is to follow the paragraph write before these subsections.

Essentially, what happens is that not only are weights decreased from their uninformed magnitude, but that nonessential weights are sent to 0. Literally 0. Some people view this prior as an approximation to a regularizer that asks for as few nonzero weights as possible. This prior is therefore, in some sense, a way to do feature selection.

1.2 Classification

Now we imagine a different sort of problem, in which y_i is one of a potentially non-numeric set. For instance, y_i might be "apple" or "banana", and x_i might be the size of the fruit and the color of its skin. Or, more relevant to biophysics: maybe y_i is whether or not a bacterium runs or tumbles and x_i is the history of chemoattractant that it has seen. Either way, we are still trying to model $p(y_i|x_i)$. The quickest and easiest

model to make is that of logistic regression, which applies to the case in which we are trying to evaluate probabilities of things that are either *this* ($y = 0$) or *that* ($y = 1$):

$$p(y_i = 1|x_i) = \frac{\exp\left(w^\top x_i + b\right)}{1 + \exp\left(w^\top x_i + b\right)}. \tag{65}$$

Equivalently, we have

$$p(y_i = 0|x_i) = \frac{1)}{1 + \exp\left(w^\top x_i + b\right)}. \tag{66}$$

Just as in linear regression, we are trying to learn w and b. Notice that unlike linear regression, we do not have a Gaussian noise model. Rather, the noise model is implicit in Eq. 65.

Brainstorm, with your in-class group, three or four datasets that you could use logistic regression to understand. Clearly identify what x_i and y_i are. Is a linear model reasonable? How could you make a nonlinear model that is still within the purview of logistic regression?

One can certainly envision datasets that have more than two classes. In that case, what should we do? We are now making a model of $p(y_i|x_i)$ where y_i is not just (say) active or inactive, but instead equal to 0, 1, 2, 3, or 4. In such cases, one can keep the spirit of logistic regression without its particulars. Our model of $p(y_i|x_i)$ can be proportional to an exponential of a linear function of x_i. This is not the only prescription. When making models of neural firing, one often employs a Generalized Linear Model in which $p(y_i|x_i)$, with y_i the number of spikes, is assumed to be a Poisson distribution with the rate equal to some nonlinear function of a linear function of x_i. What model you choose has a lot to do with your data, and is an art more than a science. If you have no idea what noise model to choose, try to choose a noise model that has so many parameters that it can approximate any noise model under the Sun. (This is machine learning in a nutshell.)

But back to logistic regression. We would like to evaluate, again from a Bayesian perspective, what w and b to choose. As before, this comes out to choosing $\theta = (w, b)$ so as to maximize the posterior:

$$p(\theta|\{x_i, y_i\}) \quad = \quad \frac{p(\{x_i, y_i\}|\theta)p(\theta)}{p(\{x_i, y_i\})}. \tag{67}$$

Or,

$$\begin{aligned}
\theta^* &= \arg\max_\theta p(\theta|\{x_i, y_i\}) && (68)\\
&= \arg\max_\theta \frac{p(\{x_i, y_i\}|\theta)p(\theta)}{p(\{x_i, y_i\})} && (69)\\
&= \arg\max_\theta \log p(\{x_i, y_i\}|\theta) + \log p(\theta). && (70)
\end{aligned}$$

Again, as in linear regression, the first term is the log likelihood, and the second term depends on our priors for what w and b are likely to be. Just as before, assuming that the inputs are equally likely to be anywhere,

$$\log p(\{x_i, y_i\}|\theta) = \sum_i p(y_i|x_i, \theta) = |\mathcal{D}|\langle p(y|x, \theta)\rangle. \tag{71}$$

So just as with linear regression, the log likelihood term is far bigger when there is more data. Let's talk for a second about the low data limit. In linear regression, this led us to consider the issue of overfitting, wherein we could choose a model that perfectly modeled all the given data but did a terrible job of extrapolating. A similar thing can happen here. In fact, we can do this by a pretty simple prescription if we have few enough data points: find w and b such that $w^\top x + b$ has the right sign for each of the given inputs, and then multiply w and b by arbitrarily large constants. With this prescription, you are guaranteed to model the input data perfectly. However, on new data, you are likely to be very certain and very wrong. Hence,

$\log p(\theta)$ again plays the role of regularization to prevent this overfitting from happening. The method for choosing priors is roughly the same as before. There will usually be some shape parameter λ as before that modulates the strength of the priors, and as with linear regression, the parameter that governs how closely we pay attention to priors is $\lambda|\mathcal{D}|$.

There are no closed-form solutions that I know of for w and b, so you are safe from grungy algebra!

2 Dimensionality reduction

Sometimes, instead of modeling the relationship between inputs and outputs, we simply have a ton of inputs, and we want to find structure in the inputs. Typically, this looks like a realization that the inputs are well-described by some "latent variable" that is low-dimensional. For instance, maybe the inputs look like **Fig. ??**. It appears that there are two clusters of some size. Wouldn't it be easier to simply tell the experimentalist whose data you are analyzing that they essentially have two types of input?

With that idea in mind, let's develop an objective function to uncover these latent variables. We imagine that if we have input x, we encode it as $f_\theta(x)$, where $f_\theta(x)$ is lower-dimensional. (So, if x is a real-valued vector with five dimensions, $f_\theta(x)$ might be a real-valued scalar.) We then decode our encoding using g_ϕ, so that we nearly recover x as $g_\phi(f_\theta(x))$. At the end of all this, we mark ourselves as having come $d(x, g_\phi(f_\theta(x)))$ to recovering the input, where d is some "distortion measure", to be discussed. What we'd like to do is find parameters $\theta,\ \phi$ that underlie our encoding and decoding functions such that we minimize

$$\phi^*,\ \theta^* = \arg\min_{\phi,\ \theta} \langle d(x, g_\phi(f_\theta(x)))\rangle. \tag{72}$$

If f were allowed to be any dimension, this problem would be very easily solved: simply set f and g to the identity. But we include a constraint: we say that f is of a certain dimension that is lower than the dimension of the input. In this way, we usually guarantee that there will be some error in recovery. The problem of choosing optimal encoders and decoders becomes difficult.

When things are difficult, we always turn to a few typical simplifying assumptions. First, we will set d to be

$$d(x, g_\phi(f_\theta(x))) = (x - g_\phi(f_\theta(x)))^\top (x - g_\phi(f_\theta(x))), \tag{73}$$

so that the expected distortion is the mean-squared error. This is not always a good choice. For instance, if you are trying to recover hidden structure in images, the mean-squared error is a horrible choice of distortion measure. Sometimes an image that has similar mean-squared error relative to the original as another can look, perceptually, far better. But the benefit of mean-squared error is that you often get simple algorithms with analytic or nearly analytic results. The second approximation we'll make is that g and f are both linear maps, so that

$$g_\phi(f_\theta(x)) = Px, \tag{74}$$

where P is a low-rank matrix. The combination of these two assumptions will allow us to recover a popular algorithm called Principal Components Analysis (PCA).

The intuition behind the algorithm is so simple that I will first deal with a minimal example and build from there. Let's imagine that x is a d-dimensional vector whose independent components x_i have variance σ_i^2 and mean 0. What should P be? A good guess is that P should cause us to carry over, with complete fidelity, certain components of the vector. The other components of the vector are most likely to be 0, and so we would guess 0 for those components (the decoding). If the components of x were not independent, a different solution would be warranted, as we'll see, but for now, this amounts to us choosing P to be a "projection matrix". A projection matrix is a diagonal matrix that looks like the identity matrix, but with 0's instead of 1's in certain entries. Our goal is to figure out which entries should have 0's instead of 1's.

(You might be wondering how big of an assumption it is to say that the data is mean zero. That is not much of an assumption at all! I can *pre-process* the data so that it has zero mean without loss of generality.)

Let's call the set of vector elements that we reproduce exactly $\mathcal{X}_{in}$ and the set of vector elements that we guess 0 for $\mathcal{X}_{out}$. A moment's thought gives us

$$\langle d(x, Px) \rangle = \sum_{i \in \mathcal{X}_{out}} \langle x_i^2 \rangle = \sum_{i \in \mathcal{X}_{out}} \sigma_i^2. \tag{75}$$

We'd like to make this as small as possible, given the constraint that there are only k elements transmitted with perfect fidelity, $|\mathcal{X}_{in}| \leq k$. But that's easy: we merely choose the $d - k$ elements with the smallest variance and place them in the out group. In other words, we choose to keep track of only those components of x with the largest variance. This, in essence, is PCA.

Now let's add some complications and see what happens. Imagine that, instead of viewing our data in the original space, we rotate the original space so that our elements are all mixed up. What should we do? We still want to pull out just those components with the highest variance. So, we rotate our data back into its original frame, pull out those components with the highest variance, and rotate the processed data back into the new frame afterwards.

That is the geometric intuition. That sounds like it'd be hard to do, but really what we're doing is deciding how to rotate back and forth using the eigenvectors of a matrix that describes the covariance of the data. It turns out that the analytic procedure for PCA– which is no more than what we've described above– is as follows:

- Zero-mean your data so that $\langle x \rangle = \vec{0}$.
- Calculate the covariance matrix $\langle xx^\top \rangle$.
- Calculate the eigenvectors (the rotated axis) and corresponding eigenvalues σ_i^2 of the covariance matrix and sort the eigenvectors based on eigenvalues, so that high-value eigenvalues are preferred.
- Choose a number k (arbitrary for now, one second) and take the first k sorted eigenvectors.

The only thing left to do is to visualize these eigenvectors. They might correspond to representations of people's faces, or of handwritten digits, or something more abstract. But these eigenvectors are the salient low-dimensional features of your data that you'll want to keep track of if you only get k components.

The final piece of the puzzle: how do we choose k? There are a number of rules for this, and not one of them has completely taken over. The easiest rule is the "Elbow Rule", in which you plot the unexplained variance ($\sum_{i \in \mathcal{X}_{in}} \sigma_i^2$) as a function of k. These graphs tend to look like an elbow. It seems reasonable to choose k to be the location of the kink, so that you don't run into the problem of diminishing returns.

Alternatively, you might actually have a noisy channel that you want to send things down through. In that case, k should be adapted to the noisy channel. That leads us to a perspective on dimensionality reduction related to rate-distortion theory, which we won't go through here, but essentially the constraint on k is replaced with a constraint on rate.

There are plenty of advancements in dimensionality reduction since PCA, and they essentially all rely on better choices of f and g, and perhaps d. An autoencoder– a neural network with a forced bottleneck– can substitute for f and g. Sometimes, those autoencoders can be evaluated using the rate-distortion framework, if what one desires is to eventually send the latent variable somewhere!