Data Science for Biodiversity

Fundamentals and Applications

Timothée Poisot

2023-08-13

Table of contents

Preface 1

1. Introduction 3

1.1. Core concepts in data science 4
  1.1.1. EDA 4
  1.1.2. Clustering and regression 4
  1.1.3. Supervised and unsupervised 4
  1.1.4. Training, testing, and validation 4
  1.1.5. Transformations and feature engineering 4
1.2. An overview of the content 4
1.3. How to read this book 4
1.4. Some rules about this book 4
  1.4.1. No code 4
  1.4.2. No simulated data 5
  1.4.3. No `iris` 5

I. Getting started 7

2. The *k*-means algorithm 9

2.1. A digression: which birds are red? 9
2.2. The problem: classifying pixels from an image 11
2.3. The theory behind *k*-means clustering 17
  2.3.1. Overview of the algorithms 17
2.4. Identification of the optimal number of clusters 17
2.5. Application: optimal clustering of the satellite image data . 17

2.6. Alternatives and improvements 17

3. Gradient descent 21
  3.1. A digression: what is a trained model? 21
  3.2. The problem: how many interactions in a food web? 21
  3.3. Gradient descent 22
    3.3.1. Defining the loss function 23
    3.3.2. Calculating the gradient 25
    3.3.3. Descending the gradient 26
    3.3.4. A note on the learning rate 27
  3.4. Application: how many links are in a food web? 28
  3.5. Notes on regularization 28

II. Exploring and preparing data 29

4. Testing, training, validating 31
  4.1. How can we split a dataset? 31
    4.1.1. Training 31
    4.1.2. Testing 33
    4.1.3. Validating 33
  4.2. Strategies to split data 33

5. Data leakage 35
  5.1. Consequences of data leakage 35
  5.2. Sources of data leakage 36
  5.3. Avoiding data leakage 36

6. Summary 37

References 39

Preface

Data science is now an established methodology to study biodiversity, and this is a problem.

This may be an opportunity when it comes to advancing our knowledge of biodiversity, and in particular when it comes to translating this knowledge into action (Tuia et al. 2022); but make no mistake, this is a problem for us, biodiversity scientists, as we suddenly need to develop competences in an entirely new field. And as luck would have it, there are easier fields to master than data science. The point of this book, therefore, is to provide an introduction to fundamental concepts in data science, from the perspective of a biodiversity scientist, by using examples corresponding to real-world use-cases of these techniques.

But what do we mean by *data science*? Most science, after all, relies on data in some capacity. What falls under the umbrella of data science is, in short, embracing in equal measure quantitative skills (mathematics, machine learning, statistics), programming, and domain expertise, in order to solve well-defined problems. A core tenet of data science is that, when using it, we seek to "deliver actionable insights", which is MBA-speak for "figuring out what to do next". One of the ways in which this occurs is by letting the data speak, after they have been, of course, properly cleaned and transformed and engineered beyond recognition. This entire process is driven by (or subject to, even) domain knowledge. There is no such thing as data science, at least not in a vacuum: there is data science as a methodology applied to a specific domain.

Preface

Before we embark into a journey of discovery on the applications of data science to biodiversity, allow me to let you in on a little secret: data *science* is a little bit of a misnomer.

To understand why, it helps to think of science (the application of the scientific method, that is) as cooking. There are general techniques one must master, and specific steps and cultural specifics, and there is a final product. When writing this preface, I turned to my shelf of cookbooks, and picked my two favorites: Robuchon's *The Complete Robuchon* (a no-nonsense list of hundreds of recipes with no place for improvisation), and Bianco's *Pizza, Pasta, and Other Food I Like* (a short volume with very few pizza and pasta, and wonderful discussions about the importance of humility, creativity, and generosity). Data science, if it were cooking, would feel a lot like the second. Deviation from the rules (they are mostly recommendations, in fact) is often justifiable if you feel like it. But this improvisation requires good skills, a clear mental map of the problem, and a library of patterns that you can draw from.

This book will not get you here. But it will speed up the process, by framing the practice of data science as a natural way to conduct research on biodiversity.

1. Introduction

This book started as a collection of notes from several classes I gave in the Department of Biological Sciences at the Université de Montréal, as well as a few workshops I ran with the Québec Centre for Biodiversity Sciences. In teaching data synthesis, data science, and machine learning to biology students, I realized that the field was missing a stepping stone to proficiency. There are excellent manuals covering the mathematics of data science and machine learning **REFS**; there are many good papers giving overviews of some applications of data science to biological problems **REFS**; and there are, of course, thousands of tutorials about how to write code. But one thing that students commonly called for was an attempt to tie concepts together. This is this attempt.

1.1. Core concepts in data science

1.1.1. EDA

1.1.2. Clustering and regression

1.1.3. Supervised and unsupervised

1.1.4. Training, testing, and validation

1.1.5. Transformations and feature engineering

1.2. An overview of the content

1.3. How to read this book

1.4. Some rules about this book

When I started aggregating these notes, I decided on a series of three rules. No code, no simulated data, and no `iris`. In this section, I will go through *why* I decided to adopt these rules, and how it can change the way you interact with the book.

1.4.1. No code

This is, maybe, the most surprising rule, because data science *is* programming (in a sense). But sometimes there is so much focus on programming that we lose track of the other, important aspects of the practice of data science: abstractions, relationship with data, and domain knowledge.

This book *did* involve a lot of code. Specifically, this book was written using *Julia* (Bezanson et al. 2017), and every figure is generated by a notebook, and they are available online, and used in the classroom. But code is *not* a universal language, and unless you are familiar with the language, code can obfuscate. I had no intention to write a *Julia* book (or an *R* book, or a *Python* book). The point is to think about data science applied to ecological research, and I felt like it would be more inclusive to do this in a language agnostic way.

And finally, code rots. Code with more dependencies rots faster. It take a single change in the API of a package to break the examples, and then you are left with a very expensive doorstop[1].

1.4.2. No simulated data

1.4.3. No `iris`

---

[1] Which will not happen with this book! Ever! Not because it will not become obsolete, but because it's online, and free, two sever impediments to being an expensive doorstop.

Part I.

Getting started

2. The *k*-means algorithm

As we mentioned in the introduction, a core idea of data science is that things that look the same (in that, when described with data, they resemble one another) are likely to be the same. Although this sounds like a simplifying assumption, this can provide the basis for a very powerful technique in which we *create* groups in data that have no labels. This task is called unsupervised clustering: we seek to add a *label* to each observation, in order to form groups, and the data we work from do *not* have a label that we can use to train a model.

2.1. A digression: which birds are red?

Before diving in, it is a good idea to ponder a simple case. We can divide everything in just two categories: things with red feathers, and things without red feathers. An example of a thing with red feathers is the Northern Cardinal (*Cardinalis cardinalis*), and things without red feathers are the iMac G3, Haydn's string quartets, and of course the Northern Cardinal (*Cardinalis cardinalis*).

See, biodiversity data science is complicated, because it tends to rely on the assumption that we can categorize the natural world, and the natural world (mostly in response to natural selection) comes up with ways to be, well, diverse. In the Northern Cardinal, this is shown in males having red feathers, and females having mostly brown feathers. Before moving forward, we need to consider ways to solve this issue, as this issue will come up *all the time.*

The first mistake we have made is that the scope of objects we want to classify, which we will describe as the "domain" of our classification, is much too broad: there are few legitimate applications where we will have a dataset with Northern Cardinals, iMac G3s, and Haydn's string quartets. Picking a reasonable universe of classes would have solved our problem a little. For example, among the things that do not have red feathers are the Mourning Dove, the Kentucky Warbler, and the House Sparrow.

The second mistake that we have made is improperly defining our classes; bird species exhibit sexual dimorphism (not in an interesting way, like wrasses, but you let's still give them some credit for trying). Assuming that there is such a thing as a Northern Cardinal is not necessarily a reasonable assumption! And yet, the assumption that a single label is a valid representation of non-monomorphic populations is a surprisingly common one, with actual consequences for the performance of image classification algorithms (Luccioni and Rolnick 2023). This assumption reveals a lot about our biases: male specimens are over-represented in museum collections, for example (Cooper et al. 2019). In a lot of species, we would need to split the taxonomic unit into multiple groups in order to adequately describe them.

The third mistake we have made is using predictors that are too vague. The "presence of red feathers" is not a predictor that can easily discriminate between the Northen Cardinal (yes for males, sometimes for females), the House Finch (a little for males, no for females), and the Red-Winged Black Bird (a little for males, no for females). In fact, it cannot really capture the difference between red feathers for the male House Finch (head and breast) and the male Red Winged Black Bird (wings, as the name suggests).

The final mistake we have made is in assuming that "red" is relevant as a predictor. In a wonderful paper, Cooney et al. (2022) have converted the color of birds into a bird-relevant colorimetric space, revealing a clear latitudinal trend in the ways bird colors, as perceived by other birds, are distributed. This analysis, incidentally, splits all species into males and females. The use of a color space that accounts for the way colors are perceived is a fantastic example of why data science puts domain knowledge front and center.

Deciding which variables are going to be accounted for, how the labels will be defined, and what is considered to be within or outside the scope of the classification problem is *difficult*. It requires domain knowledge (you must know a few things about birds in order to establish criteria to classify birds), and knowledge of how the classification methods operate (in order to have just the right amount of overlap between features in order to provide meaningful estimates of distance).

2.2. The problem: classifying pixels from an image

Throughout this chapter, we will work on a single image – we may initially balk at the idea that an image is data, but it is! Specifically, an image is a series of instances (the pixels), each described by their position in a multidimensional colorimetric space. Greyscale images have one dimension, and images in color will have three: their red, green, and blue channels. Not only are images data, this specific dataset is going to be far larger than many of the datasets we will work on in practice: the number of pixels we work with is given by the product of the width and height of the image!

In fact, we are going to use an image with a lot more dimensions: the data in this chapter are coming from a Landsat 9 image Vermote et al. (2016), for which we have access to 7 different bands (the full data product has more bands, but we will not use them all.

Band number	Information
1	Aerosol
2	Visible blue
3	Visible red

Band number	Information
4	Visible green
5	Near-infrared (NIR)
6	Short wavelength IR (SWIR 1)
7	SWIR 2

From these channels, we can reconstruct an approximation of what the landscape looked like (by using the red, green, and blue channels) – this information is presented in Figure 2.1 . Or is it? If we were to invent a time machine, and go stand directly under Landsat 9 at the exact center of this scene, and look around, what would we see? We would see colors, and they would admit a representation as a three-dimensional vector of red, green, and blue. But we would see so much more than that! And even if we were to stand within a pixel, we would see a *lot* of colors. And texture. And depth. We would see something entirely different from this map; and we would be able to draw a lot more inferences about our surroundings than what is possible by knowing the average color of a 30x30 meters pixel.

But just like we can get more information that Landsat 9, so to can Landsat 9 out-sense us when it comes to getting information. In the same way that we can extract a natural color composite out of the different channels, we can extract a fake color one to highlight differences in the landscape; in Figure 2.2, we show such a fake color composite, that is particularly efficient at drawing our attention to the location of water in this area.

Both Figure 2.2 and Figure 2.1 represent the same physical place at the same moment in time; but through them, we are looking at this place with very different purposes. This is not an idle observation, but a core notion in data science: what we measure defines what we can see. In order to tell something meaningful about this place, we need to look at it in the “right” way.

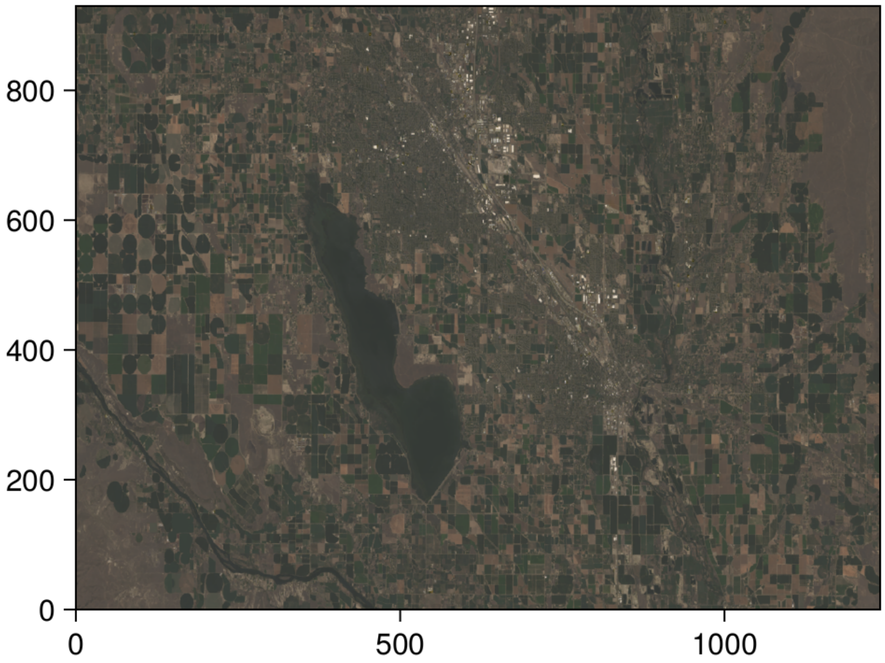

Figure 2.1.: The Landsat 9 data are combined into the “Natural Color” image, in which the red, green, and blue bands are mapped to their respective channels. This looks eerily like the way we perceive the landscape. Note how little difference there is between the large body of water and the surrounding crops. This emphasizes that the combination of channels we use will limit our ability to separate the pixels into groups.

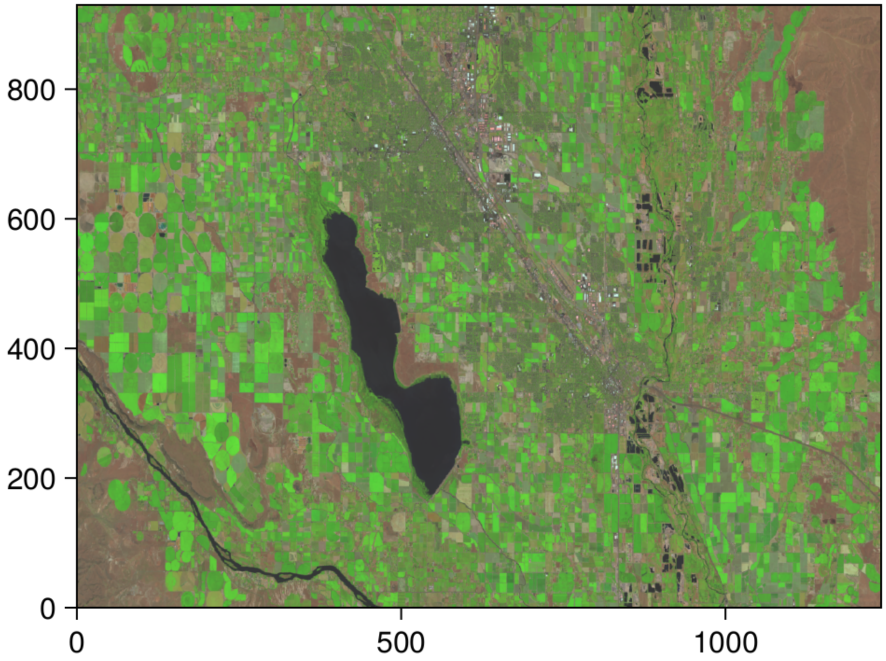

Figure 2.2.: The same landscape as above is now presented in a fake color composite, where SWIR is mapped to the red channel, NIR to the green channel, and red to the blue channel. This highlights different values in the landscape, but is no more or less "real" than the true color composite. It is a visualization of the data, that represents different choices, questions, and assumptions. Compared to the Natural Color composite, this version of the data highlights the water, areas with vegetation, and more arid areas.

So far, we have looked at this area by combining the raw data. Depending on the question we have in mind, they may not be the *right* data. In fact, they may not hold information that is relevant to our question *at all*; or worse, they can hold more noise than signal. Looking at Figure 2.1, we might wonder, “where are the fields?”. And based on our knowledge of what plants do, we can start thinking about this question in a different way. Specifically, “is there a series of features of fields that are not shared by non-fields?”. But this a complicated question to answer, and so we can simplify this by asking, “how can I combine data from the image to know if there is a plant?”.

One way to do this is to calculate the normalized difference vegetation index, or NDVI (Kennedy and Burbach 2020). NDVI is derived from the band data (we will see how in a minute), and is an adequate heuristic to make a difference between vegetation, barren soil, and water. Because we are specifically thinking about fields, we can also consider the NDWI (water) and NDMI (moisture) dimensions: taken together, these information will represent every pixel in a three-dimensional space, telling us whether there are plants (NDVI), whether they are stressed (NDMI), and whether this pixel is a water body (NDWI).

Because there are a few guidelines (educated guesses, in truth, and the jury is still out on the “educated” part) about the values, we can look at the relationship between the NDVI and NDMI data Figure 2.3. For example, NDMI values around -0.1 (note how there is a strong cluster of points here) are low-canopy cover with low water stress; NDVI values from 0.2 to 0.5 are good candidates for moderately dense crops.

By picking these three values, instead of simply looking at the clustering of all the bands in the raw data, we are starting to refine what the algorithm see, through the lens of what we know is important about the system.

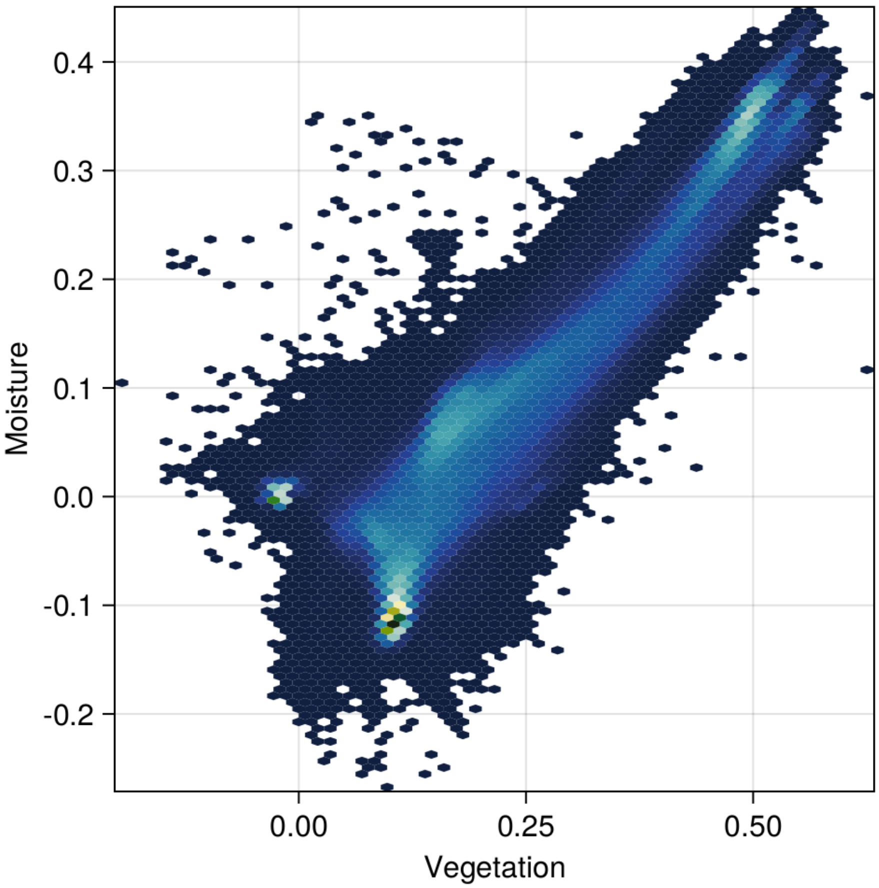

Figure 2.3.: The pixels acquired from Landsat 8 exist in a space with many different dimensions (one for each band). Because we are interested in a landscape classification based on water/vegetation data, we use the NDVI, NDMI, and NDWI combinations of bands. These are *derived* data, and represent an instance of feature engineering: we have derived these values from the raw data.

2.3. The theory behind *k*-means clustering

In order to understand the theory underlying *k*-means, we will work backwards from its output. As a method for unsupervised clustering, *k*-means will return a vector of *class memberships*, which is to say, a list that maps each observation (pixel, in our case) to a class (tentatively, a cohesive landscape unit). What this means is that *k*-means is a transformation, taking as its input a vector with three dimensions (red, green, blue), and returning a scalar (an integer, even!), giving the class to which this pixel belongs. These are the input and output of our blackbox, and now we can start figuring out its internals.

2.3.1. Overview of the algorithms

2.4. Identification of the optimal number of clusters

2.5. Application: optimal clustering of the satellite image data

2.6. Alternatives and improvements

EM

k-median

k-medoids

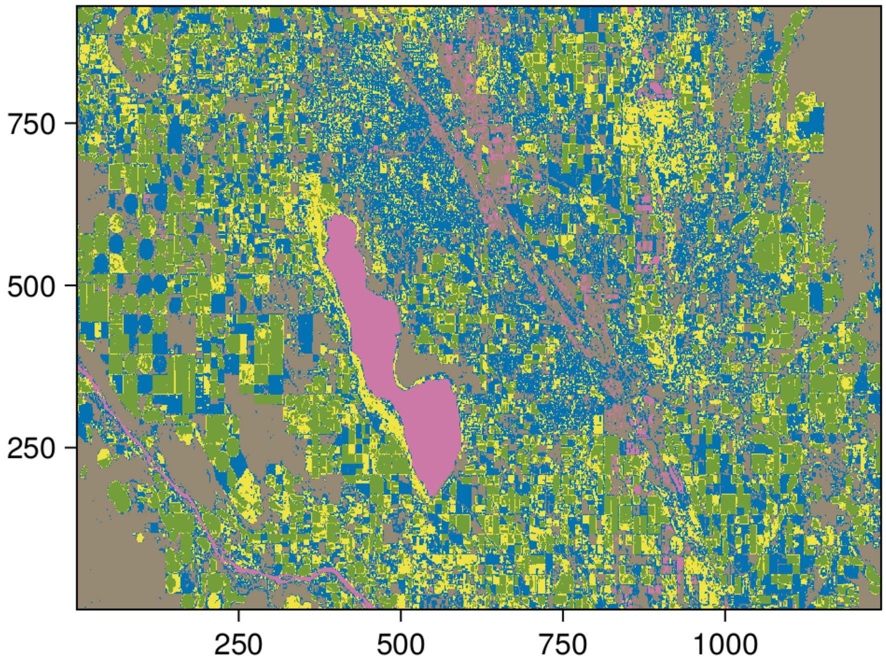

Figure 2.4.: After iterating the k-means algorithm, we obtain a classification for every pixel in the landscape. This classification is based on the values of NDVI, NDMI, and NDWI indices, and therefore groups pixels based on a specific hypothesis. This clustering was produced using $k = 5$, *i.e.* we want to see what the landscape would look like when divided into five categories.

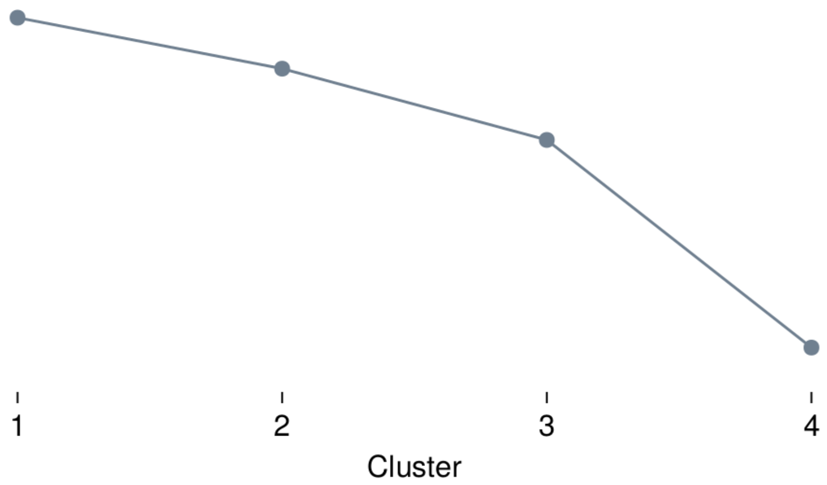

Figure 2.5.: Number of pixels assigned to each class in the final landscape classification. In most cases, *k*-means will create clusters with the same number of points in them. This may be an issue, or this may be a way to ensure that whatever classes are produced will be balanced in terms of their representation.

3. Gradient descent

As we progress into this book, the process of delivering a trained model is going to become increasingly complex. In Chapter 2, we work with a model that did not really require training (but did require to pick the best hyper-parameter). In this chapter, we will only increase complexity very slightly, by considering how we can train a model when we have a reference dataset to compare to.

Doing do will require to introduce several new concepts, and so the “correct” way to read this chapter is to focus on the high-level process. The problem we will try to solve (which is introduced in **?@sec-gradientdescent-problem**) is very simple; in fact, the empirical data looks more fake than many simulated datasets!

3.1. A digression: what is a trained model?

3.2. The problem: how many interactions in a food web?

One of the earliest observation that ecologists made about food webs is that when there are more species, there are more interactions. A remarkably insightful crowd, food web ecologists. Nevertheless, it turns out that this apparently simple question had received a few different answers over the years. In

The initial model was proposed by Cohen and Briand (1984): the number of interactions L scales linearly with the number of species S. After all, we can assume that when averaging over many consumers, there will be an average diversity of resources they consume, and so the number of interactions could be expressed as $L \approx b \times S$.

Not so fast, said Martinez (1992). When we start looking a food webs with more species, the increase of L with regards to S is superlinear. Thinking in ecological terms, maybe we can argue that consumers are flexible, and that instead of sampling a set number of resources, they will sample a set proportion of the number of consumer-resource combinations (of which there are S^2). In this interpretation, $L \approx b \times S^2$.

But the square term can be relaxed; and there is no reason not to assume a power law, with $L \approx b \times S^a$. This last formulation has long been accepted as the most workable one, because it is possible to approximate values of its parameters using other ecological processes (Brose et al. 2004).

The "reality" (*i.e.* the relationship between S and L that correctly accounts for ecological constraints, and fit the data as closely as possible) is a little bit different than this formula (MacDonald, Banville, and Poisot 2020). But for the purpose of this chapter, figuring out the values of a and b from empirical data is a very instructive exercise.

3.3. Gradient descent

Gradient descent is built around a remarkably simple intuition: knowing the formula that gives rise to our prediction, and the value of the error we made for each point, we can take the derivative of the error with regards to each parameter, and this tells us how much this parameter contributed to the error. Because we are taking the derivative, we can futher know whether to increase, or decrease, the value of the parameter in order to make a smaller error next time.

In this section, we will use linear regression as an example, because it is the model we have decided to use when exploring our ecological problem in **?@sec-gradientdescent-problem**, and because it is suitably simple to keep track of everything when writing down the gradient by hand.

Before we start assembling the different pieces, we need to decide what our model is. We have settled on a linear model, which will have the form $\hat{y} = m \times x + b$. The little hat on $\hat{y}$ indicates that this is a prediction. The input of this model is x, and its parameters are m (the slope) and b (the intercept). Using the notation we adopted in **?@sec-gradientdescent-problem**, this would be $\hat{l} = a \times s + b$, with $l = \text{log}L$ and $s = \text{log}S$.

3.3.1. Defining the loss function

The loss function is an important concept for anyone attempting to compare predictions to outcomes: it quantifies how far away an ensemble of predictions is from a benchmark of known cases. There are many loss functions we can use, and we will indeed use a few different ones in this book. But for now, we will start with a very general understanding of what these functions *do*.

Think of prediction as throwing a series of ten darts on ten different boards. In this case, we know what the correct outcome is (the center of the board, I assume, although I can be mistaken since I have only played darts once, and lost). A cost function would be any mathematical function that compares the position of each dart on each board, the position of the correct event, and returns a score that informs us about how poorly our prediction lines up with the reality.

In the above example, you may be tempted to say that we can take the Euclidean distance of each dart to the center of each board, in order to know, for each point, how far away we landed. Because there are several boards, and because we may want to vary the number of boards while still retaining the ability to compare our performances, we would then take the average of these measures.

We will note the position of our dart as being $\hat{y}$, the position of the center as being y (we will call this the *ground truth*), and the number of attempts n, and so we can write our loss function as

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.1}$$

This loss function is usually called the MSE (Mean Standard Error), or L2 loss, or the quadratic loss, because the paths to machine learning terminology are many. This is a good example of a loss function for regression (and we will discuss loss functions for classification later in this book). There are alternative loss functions to use for regression problems in Table 3.1.

Table 3.1.: List of common loss functions for regression problems

Measure	Expression	Remarks
Mean Squared Error (MSE, L2)	$\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2$	Large errors are (proportionally) more penalized because of the squaring
Mean Absolute Error (MAE, L1)	$\frac{1}{n}\sum_{i=1}^{n}\lVert y_i - \hat{y}_i\rVert$	Error measured in the units of the response variable
Root Mean Square Error (RMSE)	$\sqrt{\text{MSE}}$	Error measured in the units of the response variable
Mean Bias Error	$\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)$	Errors *can* cancel out, but this can be used as a measure of positive/negative bias

Throughout this chapter, we will use the L2 loss (Equation 3.1), because it has *really* nice properties when it comes to taking derivatives, which we will do a lot of. In the case of a linear model, we can rewrite Equation 3.1 as

$$f = \frac{1}{n}\sum\left(y_i - m \times x_i - b\right)^2 \tag{3.2}$$

There is an important change in Equation 3.2: we have replaced the prediction $\hat{y}_i$ with a term that is a function of the predictor x_i and the model parameters: this means that we can calculate the value of the loss as a function of a pair of values (x_i, y_i), and the model parameters.

3.3.2. Calculating the gradient

With the loss function corresponding to our problem in hands (Equation 3.2), we can calculate the gradient. Given a function that is scalar-valued (it returns a single value), taking several variables, that is differentiable, the gradient of this function is a vector-valued (it returns a vector) function; when evaluated at a specific point, this vectors indicates both the direction and the rate of fastest increase, which is to say the direction in which the function increases away from the point, and how fast it moves.

We can re-state this definition using the terms of the problem we want to solve. At a point $p = [m \quad b]^\top$, the gradient ∇f of f is given by:

$$\nabla f\left(p\right) = \begin{bmatrix}\frac{\partial f}{\partial m}(p)\\ \frac{\partial f}{\partial b}(p)\end{bmatrix}\,. \tag{3.3}$$

This indicates how changes in m and b will *increase* the error. In order to have a more explicit formulation, all we have to do is figure out an expression for both of the partial derivatives. In practice, we can let auto-differentiation software calculate the gradient for us (Innes 2018); these packages are now advanced enough that they can take the gradient of code directly.

Solving $(\partial f/\partial m)(p)$ and $(\partial f/\partial c)(p)$ is easy enough:

$$\nabla f\left(p\right) = \begin{bmatrix} -\frac{2}{n}\sum\left[x_i \times \left(y_i - m \times x_i - b\right)\right] \\ -\frac{2}{n}\sum\left(y_i - m \times x_i - b\right) \end{bmatrix} . \tag{3.4}$$

Note that both of these partial derivatives have a term in $2n^{-1}$. Getting rid of the 2 in front is very straightforward! We can modify Equation 3.2 to divide by $2n$ instead of n. This modified loss function retains the important characteristics: it increases when the prediction gets worse, and it allows comparing the loss with different numbers of points. As with many steps in the model training process, it is important to think about *why* we are doing certain things, as this can enable us to make some slight changes to facilitate the analysis.

With the gradient written down in Equation 3.4, we can now think about what it means to *descend* the gradient.

3.3.3. Descending the gradient

Recall from **TK** that the gradient measures how far we *increase* the function of which we are taking the gradient. Therefore, it measures how much each parameter contributes to the loss value. Our working definition for a trained model is “one that has little loss”, and so in an ideal world, we could find a point p for which the gradient is as small as feasible.

Because the gradient measures how far away we increase error, and intuitive way to use it is to take steps in the *opposite* direction. In other words, we can update the value of our parameters using $p := p - \nabla f(p)$, meaning that we substract from the parameter values their contribution to the overall error in the predictions.

But, as we will discuss further in **?@sec-gradientdescent-learningrate**, there is such a thing as "too much learning". For this reason, we will usually not move the entire way, and introduce a term to regulate how much of the way we actually want to descend the gradient. Our actual scheme to update the parameters is

$$p := p - \eta \times \nabla f(p)\,.$$

This formula can be *iterated*: with each successive iteration, it will get us closer to the optimal value of p, which is to say the combination of m and b that minimizes the loss.

3.3.4. A note on the learning rate

The error we can make on the first iteration will depend on the value of our initial pick of parameters. If we are *way off*, especially if we did not rescale our predictors and responses, this error can get very large. And if we make a very large error, we will have a very large gradient, and we will end up making very big steps when we update the parameter values. There is a real risk to end up over-compensating, and correcting the parameters too much.

In order to protect against this, in reality, we update the gradient only a little, where the value of "a little" is determined by an hyper-parameter called the *learning rate*, which we noted η. This value will be very small (much less than one). Picking the correct learning rate is not simply a way to ensure that we get correct results (though that is always a nice bonus), but can be a way to ensure that we get results *at all*. The representation of numbers in a computer's memory is tricky, and it is possible to create an overflow: a number so large it does not fit within 64 (or 32, or 16, or however many we are using) bits of memory.

The conservative solution of using the smallest possible learning rate is not really effective, either. If we almost do not update our parameters at every epoch, then we will take almost forever to converge on the correct parameters. Figuring out the learning rate is an example of hyper-parameter tuning, which we will get back to later in this book.

3.4. Application: how many links are in a food web?

3.5. Notes on regularization

Part II.

Exploring and preparing data

4. Testing, training, validating

In Chapter 2, we were very lucky. Because we applied an unsupervised method, we didn't really have a target to compare to the output. Whatever classification we got, we had to live with it. It was incredibly freeing. Sadly, in most applications, we will have to compare our predictions to data, and data are incredibly vexatious. In this chapter, we will develop intuitions on the notions of training, testing, and validation; we will further think about data leakage, why it is somehow worse than it sounds, and how to protect against it.

In a sense, we started thinking about these concepts in **?@sec-gradientdescent**; specifically, we came up with a way to optimize the parameters of our model (*i.e.* of *training* our model) based on a series of empirical observations, and a criteria for what a "good fit" is. We further appraised the performance of our model by measuring the loss (our measure of how good the fit is) on a dataset that was not accessible during training.

4.1. How can we split a dataset?

4.1.1. Training

In data science (in applied machine learning in particular), we do *fit* models. We *train* them. This is an important difference: training is an iterative process, that we can repeat, optimize, and tweak. The outcome of training and the outcome of fitting are essentially the same (a model that is parameterized to work as well as possible on a given dataset), but it is good practice to adopt the language of a field, and the language of data science emphasizes the different practices in model training.

Training, to provide a general definition, is the action of modifying the parameters of a model, based on knowledge of the data, and the error that results from using the current parameter values. In **?@sec-gradientdescent**, for example, we will see how to train a linear model using the technique of gradient descent. Our focus in this chapter is not on the methods we use for training, but on the data that are required to train a model.

Training a model is a process akin to rote learning: we will present the same input, and the same expected responses, many times over, and we will find ways for the error on each response to decrease.

In order to initiate this process, we need an untrained model. Untrained, in this context, refers to a model that has not been trained *on the specific problem* we are addressing; the model may have been trained on a different problem (for example, we want to predict the distribution of a species based on a GLM trained on a phylogenetically related species). It is important to note that by "training the model", what we really mean is "change the structure of the parameters until the output looks right". For example, assuming a simple linear model like $c(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2$, training this model would lead to changes in the values of β, but not to the consideration of a new model $c(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2$. Comparing models is the point of validation, which we will address later on.

need for instances

need for responses

4.1.2. Testing

4.1.3. Validating

4.2. Strategies to split data

5. Data leakage

Data leakage is a concept that is, surprisingly, grosser than it sounds. The purpose of this section is to put the fear of data leakage in you, because it can, and most assuredly *will*, lead to bad models, which is to say (as we discussed in **?@sec-gradientdescent-trainedmodel**), models that do not adequately represent the underlying data, in part because we have built-in some biases into them. In turn, this can eventually lead to decreased explainability of the models, which erodes trust in their predictions (Amarasinghe et al. 2023). As illustrated by Stock, Gregr, and Chan (2023), a large number of ecological applications of machine learning are particularly susceptible to data leakage, meaning that this should be a core point of concern for us.

5.1. Consequences of data leakage

We take data leakage so seriously because it is one of the top ten mistakes in applied machine learning (Nisbet et al. 2018). Data leakage happens information "leaks" from the training conditions to the evaluation conditions. In other words, when the model is evaluated after mistakenly being fed information that would not be available in real-life situations. Note that this definition of leakage is different from another notion, namely the loss of data availability over time (Peterson et al. 2018).

It is worth stopping for a moment to consider what these "real-life situations" are, and how they differ from the training of the model. Most of this difference can be summarized by the fact that when we are *applying* a model, we can start from the model *only*. Which is to say, the data that have been used for the training and validation of the model may have been lost, without changing the applicability of the model: it works on entirely new data.

Because this is the behavior we want to simulate with a validation dataset, it is very important to fully disconnect the validation data from the rest of the data. We can illustrate this with an example. Let's say we want to work on a time series of population size, such as provided by the *BioTIME* project (Dornelas et al. 2018). One naïve approach would be to split this the time series at random into three datasets. We can use one to train the models, one to test these models, and a last one for validation.

Congratulations! We have created data leakage! Because we are splitting our timeseries at random, the model will likely have been trained using data that date from *after* the start of the validation dataset. In other words: our model can peek into the future. This is highly unlikely to happen in practice, due to the laws of physics. A strategy that would prevent leakage would have been to pick a cut-off date to define the validation dataset, and then to decide how to deal with the training and testing sets.

5.2. Sources of data leakage

instances

features

5.3. Avoiding data leakage

learn/predict separation (Kaufman, Rosset, and Perlich 2011)

6. Summary

In summary, this book has no content whatsoever.

References

Amarasinghe, Kasun, Kit T. Rodolfa, Hemank Lamba, and Rayid Ghani. 2023. "Explainable Machine Learning for Public Policy: Use Cases, Gaps, and Research Directions." *Data & Policy* 5. https://doi.org/10.1017/dap.2023.2.

Bezanson, Jeff, Alan Edelman, Stefan Karpinski, and Viral B. Shah. 2017. "Julia: A Fresh Approach to Numerical Computing." *SIAM Review* 59 (1): 65–98. https://doi.org/10.1137/141000671.

Brose, Ulrich, Annette Ostling, Kateri Harrison, and Neo D. Martinez. 2004. "Unified Spatial Scaling of Species and Their Trophic Interactions." *Nature* 428 (6979): 167–71. https://doi.org/10.1038/nature02297.

Cohen, J. E., and F. Briand. 1984. "Trophic Links of Community Food Webs." *Proc Natl Acad Sci U S A* 81 (13): 4105–9.

Cooney, Christopher R., Yichen He, Zoë K. Varley, Lara O. Nouri, Christopher J. A. Moody, Michael D. Jardine, András Liker, Tamás Székely, and Gavin H. Thomas. 2022. "Latitudinal Gradients in Avian Colourfulness." *Nature Ecology & Evolution* 6 (5): 622–29. https://doi.org/10.1038/s41559-022-01714-1.

Cooper, Natalie, Alexander L. Bond, Joshua L. Davis, Roberto Portela Miguez, Louise Tomsett, and Kristofer M. Helgen. 2019. "Sex Biases in Bird and Mammal Natural History Collections." *Proceedings of the Royal Society B: Biological Sciences* 286 (1913): 20192025. https://doi.org/10.1098/rspb.2019.2025.

Dornelas, Maria, Laura H. Antão, Faye Moyes, Amanda E. Bates, Anne E. Magurran, Dušan Adam, Asem A. Akhmetzhanova, et al. 2018. "BioTIME: A Database of Biodiversity Time Series for the Anthropocene." *Global Ecology and Biogeography* 27 (7): 760–86. https://doi.org/10.1111/geb.12729.

Innes, Michael. 2018. "Don't Unroll Adjoint: Differentiating SSA-Form Programs." https://doi.org/10.48550/ARXIV.1810.07951.

Kaufman, Shachar, Saharon Rosset, and Claudia Perlich. 2011. "Leakage in Data Mining." *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August. https://doi.org/10.1145/2020408.2020496.

Kennedy, Stephanie, and Mark Burbach. 2020. "Great Plains Ranchers Managing for Vegetation Heterogeneity: A Multiple Case Study." *Great Plains Research* 30 (2): 137–48. https://doi.org/10.1353/gpr.2020.0016.

Luccioni, Alexandra Sasha, and David Rolnick. 2023. "Bugs in the Data: How ImageNet Misrepresents Biodiversity." *Proceedings of the AAAI Conference on Artificial Intelligence* 37 (12): 14382–90. https://doi.org/10.1609/aaai.v37i12.26682.

MacDonald, Arthur Andrew Meahan, Francis Banville, and Timothée Poisot. 2020. "Revisiting the Links-Species Scaling Relationship in Food Webs." *Patterns* 1 (0). https://doi.org/10.1016/j.patter.2020.100079.

Martinez, Néo D. 1992. "Constant Connectance in Community Food Webs." *The American Naturalist* 139 (6): 1208–18. http://www.jstor.org/stable/2462337.

Nisbet, Robert, Gary Miner, Ken Yale, John F. Elder, and Andrew F. Peterson. 2018. *Handbook of Statistical Analysis and Data Mining Applications*. Second edition. London: Academic Press.

Peterson, A. Townsend, Alex Asase, Dora Canhos, Sidnei de Souza, and John Wieczorek. 2018. "Data Leakage and Loss in Biodiversity Informatics." *Biodiversity Data Journal* 6 (November). https://doi.org/10.3897/bdj.6.e26826.

Stock, Andy, Edward J. Gregr, and Kai M. A. Chan. 2023. "Data Leakage Jeopardizes Ecological Applications of Machine Learning." *Nature Ecology & Evolution*, August. https://doi.org/10.1038/s41559-023-02162-1.

Tuia, Devis, Benjamin Kellenberger, Sara Beery, Blair R. Costelloe, Silvia Zuffi, Benjamin Risse, Alexander Mathis, et al. 2022. “Perspectives in Machine Learning for Wildlife Conservation.” *Nature Communications* 13 (1): 792. https://doi.org/10.1038/s41467-022-27980-y.

Vermote, Eric, Chris Justice, Martin Claverie, and Belen Franch. 2016. “Preliminary Analysis of the Performance of the Landsat 8/OLI Land Surface Reflectance Product.” *Remote Sensing of Environment* 185 (November): 46–56. https://doi.org/10.1016/j.rse.2016.04.008.