MACHINE LEARNING AND STATISTICS

an introduction

Samuel S. Watson

Contents

1 Introduction

1.1 Two examples

We will begin by conducting an end-to-end exploration of two toy machine learning problems. Along the way we will develop various computational and statistical ideas.

1.1.1 Regression

Example 1.1.1

Suppose that X is a random variable which represents the number of hours a student studies for an exam, and Y is a random variable which represents their performance on the exam. Suppose that the joint density of X and Y is given by

$$f(x,y) = \frac{3}{4000(3/2)\sqrt{2\pi}} x(20-x) \mathrm{e}^{-\frac{1}{2(3/2)^2}\left(y-2-\frac{1}{50}x(30-x)\right)^2}.$$

Given that a student's number of hours of preparation is known to be x, what is the best prediction for Y (as measured by mean squared error)?

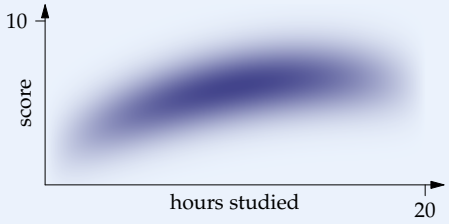


Figure 1.1 The probability density function of the joint distribution of (X, Y), where X is the number of hours of studying and Y is the exam score.

Solution

The best guess of a random variable (as measured by mean squared error) is the expectation of the random variable. Likewise, the best guess of Y given $\{X = x\}$ is the *conditional* expectation of Y given $\{X = x\}$. Inspecting the density function, we recognize that restricting it to a vertical line at position x gives an expression which is proportional to the Gaussian density centered at $2 + \frac{1}{50}x(30-x)$. Therefore,

$$\mathbb{E}[Y \mid X = x] = 2 + \frac{1}{50}x(30-x).$$

A graph of this function is shown in red in Figure 1.2.

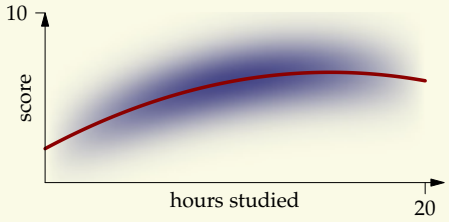


Figure 1.2 The conditional expectation of Y given $\{X = x\}$, as a function of x.

We call $r(x) = \mathbb{E}[Y \mid X = x]$ the **regression** function for the problem of predicting Y given the value of X. Given the distribution of (X, Y), finding the regression function is a probability problem. However, in practice the probability measure is typically only known *empirically*, that is, by way of a collection of independent samples from the measure. For example, to understand the relationship between exam scores and studying hours, we would record the values (X_i, Y_i) for many past exams $i \in \{1, 2, \ldots, n\}$ (see Figure 1.3).

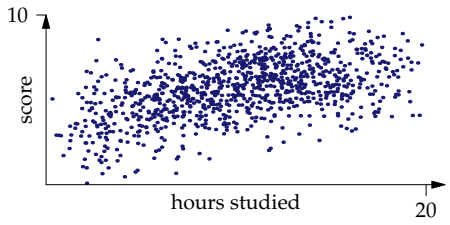


Figure 1.3 One thousand independent samples from the joint distribution of (X, Y).

Exercise 1.1.1

Describe an example of a random experiment where the probability measure may be reasonably inferred without the need for experimentation or data collection. What is the difference between such an experiment and one for which an empirical approach is required?

It is easy to convince oneself that r, and indeed the whole joint distribution of X and Y, is approximately recoverable from the samples shown in Figure 1.3: if we draw a curve roughly through the middle of the point cloud, it would be pretty close to the graph of r. If we shade the region occupied by the point cloud, darker where there are more points and lighter where there are fewer, it will be pretty close to the heat map of the density function. We will of course want to sharpen this intuition into a computable algorithm, but the picture gives us reason to be optimistic.

Exercise 1.1.2

In the context of Figure 1.3, why doesn't it work to approximate the conditional distribution of Y given $\{X = x\}$ using all of the samples which are along the vertical line at position x?

1.1.1.1 Kernel density estimation

The **empirical** distribution of (X, Y) is the probability measure which assigns mass $\frac{1}{n}$ to each of the n observed samples from (X, Y). Exercise 1.1.2 illustrates shortcomings of using the empirical distribution directly for this problem: the mass is not appropriately spread out in the rectangle. It makes more sense to conclude from the presence of a sample at a point (x, y) that the unknown distribution of (X, Y) has some mass *near* (x, y), not necessarily exactly *at* (x, y).

We can capture this idea mathematically by spreading out $\frac{1}{n}$ units of probability mass around each sample (x_i, y_i). We have to choose a function—called the **kernel**—to specify how the mass is spread out. There are many kernel functions in common use; we will base our kernel on the *tricube* function, which is defined by*

$$D(u) = \begin{cases} \frac{70}{81}(1 - |u|^3)^3 & \text{if } |u| < 1 \\ 0 & \text{if } |u| \geq 1. \end{cases}$$

We define $D_\lambda(u) = \frac{1}{\lambda} D\left(\frac{u}{\lambda}\right)$; The parameter $\lambda > 0$ can be tuned to adjust how tightly the measure is concentrated at the center.

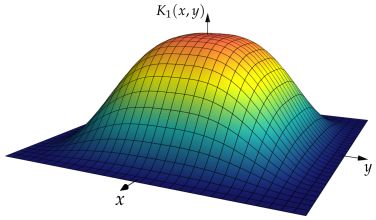


Figure 1.4 The graph of the kernel $K_\lambda(x, y)$ with $\lambda = 1$.

* The factor $\frac{70}{81}$ is there so that D integrates to 1 over the plane.

We define the kernel function K_λ as a product of two copies of D_λ, one for each coordinate (see Figure 1.4)

for a graph):

$$K_\lambda(x,y) = D_\lambda(x)D_\lambda(y).$$

Exercise 1.1.3

Is the probability mass more or less tightly concentrated around the origin when λ is small?

Now, let's place small piles of probability mass with the shape shown in Figure 1.4 at each sample point. The resulting approximation of the joint density $f(x,y)$ is

$$\widehat{f}_\lambda(x,y) = \frac{1}{n}\sum_{i=1}^{n} K_\lambda(x - X_i, y - Y_i).$$

Graphs of $\widehat{f}_\lambda$ for various values of λ are shown below. We can see that the best match is going to be somewhere between the $\lambda \to 0$ and $\lambda \to \infty$ extremes (which are too concentrated and too diffuse, respectively).

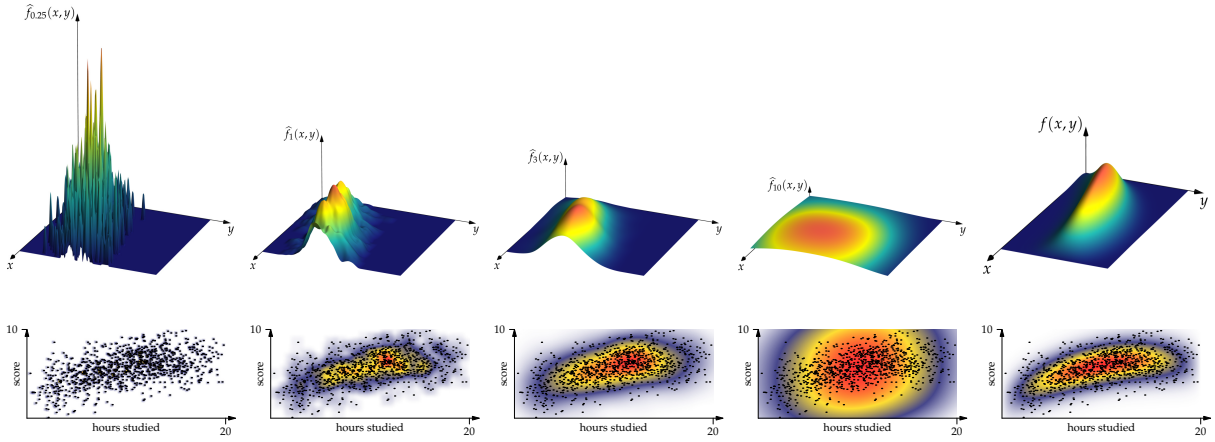


Figure 1.5 The density estimator $\widehat{f}_\lambda$ for various values of λ.

We can take advantage of our knowledge of the density function to determine which λ value works best. We measure the difference between f and $\widehat{f}_\lambda$ by evaluating both at each point in a square grid and finding the sum of the squares of the pointwise differences.*

* Note the use of the unicode square root symbol. You can get this by typing `\sqrt«tab»` in the Julia REPL or in an editor.

Julia

```
using LinearAlgebra, Statistics, Roots, Optim, Plots, Random; Random.seed!(1234)
n = 1000 # number of samples to draw

# the true regression function
r(x) = 2 + 1/50*x*(30-x)
# the true density function
σy = 3/2
f(x,y) = 3/4000 * 1/√(2π*σy^2) * x*(20-x)*exp(-1/(2σy^2)*(y-r(x))^2)

# x values and y values for a grid
xs = 0:1/2^3:20
ys = 0:1/2^3:10
```

```julia
# F(t) is the CDF of X
F(t) = -t^3/4000 + 3t^2/400
# Inverse CDF sampling
function sampleX()
    U = rand()
    find_zero(t->F(t)-U,(0,20),Bisection())
end

function sampleXY(r,σ)
    X = sampleX()
    Y = r(X) + σ*randn()
    (X,Y)
end

samples = [sampleXY(r,σy) for i=1:n]

D(u) = abs(u) < 1 ? 70/81*(1-abs(u)^3)^3 : 0 # tri-cube function
D(λ,u) = 1/λ*D(u/λ) # scaled tri-cube
K(λ,x,y) = D(λ,x) * D(λ,y) # kernel
kde(λ,x,y,samples) = sum(K(λ,x-Xi,y-Yi) for (Xi,Yi) in samples)/length(samples)

# `optimize` takes functions which accept vector arguments, so we treat
# λ as a one-entry vector
L(λ) = sum((f(x,y) - kde(λ,x,y,samples))^2 for x=xs,y=ys)*step(xs)*step(ys)

# minimize L using the BFGS method
λ_best = optimize(λ->L(first(λ)),[1.0],BFGS())
```

We find that the best λ value comes out to about $\lambda = 1.92$.

1.1.1.2 Cross-validation

Even without access to the density function, we can still approximate the optimal λ value. The idea is to reserve one sample point and approximate the density using the other $n-1$ samples. We evaluate this density approximation at the reserved sample point, and we repeat all of these steps for each sample in the data set. If the resulting density value is consistently very small, then our density estimator is being consistently "surprised" by the location of the reserved sample. This suggests that our λ value is too large or too small. This idea is called **cross-validation**.

We can put this idea on firmer mathematical footing. We are aiming to minimize the **loss** (or **error**, or **risk**) of our estimator, which we define to be the integrated squared difference between $\widehat{f}_\lambda$ and the true density f. We can write this integral as*

$$\int (\widehat{f}_\lambda - f)^2 = \int \widehat{f}_\lambda^2 - 2\int \widehat{f}_\lambda f + \int f^2. \tag{1.1.1}$$

* All integrals are over $\mathbb{R}^2$ or $\mathbb{R}$, as appropriate.

The third term does not involve $\widehat{f}$, so minimizing $\int (\widehat{f}_\lambda - f)^2$ is the same as minimizing $\int \widehat{f}_\lambda^2 - 2\int \widehat{f}_\lambda f$, which we will call $J(\lambda)$. Recalling the expectation formula*

$$\mathbb{E}[g(X,Y)] = \int g(x,y)\, f_{X,Y}(x,y)\, \mathrm{d}x\, \mathrm{d}y, \tag{1.1.2}$$

we recognize the second term in (1.1.1) as $-2\mathbb{E}[\widehat{f}_\lambda(X,Y)]$, where (X,Y) is a sample from the true density f which is *independent* of $\widehat{f}_\lambda$. To get samples which are independent of the estimator, we will define $\widehat{f}_\lambda^{(-i)}$ to be

* We derived this formula for non-random functions g, but it can be applied to a random function g if (i) g is independent of (X,Y), and (ii) the expectation is understood to be with respect to (X,Y).

the estimator obtained using the samples other than the ith one. Since the samples (X_i, Y_i) are drawn from the joint density of (X, Y) and are assumed to be independent, we can suggest the approximation

$$\mathbb{E}[\widehat{f}_\lambda(X,Y)] \approx \frac{1}{n}\sum_{i=1}^{n} \widehat{f}_\lambda^{(-i)}(X_i, Y_i),$$

We call

$$\widehat{J}(\lambda) = \int \widehat{f}_\lambda^2 - \frac{2}{n}\sum_{i=1}^{n} \widehat{f}_\lambda^{(-i)}(X_i, Y_i)$$

the **cross-validation loss estimator**. Let's find the value of λ which minimizes $\widehat{J}(\lambda)$ for the present example:

```
"Evaluate the summation Σᵢ f^(-i)(Xᵢ,Yᵢ) in J(λ)'s second term"
function kdeCV(λ,i,samples)
    x,y = samples[i]
    newsamples = copy(samples)
    deleteat!(newsamples,i)
    kde(λ,x,y,newsamples)
end

# first line approximates f̂², the second line approximates -(2/n)f̂f
J(λ) = sum([kde(λ,x,y,samples)^2 for x=xs,y=ys])*step(xs)*step(ys) -
                2/length(samples)*sum(kdeCV(λ,i,samples) for i=1:length(samples))
λ_best_cv = optimize(λ->J(first(λ)),[1.0],BFGS())
```

The cross-validated minimizing value is 1.89 (recall that the true minimizer is 1.92). So cross validation gets us quite close to the optimal λ value without needing to know the actual density. In fact, the cross-validation estimator performs well in general, in the following sense:

Theorem 1.1.1: Stone's Theorem

Suppose that f is a bounded probability density function. Let $\widehat{f}_n^{\mathrm{CV}}$ be the kernel density estimator with bandwidth λ obtained by cross-validation, and let $\widehat{f}_n$ be the kernel density estimator with optimal bandwidth $\lambda_n^{\min}$. Then

$$\frac{\int (f - \widehat{f}_n^{\mathrm{CV}})^2}{\int (f - \widehat{f}_n)^2}$$

converges in probability to 1 as $n \to \infty$. Furthermore, there are constants C_1 and C_2 such that* $\int (f - \widehat{f}_n)^2 \approx C n^{-4/5}$ and $\lambda_n^{\min} \approx C n^{-1/5}$ for large n.

* Here $f \approx g$ means that $f/g \to 1$.

1.1.1.3 Nonparametric regression

With an estimate of the joint density of X and Y in hand, we can turn to the problem of estimating the regression function $r(x) = \mathbb{E}[Y \mid X = x]$. If we restrict the density estimate $\widehat{f}_\lambda$ to the vertical line at position x, we find that the conditional density is

$$\widehat{f}_{Y \mid X=x}(y) = \frac{\frac{1}{n}\sum_{i=1}^{n} K_\lambda(x - X_i, y - Y_i)}{\int \frac{1}{n}\sum_{i=1}^{n} K_\lambda(x - X_i, y - Y_i)\,\mathrm{d}y}.$$

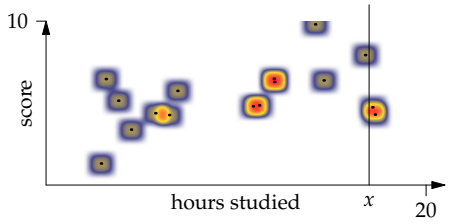


Figure 1.6 A kernel density estimator with 16 samples and $\lambda = 1$.

So the conditional expectation of Y given $\{X = x\}$ is

$$\widehat{r}(x) = \int y \widehat{f}_{Y \mid X=x}(y)\,\mathrm{d}y = \frac{\int y \frac{1}{n}\sum_{i=1}^{n} K_\lambda(x - X_i, y - Y_i)\,\mathrm{d}y}{\int \frac{1}{n}\sum_{i=1}^{n} K_\lambda(x - X_i, y - Y_i)\,\mathrm{d}y}. \tag{1.1.3}$$

Looking at Figure 1.6, we can see by the symmetry of the function D that each sample (X_i, Y_i) contributes $Y_i D_\lambda(x - X_i)$ to the numerator of (1.1.3). To arrive at the same conclusion by manipulating the formula directly, we write

$$\int \frac{1}{n}\sum_{i=1}^{n} K_\lambda(x - X_i, y - Y_i)\,\mathrm{d}y = \frac{1}{n}\sum_{i=1}^{n} D_\lambda(x - X_i) \int y D_\lambda(y - Y_i)\,\mathrm{d}y.$$

Then $\int y D(y - Y_i)\,\mathrm{d}y$ is the average of the probability measure with density D centered at Y_i. Since D is symmetric, this average is the center point Y_i.

Applying a similar idea to the denominator (note that instead of Y_i, we get the total mass 1 from integrating $D(y - Y_i)$), we find that

$$\widehat{r}(x) = \frac{\sum_{i=1}^{n} D_\lambda(x - X_i) Y_i}{\sum_{i=1}^{n} D_\lambda(x - X_i)}.$$

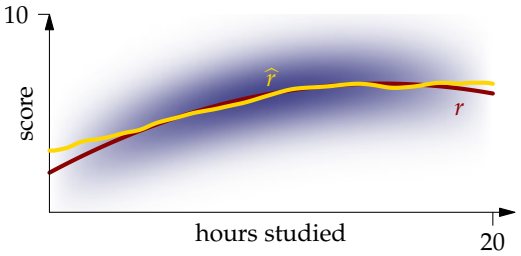


Figure 1.7 The Nadaraya-Watson estimator $\widehat{r}$.

The graph of this function is shown in Figure 1.7. We see that it matches the graph of r quite closely except near the ends of the interval.

```
λ = first(λ_best_cv.minimizer)
r̂(x) = sum(D(λ,x-Xi)*Yi for (Xi,Yi) in samples)/sum(D(λ,x-Xi) for (Xi,Yi) in samples)
plot(r̂,0,20)
```

The approximate integrated squared error of this estimator is `sum((r(x)-r̂(x))^2 for x in xs)*step(xs)` = 1.9.

1.1.1.4 Parametric regression

Another common approach to estimating the regression function of a joint distribution is to assume that the regression function takes a particular form. For example, to perform a **linear regression**, we posit that $r(x) = \beta_0 + \beta_1 x$ for some constants β_0 and β_1. To estimate $\boldsymbol{\beta} = [\beta_0, \beta_1]$ from the n samples $\{(x_i, y_i)\}_{i=1}^n$, one typically* minimizes the **residual sum of squares**:

$$\mathrm{RSS}(\boldsymbol{\beta}) = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2.$$

* The *Gauss-Markov* theorem provides justification: the RSS minimizer is the *best linear unbiased estimator*

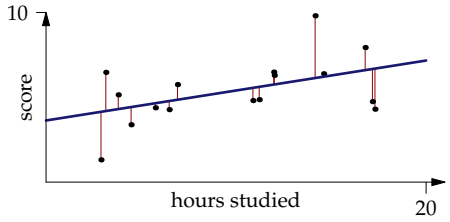


Figure 1.8 The line of best fit minimizes the sum of the squares of the lengths of the red segments.

Example 1.1.2

Find the value of $\boldsymbol{\beta}$ which minimizes $\mathrm{RSS}(\boldsymbol{\beta})$.

Solution

We can write $\mathrm{RSS}(\boldsymbol{\beta})$ as

$$|\mathbf{y} - X\boldsymbol{\beta}|^2,$$

where $\mathbf{y} = [y_1, \ldots, y_n]$ and

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}.$$

The function $\boldsymbol{\beta} \mapsto |\mathbf{y} - X\boldsymbol{\beta}|^2$ goes to infinity as $\boldsymbol{\beta} \to \infty$ and is continuous, so it necessarily has at least one global minimum. Differentiating, we get

$$\frac{\partial}{\partial \boldsymbol{\beta}} (\mathbf{y} - X\boldsymbol{\beta})'(\mathbf{y} - X\boldsymbol{\beta}) = -2X'(\mathbf{y} - X\boldsymbol{\beta}).$$

we can solve to find that

$$\boldsymbol{\beta} = (X'X)^{-1}X'\mathbf{y}.$$

is the only critical point. Therefore, it must be the unique global minimizer.

Let's apply this formula to find the linear regression estimate for our running exam-scores example.

```
y = [Y for (X,Y) in samples]
X = [ones(n) [X for (X,Y) in samples]]
β = (X'*X) \ X'*y # (computes (X'X)⁻¹X'y)
scatter(samples)
plot!(xs,[β[1,x] for x in xs])
```

See Figure 1.9 for a graph of this function.

We can also use linear regression techniques to handle *polynomial* regression. If we posit that

$$r(x) = \beta_0 + \beta_1 X + \beta_2 X^2,$$

then we find the coefficients $\boldsymbol{\beta} = [\beta_0, \beta_1, \beta_2]$ which minimize the residual sum of squares by writing

$$\mathrm{RSS}(\boldsymbol{\beta}) = |y - X\boldsymbol{\beta}|^2,$$

where

$$X = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{bmatrix}.$$

Example 1.1.3

Perform a quadratic regression on the running exam-scores example, and show that its integrated squared difference from the true regression function is lower than that of the linear estimator and the Nadaraya-Watson estimator.

Solution

```
y = [Y for (X,Y) in samples]
X = [ones(n) [X for (X,Y) in samples] [X^2 for (X,Y) in samples]]
βq = (X'*X) \ X'*y
scatter(samples)
plot!(xs,[βq[1,x,x^2] for x in xs])
quaderr = sum((r(x)-βq[1,x,x^2])^2 for x in xs)*step(xs)
linerr = sum((r(x)-β[1,x])^2 for x in xs)*step(xs)
```

We perform the quadratic regression by doing the same calculation as for the linear regression but with an extra column in X. We approximate the integrated squared error using a Riemann sum as we did for the kernel density estimator. We find that the integrated squared errors for the quadratic estimator, the kernel density estimator, and the linear estimator are 0.61, 1.9, and 9.24, respectively.

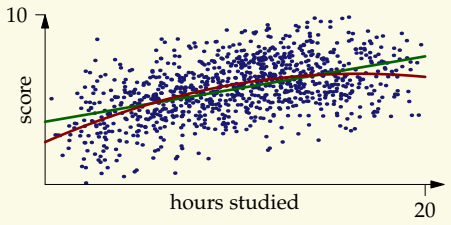


Figure 1.9 Linear and quadratic regression curves.

The results of Example 1.1.3 are telling: the true regression function was quadratic for this example, and assuming quadraticness enabled us to achieve more than three times lower error than for the nonparametric estimator. On the other hand, the assumption that the regression function is linear resulted in significantly *greater* error. The imposition of assumptions on our model is called **inductive bias**, and we can see that it is a double-edged sword: it tends to enhance accuracy if the assumptions applied are correct or approximately correct, and it can result in lower accuracy if not.

1.1.2 Classification

The task in Example 1.1.1 is to predict a quantitative random variable Y given the value of a random variable X. Another common machine learning task is to predict a random variable Y taking values in a discrete set of *labels*. These are called **classification** problems.

Exercise 1.1.4

Suppose that (X, Y) are random variables defined on the same probability space, and suppose that Y takes values in $\{1, 2, 3\}$. For example, suppose that we select a forest animal at random and let X be its weight and Y the kind of animal it is (where $1, 2$, and 3 correspond to squirrel, bear, and fox, respectively). Suppose that f is a function which is intended to predict the value of Y based on the value of X. Explain why the mean squared error $\mathbb{E}[(Y - f(X))^2]$ is not a reasonable way to measure the accuracy of the prediction function.

The most common way to judge a prediction function for a classification problem is the **0-1 loss**, which applies a penalty of 1 for misclassification and 0 for correct classification:

$$L(f) = \mathbb{E}[\mathbf{1}_{Y \neq f(X)}] = \mathbb{P}(Y \neq f(X)).$$

Since it is typically not meaningful to put the possible classifications in order along an axis, we usually represent a data point's classification graphically using the point's shape or color. This allows us to use all of the spatial dimensions in the figure for the X values, which is helpful if X is multidimensional.

Example 1.1.4

Given a flower randomly selected from a field, let X_1 be its petal width in centimeters, X_2 its petal length in centimeters, and $Y \in \{\mathrm{R}, \mathrm{G}, \mathrm{B}\}$ its color. Let

$$\boldsymbol{\mu}_\mathrm{R} = \begin{bmatrix} 9 \\ 5 \end{bmatrix} \quad \boldsymbol{\mu}_\mathrm{G} = \begin{bmatrix} 4 \\ 10 \end{bmatrix} \quad \boldsymbol{\mu}_\mathrm{B} = \begin{bmatrix} 7 \\ 9 \end{bmatrix}$$
$$A_\mathrm{R} = \begin{bmatrix} 1.5 & -1 \\ 0 & 1 \end{bmatrix} \quad A_\mathrm{G} = \begin{bmatrix} 0.5 & 0.25 \\ 0 & 0.5 \end{bmatrix} \quad A_\mathrm{B} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}.$$

Suppose that the joint distribution of X_1, X_2, and Y has the property that for any $A \subset \mathbb{R}^2$ and color $c \in \{\mathrm{R}, \mathrm{G}, \mathrm{B}\}$, we have

$$\mathbb{P}(A \times \{c\}) = p_c \int_{\mathbb{R}^2} f_c(x_1, x_2)\, \mathrm{d}x_1\, \mathrm{d}x_2,$$

where $(p_R, p_G, p_B) = (1/3, 1/6, 1/2)$ and f_c is the multivariate normal density with mean μ_c and covariance matrix $A_c A_c'$.

Three hundred samples from the distribution of (X_1, X_2, Y) are shown in Figure 1.10.

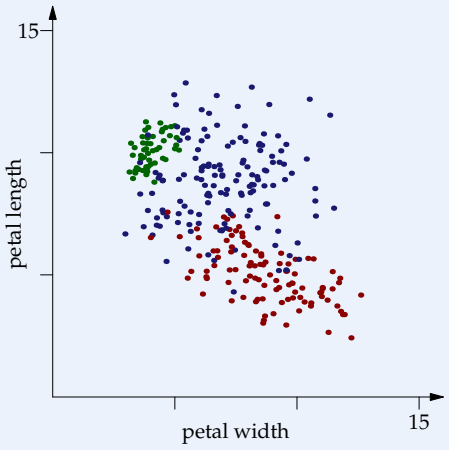


Figure 1.10 Colors and petal dimensions of 300 randomly selected flowers.

Find the best predictor of Y given $(X_1, X_2) = (x_1, x_2)$ (using the 0-1 loss function), and find a way to estimate that predictor using the given samples.

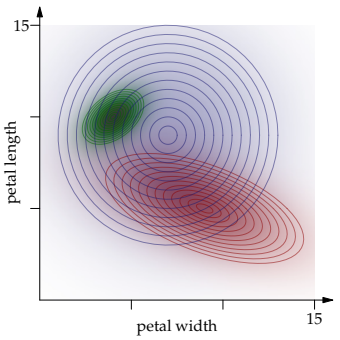


Figure 1.11 The conditional densities of the petal dimensions given flower color.

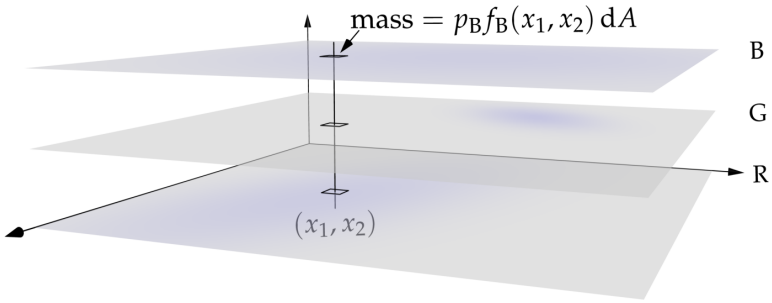


Figure 1.12 The probability mass in $\mathbb{R}^3$ for the joint distribution of (X_1, X_2, Y), with the association $R \to 0$, $G \to 1$, $B \to 2$.

As in the regression example, we can do a decent job of classification with our eyes. If (X_1, X_2) is located where there are lots of green sample points, we would predict its classification as green, and similarly for blue and red. Let's think about how to approach this task mathematically.

First, suppose that the distribution of (X_1, X_2, Y) is known. Then the predictor which has minimal misclassification probability is the one which maps (x_1, x_2) to the classification with maximal conditional probability given (x_1, x_2). For example, if the conditional distribution on $\{R, G, B\}$ given (x_1, x_2) were $\{45\%, 30\%, 25\%\}$, then we would guess a classification of R for the point (x_1, x_2).

The conditional distribution of Y given (X_1, X_2) is given by

$$m_{(X_1,X_2)=(x_1,x_2)}(c) = \frac{p_c f_c(x_1, x_2)}{\sum_{d\in\{R,G,B\}} p_d f_d(x_1, x_2)} \tag{1.1.4}$$

for $c \in \{R, G, B\}$ (see Figure 1.12).

Let's depict the optimal classifier for Example 1.1.4. First, let's get 300 samples from the joint distribution of (X_1, X_2, Y):

```
using Plots, StatsBase, Random; Random.seed!(1234)
struct MNormal
    μ::Vector
    Σ::Array
end
struct Flower
    X::Vector
    color::String
end
# density function for the normal distribution N
f(x,N::MNormal) = 1/(2π*√(det(N.Σ))) * exp(-1/2*((x-N.μ)'*inv(N.Σ)*(x-N.μ)))
xs = 0:1/2^4:15
ys = 0:1/2^4:15
As = [[1.5 -1; 0 1],[1/2 1/4; 0 1/2], [2 0; 0 2]]
μs = [[9,5],[4,10],[7,9]]
Ns = [MNormal(μ,A*A') for (μ,A) in zip(μs,As)]
p = Weights([1/3,1/6,1/2])
colors = ["red","green","blue"]
function randflower(μs,As)
```

```
    i = sample(p)
    Flower(As[i]*randn(2)+μs[i],colors[i])
end
flowers = [randflower(μs,As) for i=1:300]
```

* We set `transpose` to `true` so that the coordinates are represented coordinate-plane style (origin bottom left) rather than matrix-index style (origin top left).

Next, let's make a classifier and color all of the points in a fine-mesh grid according to their predicted classifications.*

```
classify(x,p,Ns) = argmax([p[i]*f(x,Ns[i]) for i=1:3])
function classificationplot(flowers,p,Ns)
    rgb = [:red,:green,:blue]
    P = heatmap(xs,ys,[classify([x,y],p,Ns) for x=xs,y=ys],
                fillcolor=cgrad(rgb),opacity=0.4,transpose=true)
    for c in ["red","green","blue"]
        scatter!(P,[(F.X[1],F.X[2]) for F in flowers if F.color ==c],color=c)
    end
    plot!(aspect_ratio=:equal,legend=false)
    P
end
correct(flowers,p,Ns) = count(colors[classify(F.X,p,Ns)] == F.color for F in flowers)
classificationplot(flowers,p,Ns)
```

We can see in Figure 1.13 that the optimal classifier does get most of the points right, but not all of them. `correct(flowers,p,Ns)` returns 265, so the optimal classification accuracy is $265/300 \approx 88\%$ for this example.

Now suppose we don't have access to the joint distribution of (X_1, X_2, Y), but we do have n samples $\{(X_1^{(i)}, X_2^{(i)}, Y^{(i)})\}_{i=1}^n$ therefrom. We can estimate $\widehat{p}_c$ as the proportion of observed flowers of color c. We could estimate the conditional densities f_c using kernel density estimation, but in the interest of bringing in a new idea, let's fit a multivariate normal distribution to the samples of each color.

Let's begin by approximating the mean of the distribution of red flowers. The law of large numbers suggests that we use the mean of the red flower samples: if $\mathcal{R}$ is the set of red flowers, then we set

$$\widehat{\boldsymbol{\mu}}_{\mathrm{R}} = \frac{1}{|\mathcal{R}|} \sum_{i \in \mathcal{R}} \begin{bmatrix} X_1^{(i)} \\ X_2^{(i)} \end{bmatrix},$$

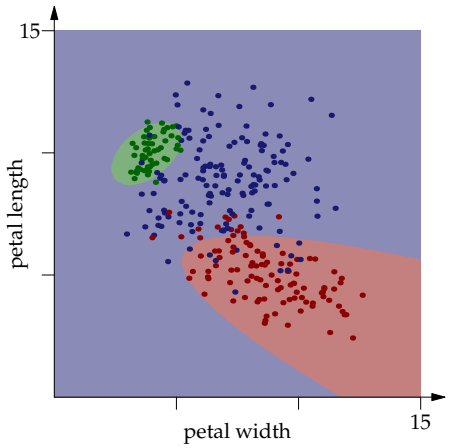


Figure 1.13 Each point in the square is colored according to the optimal classifier's prediction for the given (x_1, x_2) pair.

and similarly for the other two colors. This formula estimates $\mathbb{E}[\mathbf{X}]$ by using the empirical distribution as a proxy for the underlying distribution. Likewise, we approximate the red covariance matrix as

$$\widehat{\Sigma}_{\mathrm{R}} = \frac{1}{|\mathcal{R}|} \sum_{i \in \mathcal{R}} \left(\begin{bmatrix} X_1^{(i)} \\ X_2^{(i)} \end{bmatrix} - \widehat{\boldsymbol{\mu}}_{\mathrm{R}} \right) \left(\begin{bmatrix} X_1^{(i)} \\ X_2^{(i)} \end{bmatrix} - \widehat{\boldsymbol{\mu}}_{\mathrm{R}} \right)',$$

which evaluates the covariance matrix formula $\mathbb{E}[(\mathbf{X} - \boldsymbol{\mu}_{\mathbf{X}})(\mathbf{X} - \boldsymbol{\mu}_{\mathbf{X}})']$ with respect to the empirical distribution.

```
function mvn_estimate(flowers,color)
    flowers_subset = [F.X for F in flowers if F.color == color]
    μ̂ = mean(flowers_subset)
    Σ̂ = mean([(X - μ̂)*(X - μ̂)' for X in flowers_subset])
    MNormal(μ̂,Σ̂)
end
colorcounts = countmap([F.color for F in flowers])
p̂ = [colorcounts[c]/length(flowers) for c in colors]
N̂s = [mvn_estimate(flowers,c) for c in colors]
classificationplot(flowers,p̂,N̂s)
```

The resulting plot looks very similar to Figure 1.13, so this classifier makes the same prediction as the optimal classifier for most points (x_1, x_2).

Exercise 1.1.5

In this problem, we will implement a classifier for the flower data based on kernel density estimation.

(a) For each color, find the cross-validation kernel density estimator for the set of flowers of that color.

(b) Substitute your estimates into (1.1.4) to obtain a classifier. Make a plot similar to Figure 1.13 and compare.

1.2 Statistical learning theory

Having looked at some examples, let's establish a general framework for the machine learning models we will study in the next chapter.

At the highest level, the goal of **statistical learning** is to draw conclusions about an unknown probability measure given independent samples from the measure. These samples are called **training data**. In **supervised learning**, the unknown measure $\mathbb{P}$ is on a product space $\mathcal{X} \times \mathcal{Y}$. We aim to use the training data to predict Y given $\mathbf{X}$, where $(\mathbf{X}, Y)$ denotes a random variable in $\mathcal{X} \times \mathcal{Y}$ with distribution $\mathbb{P}$.

Example 1.2.1

Suppose that $\mathbf{X} = [X_1, X_2]$, where X_1 is the color of a banana, X_2 is the weight of the banana, and Y is a measure of banana deliciousness. Values of X_1, X_2, and Y are recorded for many bananas, and they are used to predict Y for other bananas whose $\mathbf{X}$ values are known.

Do you expect the prediction function to be more sensitive to changes in X_1 or changes in X_2?

Solution

Presumably X_1 has more influence on Y, since bananas can taste great whether small or large, while green or dark brown bananas generally taste very bad and yellow bananas mostly taste good.

Definition 1.2.1

We call the components of $\mathbf{X}$ *features*, *predictors*, or *input variables*, and we call Y the *response variable* or *output variable*.

We categorize supervised learning problems based on the structure of $\mathcal{Y}$.

Definition 1.2.2

A supervised learning problem is a **regression** problem if Y is quantitative ($\mathcal{Y} \subset \mathbb{R}$) and a **classification** problem if $\mathcal{Y}$ is a set of labels.

We can treat Example 1.2.1 as either a regression problem (if deliciousness is a score on a spectrum) or a classification problem (if the bananas are merely categorized as "good" or "bad"). We will see that many models are specific to regression or classification, while others can be adapted to handle either type of problem.

To make meaningful and unambiguous statements about a proposed prediction function $h : \mathcal{X} \to \mathcal{Y}$, we need a rubric by which to assess it. This is customarily done by defining a *loss* (or *risk*, or *error*) $L(h)$, with the idea that smaller loss is better. We might wish to define L only for h's in a specified class $\mathcal{H}$ of candidate functions. Since $L : \mathcal{H} \to \mathbb{R}$ is defined on a set of functions, we call L the **loss functional**.*

* *Functional* is a general term for a real-valued function whose domain is a set of functions.

Once a set $\mathcal{H}$ of candidate functions and a loss functional L are specified, a prediction function $h \in H$ must be proposed. This can often be done the lazy way, by defining our prediction function implicitly as the function in $\mathcal{H}$ which minimizes L:

Definition 1.2.3

Given a statistical learning problem, a space $\mathcal{H}$ of candidate prediction functions, and a loss functional $L : \mathcal{H} \to \mathbb{R}$, we define the **target function** to be $\operatorname{argmin}_{h \in \mathcal{H}} L(h)$.

Let's look at some common loss functionals. For regression, we use the **mean squared error**:

$$L(h) = \mathbb{E}[(h(X) - Y)^2]$$

If $\mathcal{H}$ contains $r(\mathbf{x}) = \mathbb{E}[Y \mid \mathbf{X} = \mathbf{x}]$, then r is the target function. For classification, we consider the **misclassification probability**

$$L(h) = \mathbb{E}\left[\mathbf{1}_{\{h(\mathbf{X}) \neq Y\}}\right] = \mathbb{P}(h(\mathbf{X}) \neq Y).$$

If $\mathcal{H}$ contains $G(\mathbf{x}) = \operatorname{argmax}_c \mathbb{P}(Y = c \mid \mathbf{X} = \mathbf{x})$, then G is the target function for this loss functional.

Note that neither of these loss functionals can be computed directly unless the probability measure $\mathbb{P}$ on $\mathcal{X} \times \mathcal{Y}$ is known. Since the goal of statistical learning is to make inferences about $\mathbb{P}$ when it is *not* known, we must approximate L (and likewise also the target function h) using the training data.

The most straightforward way to do this is to replace $\mathbb{P}$ with the **empirical probability measure** associated with the training data $\{(\mathbf{X}_i, Y_i)\}_{i=1}^n$. This is the probability measure which places $\frac{1}{n}$ units of probability mass at $(\mathbf{X}_i, Y_i)$, for each i from 1 to n. The **empirical risk** of a candidate function $h \in \mathcal{H}$ is the risk functional evaluated with respect to the empirical measure of the training data.

A **learner** is a function which takes a set of training data and returns a prediction function $\widehat{h}$. A common

way to specify a learner is to let $\widehat{h}$ be the **empirical risk minimizer** (ERM), which is the function in $\mathcal{H}$ which minimizes the empirical risk.

Example 1.2.2

Suppose that $\mathcal{X} = [0,1]$ and $\mathcal{Y} = \mathbb{R}$, and that the probability measure on $\mathcal{X} \times \mathcal{Y}$ is the one which corresponds to sampling $\mathbf{X}$ uniformly from $[0,1]$ and then sampling Y from $\mathcal{N}(X/2+1, 1)$.

Let $\mathcal{H}$ be the set of monic* polynomials of degree six or less. Given training samples $\{(\mathbf{X}_i, Y_i)\}_{i=1}^6$, find the risk minimizer and the empirical risk minimizer for the mean squared error.

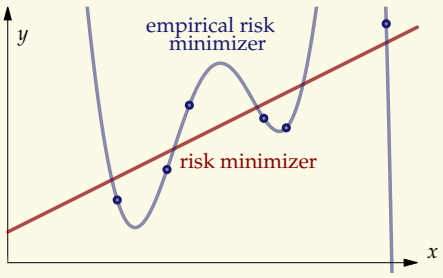
* *monic* means that the leading coefficient is 1.

Solution

The risk minimizer is $\mathbb{E}[Y \mid X]$, which is given in the problem statement as $X/2+1$.

The empirical loss function is obtained by replacing the expectation in the loss functional with an expectation with respect to the empirical measure. So the empirical risk minimizer is the function which minimizes

$$\sum_{i=1}^{n} (Y_i - h(\mathbf{X}_i))^2.$$



Since there is exactly one monic polynomial of degree 6 or less which passes through the points $\{(\mathbf{X}_i, Y_i)\}_{i=1}^6$, that function is the empirical risk minimizer.

Example 1.2.2 illustrates a phenomenon called **overfitting**. Although the empirical risk is small for the prediction function h we found, smallness of the empirical risk does not imply smallness of the true risk.* The difference between empirical risk and the actual value of the risk functional is called **generalization error** (or *test* error).

* That is, the risk functional of h evaluated with respected to the true probability measure $\mathbb{P}$

We mitigate overfitting by building **inductive bias** into the model. Common approaches include

(i) using a restrictive class $\mathcal{H}$ of candidate functions,

(ii) **regularizing**: adding a term to the loss functional which penalizes complexity, and

(iii) **cross-validating**: proposing a spectrum of candidate functions and selecting the one which performs best on witheld training samples.

Example 1.2.3

(a) Which method of introducing inductive bias does linear regression use?

(b) Which method did we use for kernel density estimation?

Solution

(a) Linear regression uses a restrictive class $\mathcal{H}$ of candidate functions. It is clear that the kind of overfitting that we saw in Example 1.2.2 is impossible if such curvy functions are excluded from consideration.

(b) We used cross-validation. We suggested a family of density estimators $\widehat{f}_\lambda$ parameterized by bandwidth λ, and we chose a value of λ by withholding a single data point at a time to estimate the integrated squared error.

Inductive bias can lead to **underfitting**: relevant relations are missed, so both training and test error are larger than necessary. The tension between underfitting and overfitting is the **bias-complexity** (or *bias-variance*) **tradeoff**.

For a proof of this theorem, see the 1996 paper *The Lack of A Priori Distinctions Between Learning Algorithms* by David Wolpert.

This tension is fundamentally unresolvable, in the sense that *all learners are equal on average* (!!).

Theorem 1.2.1: No free lunch

Suppose $\mathcal{X}$ and $\mathcal{Y}$ are finite sets, and let f denote a probability distribution on $\mathcal{X} \times \mathcal{Y}$. Let D be a collection of n independent samples from f, and let h_1 and h_2 be prediction functions (which associate a prediction $h_j(d, \mathbf{x}) \in \mathcal{Y}$ to each pair $(d, \mathbf{x})$ where d is a set of training samples and $\mathbf{x} \in \mathcal{X}$). Consider the cost random variable $C_j = (h_j(D, X) - Y)^2$ (or $C_j = \mathbf{1}_{\{h_j(D,X)-Y\}}$) for $j \in \{1, 2\}$.

The average* over all distributions f of the distribution of C_1 is equal to the average over all distributions f of the average of C_2.

* The set of distributions f can be thought of as the set of points in $\mathbb{R}^{|\mathcal{X}\times\mathcal{Y}|}$ whose coordinates are nonnegative and sum to 1. Averaging means integrating over that set and dividing by its volume.

The no-free-lunch theorem implies that machine learning is effective in real life only because the learners used by practitioners possess inductive bias which is well-aligned with data-generating processes encountered in machine learning application domains. A learner which is more effective than another on a particular class of problems must perform *worse* on average on problems which are not in that class.

Exercise 1.2.1

Cross-validation seems to be free of inductive bias: insisting that an algorithm perform well on withheld samples hardly seems to assume any particular structure to the probability distribution or regression function being estimated. Show that this is not the case.

1.2.1 Likelihood ratio classification

* A binary classification problem is a classification problem with two classes

In this section, we will introduce some vocabulary and results specific to binary* classification. Borrowing from the language of disease diagnosis, will call the two classes *positive* and *negative* (which, in the medical context, indicate presence or absence of the disease in question). Correctly classifying a positive sample is called **detection**, and incorrectly classifying a negative sample is called **false alarm** or **type I error**.

Suppose that $\mathcal{X}$ is the feature set of our classification problem and that $\mathcal{Y} = \{+1, -1\}$ is the set of classes. Denote by (X, Y) a random sample from the probability measure on $\mathcal{X} \times \mathcal{Y}$. We define p_{+1} to the probability that a sample is positive and $p_{-1} = 1 - p_{+1}$ to be the probability that a sample is negative. Let f_{+1} be the conditional PMF or PDF of X given the event $\{Y = +1\}$, and let f_{-1} be the conditional PMF or PDF of X given $\{Y = -1\}$. We call f_{+1} and f_{-1} *class conditional distributions*.

Given a function $h : \mathcal{X} \to \mathcal{Y}$ (which we call a **classifier**), we define its* **confusion matrix** to be

$$\begin{bmatrix} \mathbb{P}(h(X) = +1 \,|\, Y = +1) & \mathbb{P}(h(X) = +1 \,|\, Y = -1) \\ \mathbb{P}(h(X) = -1 \,|\, Y = +1) & \mathbb{P}(h(X) = -1 \,|\, Y = -1) \end{bmatrix}.$$

* This confusion matrix is *normalized*; if a particular set of samples is in view, counts are often used instead of probabilities.

We call the top-left entry of the confusion matrix the **detection rate** (or *true positive rate*, or *recall* or *sensitivity*) and the top-right entry the **false alarm rate** (or *false postive rate*).

Example 1.2.4

The **precision** of a classifier h is the conditional probability of $\{Y = +1\}$ given $\{h(X) = +1\}$. Show that a classifier can have high detection rate, low false alarm rate, and low precision.

Solution

Suppose that $p_{-1} = 0.999$ and that h has detection rate 0.99 and false alarm rate 0.01. Then the precision of h is

$$\mathbb{P}(Y = +1 \,|\, h(X) = +1) = \frac{\mathbb{P}(\{Y = +1\} \cap \{h(X) = +1\})}{\mathbb{P}(h(X) = +1)} = \frac{(0.001)(0.99)}{(0.001)(0.99) + (0.999)(0.01)} \approx 0.09.$$

We see that, unlike detection rate and false alarm rate, precision depends on the value of p_{-1}. If p_{-1} is very high, it can result in low precision even if the classifier has high accuracy within each class.

As discussed in Section 1.2, the **Bayes classifier**

$$h(\mathbf{x}) = \begin{cases} +1 & \text{if } p_{+1} f_{+1}(\mathbf{x}) \geq p_{-1} f_{-1}(\mathbf{x}) \\ -1 & \text{otherwise} \end{cases}$$

minimizes the probability of misclassification. In other words, it is the classifier h for which

$$\mathbb{P}(h(X) = +1 \text{ and } Y = -1) + \mathbb{P}(h(X) = -1 \text{ and } Y = +1)$$

is as small as possible. However, the two types of misclassification often have different real-world consequences* , and we might therefore wish to weight them differently. Given $t \geq 0$, we define the classifier

$$h_t(\mathbf{x}) = \begin{cases} +1 & \text{if } \frac{f_{+1}(\mathbf{x})}{f_{-1}(\mathbf{x})} \geq t \\ -1 & \text{otherwise.} \end{cases}$$

* Compare, for example, (a) sending an important email to the spam folder, and (b) letting a spam email slip through to the inbox

Example 1.2.5

Show that the likelihood ratio classifier generalizes the Bayes classifier.

Solution

If we let $t = p_{-1}/p_{+1}$, then the inequality $\frac{f_{+1}(\mathbf{x})}{f_{-1}(\mathbf{x})} \geq t$ simplifies to $p_{+1} f_{+1}(\mathbf{x}) \geq p_{-1} f_{-1}(\mathbf{x})$. Therefore, the Bayes classifier is equal to $h_{p_{-1}/p_{+1}}$.

If we increase t, then some of the predictions of h_t switch from $+1$ to -1, while others stay the same. Therefore, the detection rate and false alarm rate both decrease as t increases. Likewise, if we decrease t, then detection rate and false alarm rate both increase. If we let t range over the interval $[0, \infty]$ and plot each ordered pair $(\text{FAR}(h_t), \text{DR}(h_t))$, then we obtain a curve like the one shown in Figure 1.14. This curve is called the **receiver operating characteristic** of the likelihood ratio classifier.

The ideal scenario is that this curve passes through points near the top left corner of the square, since that means that some of the classifiers in the family $\{h_t : t \in [0, \infty]\}$ have both high detection rate and low false alarm rate. We quantify this idea using the **area under the ROC** (called the AUROC). This value is close to 1 for an excellent classifier and close to $\frac{1}{2}$ for a classifier whose ROC is the diagonal line from the origin to $(1, 1)$.

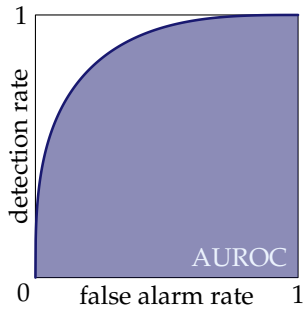


Figure 1.14 The receiver operating characteristic

Example 1.2.6

Suppose that $\mathcal{X} = \mathbb{R}$ and that the class conditional densities for -1 and $+1$ are normal distributions with unit variances and means 0 and μ, respectively. For each $\mu \in \{1/4, 1, 4\}$, predict the approximate shape of the ROC for the likelihood ratio classifier. Then calculate it explicitly and plot it.

Solution

We predict that the ROC will be nearly diagonal for $\mu = \frac{1}{4}$, since the class conditional distributions overlap heavily, and therefore any increase in detection rate will induce an approximately equal increase in false alarm rate. When $\mu = 4$, we expect to get a very large AUROC, since in that case the distributions overlap very little. The $\mu = 1$ curve will lie between these extremes.

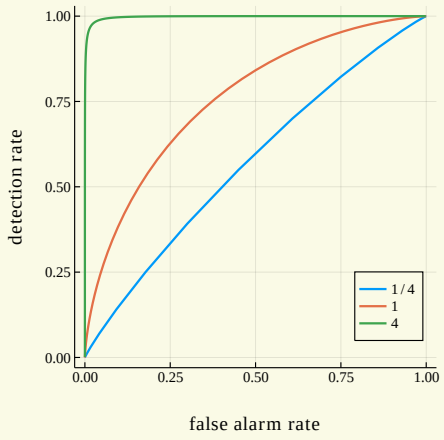
To plot these curves, we begin by calculating the likelihood ratio

$$\frac{f_{+1}(x)}{f_{-1}(x)} = \frac{e^{-(x-\mu)^2/2}}{e^{-x^2/2}} = e^{\mu x - \mu^2/2},$$

So the detection rate for h_t is the probability that a sample drawn from $\mathcal{N}(\mu, 1)$ lies in the region where $e^{\mu x - \mu^2/2} \geq t$. Solving this inequality for x, we find that the detection rate is equal to the probability mass assigned to the interval $\left[\frac{\log t}{\mu} + \frac{\mu}{2}, \infty\right)$ by the distribution $\mathcal{N}(\mu, 1)$.

Likewise, the false alarm rate is the probability mass assigned to the same interval by the negative class conditional distribution, $\mathcal{N}(0, 1)$.

```
using Plots, Distributions
FAR(μ,t) = 1-cdf(Normal(0,1),log(t)/μ + μ/2)
DR(μ,t) = 1-cdf(Normal(μ,1),log(t)/μ + μ/2)
ROC(μ) = [(FAR(μ,t),DR(μ,t))
                  for t in exp.(-20:0.1:20)]
plot(ROC(1/4),label="1/4")
plot!(ROC(1),label="1")
plot!(ROC(4),label="4")
plot!(xlabel="false alarm rate",
      ylabel="detection rate")
```



Note that for any two likelihood ratio classifiers, one of them has a worse detection rate and better false alarm rate than the other. This means that none of the likelihood ratio classifiers is uniformly better than any other. In fact, there are no classifiers *of any kind* which are uniformly better (on detection rate and false alarm rate) than any of the likelihood ratio classifiers:

Theorem 1.2.2: The Neyman-Pearson lemma

If h is a classifier, then the point $(\text{FAR}(h), \text{DR}(h))$ lies on or below the likelihood ratio ROC.

Exercise 1.2.2

Show that for each $\alpha \in [0, 1]$, there exists t such that such that the likelihood ratio classifier h_t is the function $h : \mathcal{X} \to \mathcal{Y}$ which minimizes

$$L(h) = \alpha\, \mathbb{P}(h(X) = +1 \text{ and } Y = -1) + (1 - \alpha)\mathbb{P}(h(X) = -1 \text{ and } Y = +1).$$

Identify the relationship between α and its corresponding t value. (For simplicity, assume that $\mathcal{X}$ is finite.)

2 Machine learning models

2.1 Generative models

In Examples1.1.1 and 1.1.4, we built our prediction function (either a regression estimate or classifier) using an estimate of the probability distribution of the data-generating process. Such models are called **generative**. Models which estimate the prediction function directly—such as linear or polynomial regression—are **discriminative** models. In this section we will discuss three generative models.

2.1.1 Quadratic and linear discriminant analysis

The approach we took to solving Example 1.1.4 is called **quadratic discriminant analysis** (QDA). To recap, we posited that the class conditional densities are multivariate Gaussian, and we used the sample points from each class to find a mean estimate $\widehat{\boldsymbol{\mu}}_c$ and covariance matrix estimate $\widehat{\Sigma}_c$ for the distribution of class c. We also used the sample proportions $\widehat{p}_c$ to estimate the class proportions. Then we returned the classifier $h(\mathbf{x}) = \text{argmax}_c\, \widehat{p}_c \widehat{f}_c(\mathbf{x})$ (where $\widehat{f}_c$ is the multivariate normal density with mean $\widehat{\boldsymbol{\mu}}_c$ and covariance $\widehat{\Sigma}_c$).

A common variation on this idea is to posit that the class conditional densities have the same covariance matrix. Then sample points from all of the classes can be pooled to estimate this common covariance matrix. We estimate the mean $\widehat{\boldsymbol{\mu}}_c$ of each class c, and then we average $(\mathbf{X}_i - \widehat{\boldsymbol{\mu}}_{Y_i})(\mathbf{X}_i - \widehat{\boldsymbol{\mu}}_{Y_i})'$ over all the sample points $(\mathbf{X}_i, Y_i)$. This approach is called **linear discriminant analysis** (LDA). The advantage of LDA over QDA stems from the difficulty of estimating the p^2 entries of a $p \times p$ covariance matrix if p is even moderately large. Pooling the classes allows us to marshal more samples in the service of this estimation task.

The terms *quadratic* and *linear* refer to the resulting decision boundaries: solution sets of equations of the form $p_1 f_1(\mathbf{x}) = p_2 f_2(\mathbf{x})$ are quadric hypersurfaces or hyperplanes* if p_1 and p_2 are real numbers and f_1 and f_2 are distinct multivariate normal densities. If the covariances of f_1 and f_2 are equal, then the solution set $p_1 f_1(\mathbf{x}) = p_2 f_2(\mathbf{x})$ is a hyperplane.

* A *hyperplane* is a solution set of a single linear equation in $\mathbb{R}^n$ (so, a line in $\mathbb{R}^2$, a plane in $\mathbb{R}^3$, etc.). A *quadric hypersurface* is a solution set of a quadratic equation in $\mathbb{R}^n$.

2.1.2 Naive Bayes

The **naive Bayes** approach to classification is to assume that the components of $\mathbf{X}$ are conditionally *independent* given Y. In the context of Example 1.1.4, this would mean assuming that blue-flower petal width and length are independent (which was true in that example), that the red-flower petal width and length are independent (which was not true), and that the green-flower petal width and length are independent (also not true).

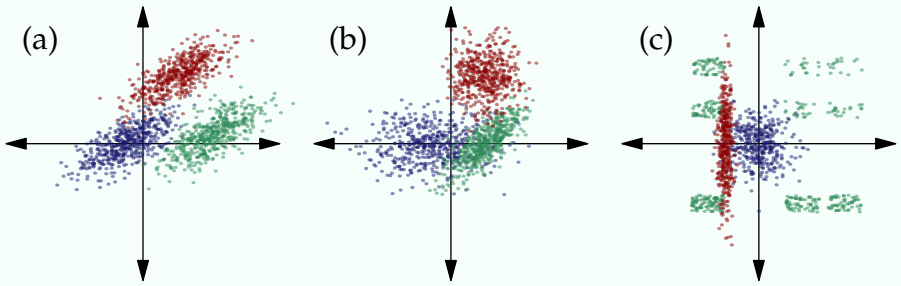
To train a naive Bayes classifier, we use the samples from each class to estimate a density $\widehat{f}_{c,i}$ on $\mathbb{R}$ for each feature component $i = 1, 2, \ldots, p$, and then we estimate

$$\widehat{f}_c(x_1, \ldots, x_n) = \widehat{f}_{c,1}(x_1)\widehat{f}_{c,2}(x_2) \cdots \widehat{f}_{c,p}(x_p),$$

in accordance with the conditional independence assumption. The method for estimating the univariate densities $\widehat{f}_{c,j}$ is up to the user; options include kernel density estimation and parametric estimation.

Exercise 2.1.1

Each scatter plot shows a set of sample points for a three-category classification problem. Match each data set to the best-suited model: Naive Bayes, LDA, QDA.



2.2 Logistic regression

In this section we discuss *logistic regression*, which is a discriminative model for binary classification.

Example 2.2.1

Consider a binary classification problem where the two classes are equally probable, the class-0 conditional density is a standard multivariable normal distribution in two dimensions, and the class-1 conditional density is a multivariate normal distribution with mean $[1,1]$ and covariance I.

Find the class boundary for the Bayes classifier.

Solution

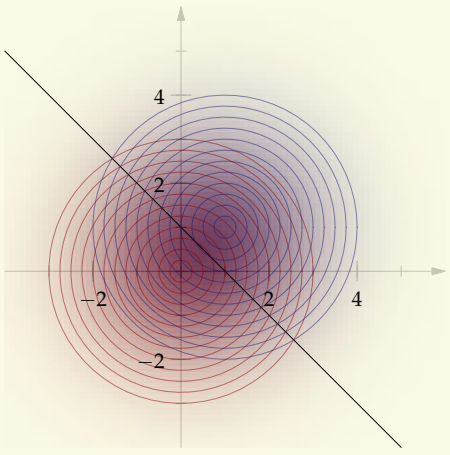
The Bayes classifier is $(x,y) \mapsto \operatorname{argmax}_i p_i f_i(x,y)$, where

$$f_0(x,y) = \frac{1}{2\pi} e^{-\frac{1}{2}(x^2+y^2)}, \text{ and}$$
$$f_1(x,y) = \frac{1}{2\pi} e^{-\frac{1}{2}((x-1)^2+(y-1)^2)}.$$

By symmetry, the classifier will predict class 1 for every point above the line $x+y=1$ and class 0 for every point below the line. We can obtain the same result by solving the equation $f_0(x,y) = f_1(x,y)$. We get

$$-\frac{1}{2}(x^2+y^2) = -\frac{1}{2}((x-1)^2+(y-1)^2),$$

which simplifies to $x+y=1$, as desired.



Example 2.2.2

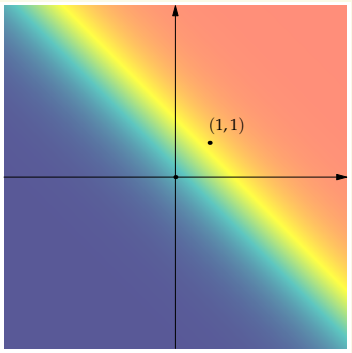
Find the regression function $r(\mathbf{x}) = \mathbb{E}[Y \,|\, \mathbf{X} = \mathbf{x}] = \mathbb{P}(Y = 1 \,|\, \mathbf{X} = \mathbf{x})$ for Example 2.2.1. Plot a heatmap of this function.

Solution

* We also customize the color gradient for the heatmap.

Let's use the multivariate normal type `MvNormal` from the `Distributions` package.*

```
using Plots, Distributions, Optim
mycgrad = cgrad([:MidnightBlue,:SeaGreen,:Gold,:Tomato])
gr(aspect_ratio=1,fillcolor=mycgrad) # Plots.jl defaults
A = MvNormal([0,0],[1.0 0; 0 1])
B = MvNormal([1,1],[1.0 0; 0 1])
xs = -5:1/2^5:5
ys = -5:1/2^5:5
r(x,y) = 0.5pdf(B,[x,y])/(0.5pdf(A,[x,y])+0.5pdf(B,[x,y]))
rs = [r(x,y) for x=xs,y=ys]
heatmap(xs,ys,rs)
```



We can see from the heatmap that restricting $r(\mathbf{x})$ to any line of slope 1 yields a function which asymptotes to 0 in the southwest direction and to 1 in the northeast direction, increasing smoothly in between. Such a function is called a **sigmoid** function.

Given the regression function $r(\mathbf{x}) = \mathbb{P}(Y = 1 \,|\, \mathbf{X} = \mathbf{x})$, we can recover the Bayes classifier by predicting class 1 whenever $r(\mathbf{x}) > \frac{1}{2}$ and class 0 whenever $r(\mathbf{x}) < \frac{1}{2}$. However, the value of the regression function also conveys the degree of confidence associated with the prediction. If $r(\mathbf{x}_1) = 0.65$ and $r(\mathbf{x}_2) = 0.95$, then samples at $\mathbf{x}_1$ and $\mathbf{x}_2$ are both predicted as class 1, but the latter with much more confidence.

The graph in Example 2.2.2 suggests modeling r parametrically as a composition of a linear map and a sigmoid function. Specifically, we posit the model $r(\mathbf{x}) = \sigma(\alpha + \boldsymbol{\beta} \cdot \mathbf{x})$, where $\boldsymbol{\beta} \in \mathbb{R}^p$, $\alpha \in \mathbb{R}$, and $\sigma(x) = 1/(1 + \mathrm{e}^{-x})$.

To select the parameters $\boldsymbol{\beta}$ and α, we penalize lack of confident correctness for each training sample. We give a sample of class 1 the penalty $\log\left(\frac{1}{r_i(x)}\right)$ (which is large if $r_i(x)$ is close to zero and nearly zero if $r_i(x)$ is close to 1). Likewise, we penalize a sample of class 0 by $\log\left(\frac{1}{1-r_i(x)}\right)$.

Example 2.2.3

Sample 1000 points by choosing one of the two distributions uniformly at random and then sampling from the selected distribution. Find the function of the form $\sigma(\boldsymbol{\beta} \cdot \mathbf{x} + \alpha)$ which minimizes

$$L(r) = \sum_{i=1}^{n} \left[y_i \log \frac{1}{r(x_i)} + (1 - y_i) \log \frac{1}{1 - r(x_i)} \right].$$

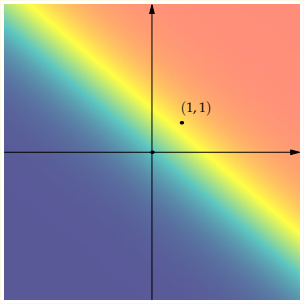
Solution

We begin by sampling the points as suggested.

```
samples = [rand(Bool) ? (rand(A),0) : (rand(B),1) for i=1:1000]
cs =  [c for ((x,y),c) in samples]
scatter([(x,y) for ((x,y),c) in samples],group=cs)
```

Next, we define the loss function and minimize it:*

```
σ(u) = 1/(1 + exp(-u))
r(β,x) = σ(β[1;x])
C(β,xᵢ,yᵢ) = yᵢ*log(1/r(β,xᵢ))+(1-yᵢ)*log(1/(1-r(β,xᵢ)))
L(β) = sum(C(β,xᵢ,yᵢ) for (xᵢ,yᵢ) in samples)
β̂ = optimize(L,ones(3),BFGS()).minimizer
r̂s = [r(β̂,[x,y]) for x=xs,y=ys]
heatmap(xs,ys,r̂s)
```



We can see that the resulting heatmap looks quite similar to the actual regression function.

* \beta«tab» \hat«tab»

Example 2.2.4

In Example 2.2.2, is it true that $r(\mathbf{x}) = \sigma(\boldsymbol{\beta} \cdot \mathbf{x} + \alpha)$ for some $\boldsymbol{\beta}$ and α?

Solution

We calculate

$$\frac{\frac{1}{2}f_0(x,y)}{\frac{1}{2}f_0(x,y) + \frac{1}{2}f_1(x,y)} = \frac{e^{-\frac{x^2}{2}-\frac{y^2}{2}}}{e^{-\frac{x^2}{2}-\frac{y^2}{2}} + e^{-\frac{(x-1)^2}{2}-\frac{(y-1)^2}{2}}} = \frac{1}{1+e^{x+y-1}},$$

which is equal to $\sigma(\boldsymbol{\beta} \cdot \mathbf{x} + \alpha)$ if $\alpha = 1$ and $\boldsymbol{\beta} = [-1,-1]$. So the assumption was correct in this example.

Exercise 2.2.1

Consider a binary classification problem for which the regression function r satisfies $r(\mathbf{x}) = \sigma(\boldsymbol{\beta} \cdot \mathbf{x} + \alpha)$ for some $\boldsymbol{\beta} \in \mathbb{R}^p$ and $\alpha \in \mathbb{R}$. Show that the decision boundary is linear.

Exercise 2.2.1 shows that directly applying logistic regression always yields linear decision boundaries. However, we can use logistic regression to find nonlinear decision boundaries by appending components to the feature vectors which are derived from the original features. For example, if we apply the map $[x_1, x_2] \mapsto [x_1, x_2, x_1^2, x_2^2, x_1 x_2]$ to each feature vector, then the linear boundary we discover in $\mathbb{R}^5$ will correspond to a quadric curve* in the original feature space $\mathbb{R}^2$.

* *Quadric* means "described by a quadratic polynomial". Quadric curves are also known as conic sections (ellipses, parabolas, hyperbolas).

2.3 Support vector machines

Logistic regression identifies linear decision boundaries by modeling the regression function $r(\mathbf{x}) = \mathbb{P}(Y = 1 \,|\, \mathbf{X} = \mathbf{x})$. In this section, we will find decision boundaries directly based on desirable features of those boundaries. We begin with a warm-up exercise on the geometry of planes in $\mathbb{R}^3$.

Example 2.3.1

Find the distance from the plane $3x + 2y + z = 6$ to the point $P = (4, 7, 1)$.

Solution

The vector $[3, 2, 1]$ is perpendicular to the given plane, so moving t units directly away from some point in the plane means adding the vector

$$t\frac{[3,2,1]}{|[3,2,1]|} = t\frac{[3,2,1]}{\sqrt{14}}$$

to that point. The value of the function $3x + 2y + z$ is 6 for any point in the plane, and adding $t\frac{[3,2,1]}{\sqrt{14}}$ increases that value by $t\sqrt{14}$:

$$3\left(x + \frac{3t}{\sqrt{14}}\right) + 2\left(y + \frac{2t}{\sqrt{14}}\right) + 1\left(z + \frac{t}{\sqrt{14}}\right) = 3x + 2y + z + \frac{9t + 4t + t}{\sqrt{14}} = 6 + t\sqrt{14}.$$

Since the value of $3x + 2y + z$ is equal to $3(4) + 2(7) + 1(1) = 27$ at the point P, it changes by $27 - 6 = 21$ as you move from the nearest point on the plane to P. We solve $t\sqrt{14} = 21$ to find that the distance from the plane to the point is $\boxed{21/\sqrt{14}}$.

Example 2.3.2

Find the distance from the hyperplane $\{\mathbf{x} \in \mathbb{R}^n \,:\, \boldsymbol{\beta} \cdot \mathbf{x} - \alpha = 0\}$ to the point $\mathbf{x} \in \mathbb{R}^n$.

Solution

Generalizing the idea we developed in the previous problem, we can say that $\boldsymbol{\beta}$ is the normal vector to the hyperplane, and moving t units directly away from the hyperplane corresponds to adding $t|\boldsymbol{\beta}|$ to the value of $\boldsymbol{\beta} \cdot \mathbf{x} - \alpha$. Therefore, we can check the value of the function $\mathbf{x} \mapsto \boldsymbol{\beta} \cdot \mathbf{x} - \alpha$ at our point $\mathbf{x}$ and divide by $|\boldsymbol{\beta}|$ to find the distance from $\mathbf{x}$ to the plane. So our distance formula is $\frac{|\boldsymbol{\beta} \cdot \mathbf{x} - \alpha|}{|\boldsymbol{\beta}|}$.

Example 2.3.3

Simulate data for a binary classification problem in the plane for which the two classes can be separated by a line. Write a Julia function for finding the thickest slab which separates the two classes.

Solution

Suppose that sample points are $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $y_i \in \{-1, 1\}$ for each $1 \leq i \leq n$. Let us describe the separating slab as $\{\mathbf{x} \in \mathbb{R}^2 : -1 \leq \boldsymbol{\beta} \cdot \mathbf{x} - \alpha \leq 1\}$. The width of this slab is $2/|\boldsymbol{\beta}|$, by Example 2.3.2.

We can check whether a point is on the correct side of the slab by checking whether $\boldsymbol{\beta} \cdot \mathbf{x} - \alpha \geq 1$ for points of class 1 and less than or equal to -1 for points of class -1. More succinctly, we can check whether $y_i(\boldsymbol{\beta} \cdot \mathbf{x}_i - \alpha) \geq 1$ for all $1 \leq i \leq n$.

So, we are looking for the values of $\boldsymbol{\beta}$ and α which minimize $|\boldsymbol{\beta}|$ subject to the conditions $y_i(\boldsymbol{\beta} \cdot \mathbf{x}_i - \alpha) \geq 1$ for all $1 \leq i \leq n$. This is a **constrained** optimization problem, since we are looking to maximize the value of a function over a domain defined by some constraining inequalities.

Constrained optimization is a ubiquitous problem in applied mathematics, and numerous solvers exist for them. Most of these solvers are written in low-level languages like C and have bindings for the most popular high-level languages (Julia, Python, R, etc.). No solver is uniformly better than the others, so solving a constrained optimization problem often entails trying different solvers to see which works best for your problem.

Julia has a package which makes this process easier by providing a single interface for problem encoding (`JuMP`). Let's begin by sampling our points and looking at a scatter plot. We go ahead and load the `Ipopt` package, which provides the solver we'll use.

Julia

```
using Plots, JuMP, Ipopt, Random; Random.seed!(1234);
gr(aspect_ratio=1)
```

Let's sample the points from each class from a multivariate normal distribution. We'll make the mean of the second distribution a parameter of the sampling function so we can change it in the next example.

```
function samplepoint(μ=[3,3])
    class = rand([-1,1])
    if class == -1
        X = randn(2)
    else
        X = μ + [1 -1/2; -1/2 1] * randn(2)
    end
    (X,class)
end
n = 100
samples = [samplepoint() for i=1:n]
ys = [y for (x,y) in samples]
scatter([(x₁,x₂) for ((x₁,x₂),y) in samples],group=ys)
```

Next we describe our constrained optimization problem in `JuMP`. The data for the optimization is stored in a `Model` object, and the solver can be specified when the model is instantiated.

```
m = Model(solver=IpoptSolver(print_level=0))
```

We add variables to the model with the `@variable` macro,* and we add constraints with the `@constraint` macro. We add the function we want to minimize or maximize with the `objective` macro.

* A macro is like a function, except that it gets to operate directly on the expressions it's given rather than the values returned by those expressions.

```
@variable(m,β[1:2]) # adds β[1] and β[2] at the same time
@variable(m,α)
for (x,y) in samples
    @constraint(m,y*(βx - α) ≥ 1)
end
@objective(m,Min,β[1]^2+β[2]^2)
```

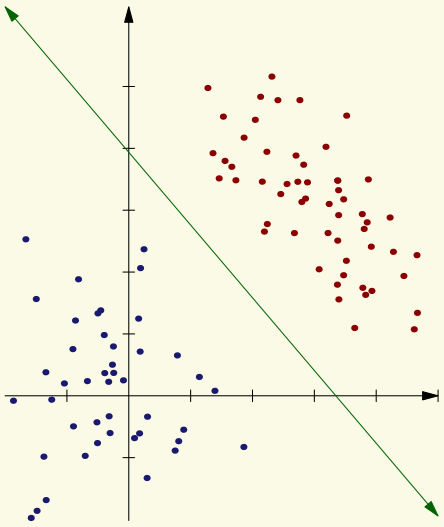
When we call **solve** on the model, it makes the optimizing values of the variables retrievable via **getvalue**.

```
solve(m)
β,α = getvalue(β), getvalue(α)
```

Now we can plot our separating line.

```
l(x₁) = (α - β[1]*x₁)/β[2]
xs = [-2,5]
plot!(xs,[l(x₁) for x₁ in xs],label="")
```

We can see that there are necessarily sample points of each class which lie on the boundary of the separating slab. These samples are called **support vectors**. If a sample is not a support vector, then moving it slightly or removing it does not change the separating hyperplane we find.



Example 2.3.3 is called the **hard-margin** SVM (support vector machine). A different approach (the *soft-margin* SVM) is required if the training samples for the two classes are not separable by a hyperplane.

Example 2.3.4

Consider a binary classification problem in $\mathbb{R}^2$ for which the training samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ are not separable by a line. Explain why*

$$L(\boldsymbol{\beta}, \alpha) = \lambda|\boldsymbol{\beta}|^2 + \frac{1}{n}\sum_{i=1}^{n}\left[1 - y_i(\boldsymbol{\beta}\cdot\mathbf{x}_i - \alpha)\right]_+$$

is a reasonable quantity to minimize.

* Note: u_+ means $\max(0, u)$, and $\lambda > 0$ is a parameter of the loss function.

Solution

Minimizing the first term $\lambda|\boldsymbol{\beta}|^2$ is the same as minimizing the value of $|\boldsymbol{\beta}|$, which was the *hard-margin* objective function we saw in Example 2.3.3. The second term penalizes points which are not on the right side of the slab (in units of margin widths: points on the slab midline get a penalty of 1, on the

wrong edge of the slab they get a penalty of 2, and so on). Thus this term imposes misclassification penalty. The parameter λ governs the tradeoff between the large-margin incentive of the first term and the correctness incentive of the second.

Example 2.3.5: Soft-margin SVM

Simulate some overlapping data and minimize the loss function given in Example 2.3.4. Select λ by leave-one-out cross-validation.

Solution

We begin by generating our samples. To create overlap, we make the mean of the second distribution closer to the origin.

```
samples = [samplepoint([1,1]) for i=1:n]
ys = [y for (x,y) in samples]
scatter([(x₁,x₂) for ((x₁,x₂),y) in samples],group=ys)
```

Next we define our loss function, including a version that takes $\boldsymbol{\beta}$ and α together in a single vector called `params`.

```
L(λ,β,α,samples) = λ*norm(β)^2 + 1/n*sum(max(0,1-y*(βx - α)) for (x,y) in samples)
L(λ,params,samples) = L(λ,params[1:end-1],params[end],samples)
```

Since this optimization problem is unconstrained, we can use `Optim` to do the optimization. We define a function `SVM` which returns the values of $\boldsymbol{\beta}$ and α which minimize the empirical loss:

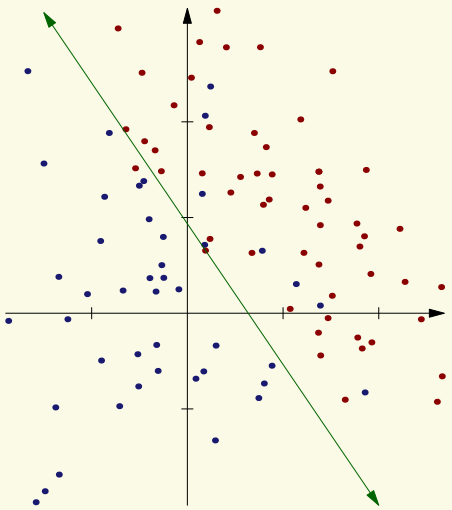
```
using Optim
function SVM(λ,samples)
    params = optimize(params->L(λ,params,samples),ones(3),BFGS()).minimizer
    params[1:end-1], params[end]
end
```

To choose λ, we write some functions to perform cross-validation:

```
function errorrate(β,α,samples)
    count(y*(βx-α) < 0 for (x,y) in samples)
end
function CV(λ,samples,i)
    β,α = SVM(λ,[samples[1:i-1];samples[i+1:end]])
    x,y = samples[i]
    y*(βx - α) < 0
end
function CV(λ,samples)
    mean(CV(λ,samples,i) for i=1:length(samples))
end
```

Finally, we optimize over λ:

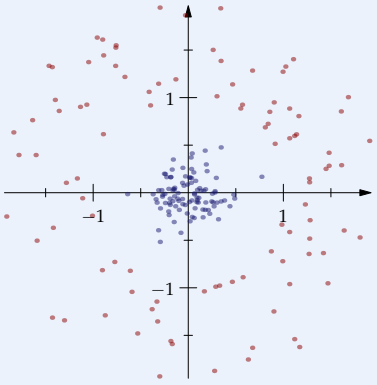
```
λ₁, λ₂ = 0.001, 0.25
λ = optimize(λ->CV(first(λ),samples),
             λ₁,λ₂).minimizer
β,α = SVM(λ,samples)
l(x₁) = (α - β[1]x₁)/β[2]
xs = [-1.5,2]
plot!(xs,[l(x₁) for x₁ in xs],label="")
```



Example 2.3.6

Support vector machines can be used to find nonlinear separating boundaries, because we can map the feature vectors into a higher-dimensional space and use a support vector machine find a separating hyperplane in that space.

Find a map from $\mathbb{R}^2$ to a higher dimensional space such that the two classes shown in the figure can be separated with a hyperplane.

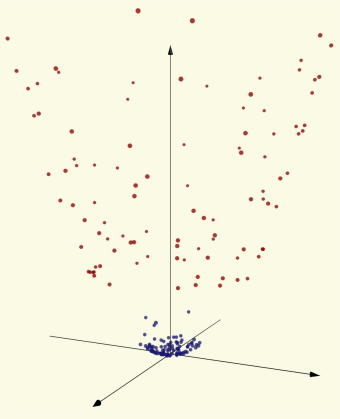


Solution

The distinguishing feature of the points is distance from the origin, so we supplement the features x_1 and x_2 with the combination $x_1^2 + x_2^2$. So our map is

$$\phi(x_1, x_2) = (x_1, x_2, x_1^2 + x_2^2).$$

We can see that the points can be separated by a plane in $\mathbb{R}^3$, so we could do a hard-margin SVM at this stage. To classify a test point (x_1, x_2), we can just check which side of the plane the point $\phi(x_1, x_2)$ is on.



There is a method for doing SVM classification without computing or *even knowing* the function ϕ. You can

use some optimization theory to re-express the parameters of the optimal hyperplane using some formulas for which the only expressions involving ϕ are of the form $\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$. You can then work out how to calculate $\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$ directly in terms of $\mathbf{x}_i$ and $\mathbf{x}_j$; this function $K(\mathbf{x}_i, \mathbf{x}_j)$ is called the **kernel** associated with ϕ.

Once you know K, you don't need to think about ϕ at all. In fact, at this stage you can use whatever kernel function K you want in the SVM formulas. One popular one is called the **radial basis function**, which is defined by $K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{|\mathbf{x}_i - \mathbf{x}_j|^2}{2\sigma^2}}$, where σ is a parameter. We will skip the derivation for the kernel trick, but in practice you will be able to use built-in library functions to try common kernels or specify your own.

2.4 Neural networks

In this section we will build prediction functions which are defined as compositions of simple functions. Using linear* functions as our building blocks is a dead end:

* more precisely, *affine*

Example 2.4.1

An **affine function** from $\mathbb{R}^t$ to $\mathbb{R}^s$ is a function of the form $\mathbf{x} \mapsto W\mathbf{x} + \mathbf{b}$, where W is an $s \times t$ matrix and $\mathbf{b} \in \mathbb{R}^t$. Show that a composition of affine functions is affine.

Solution

We have $W_2(W_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 = W_2W_1\mathbf{x} + (W_1\mathbf{b}_1 + \mathbf{b}_2)$, which is of the form (matrix times $\mathbf{x}$ plus vector).

By Example 2.4.1, composing affine functions does not yield any new functions. We will introduce nonlinearity by applying a fixed function $K : \mathbb{R} \to \mathbb{R}$ componentwise after each affine map application. We call K the **activation** function. The modern default activation to use is the **ReLU** function $K(x) = \max(0, x)$. We borrow some Julia syntax and write $K.$ for the function which applies K pointwise:

$$K.([x_1, \ldots, x_t]) = [K(x_1), \ldots, K(x_t)].$$

Given an affine map $\mathbf{x} \mapsto W\mathbf{x} + \mathbf{b}$, we call W the **weight** matrix and $\mathbf{b}$ the **bias** vector.

Example 2.4.2

Suppose that $A_1(\mathbf{x}) = \left[\begin{smallmatrix} 3 & -2 \\ 1 & 4 \end{smallmatrix}\right]\mathbf{x} + \left[\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right]$ and $A_2(\mathbf{x}) = \left[\begin{smallmatrix} -4 & 0 \\ 3 & 1 \end{smallmatrix}\right]\mathbf{x} + \left[\begin{smallmatrix} -2 \\ 2 \end{smallmatrix}\right]$. Find $(A_2 \circ K. \circ A_1)\left(\left[\begin{smallmatrix} -2 \\ -4 \end{smallmatrix}\right]\right)$, where K is the ReLU activation function.

Solution

We have $A_1(\mathbf{x}) = \left[\begin{smallmatrix} 3 \\ -17 \end{smallmatrix}\right]$. Applying K to each component yields $\left[\begin{smallmatrix} 3 \\ 0 \end{smallmatrix}\right]$. Finally, applying A_2 yields

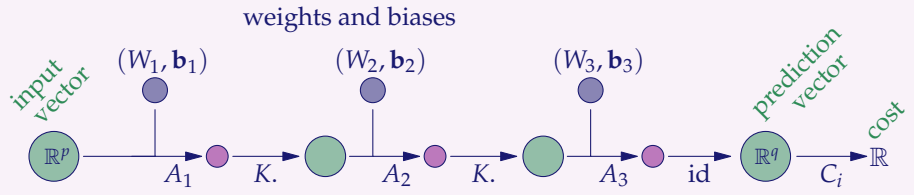
$$\left[\begin{smallmatrix} -4 & 0 \\ 3 & 1 \end{smallmatrix}\right]\left[\begin{smallmatrix} 3 \\ 0 \end{smallmatrix}\right] + \left[\begin{smallmatrix} -2 \\ 2 \end{smallmatrix}\right] = \boxed{\left[\begin{smallmatrix} -14 \\ 11 \end{smallmatrix}\right]}.$$

We will use a diagram to visualize our neural net as a composition of maps. We include the sequence of alternating affine and activation maps, and we also include one final map which associates a real-valued *cost* with each output vector. Given a training sample $(\mathbf{x}_i, \mathbf{y}_i)$, let's consider the cost $C_i(\mathbf{y}) = |\mathbf{y} - \mathbf{y}_i|^2$ (which measures squared distance from the vector $\mathbf{y}$ output by the neural net and the desired vector $\mathbf{y}_i$). Our goal will be to find values for the weights and biases which yield a small average value for $C(N(\mathbf{x}_i), \mathbf{y}_i)$ as $(\mathbf{x}_i, \mathbf{y}_i)$ ranges over the set of training samples.

Definition 2.4.1

A **neural network** function* $N : \mathbb{R}^p \to \mathbb{R}^q$ is a composition of affine transformations and componentwise applications of a function $K : \mathbb{R} \to \mathbb{R}$.

* The type of neural network we study here is called a *multilayer perceptron* in the literature.



The **architecture** of a neural network is the sequence of dimensions of the domains and codomains of its affine maps.

* Contrast this scenario with logistic regression, where the same minimizing parameters will be found by any reasonable minimization algorithm.

In principle, we have fully specified a neural network learner: given a set of training samples and a choice of architecture, we can ask for the weights and biases which minimize the average cost over the training samples. However, this is not the end of the story, for two reasons: neural net cost minimization is not a convex problem, and finding a global minimum is typically not feasible. Furthermore, even if the global minimum can be found, it is typically overfit. Therefore, building an effective neural net requires finessing not only the setup (architecture, choice of activation function, choice of cost function) but also the details of the optimization algorithm.*

* The "desired nudge" for a quantity means "$-\epsilon$ times the derivative of the cost function with respect to that quantity", where ϵ is the gradient descent learning rate.

Neural networks are typically trained using *stochastic gradient descent*. The idea is to determine for each training sample the desired nudges* to each weight and bias to reduce the cost for that sample. Rather than performing this calculation for every sample in the training set (which is, ideally, enormous), we do it for a randomly selected subset of the training data.

The main challenge in implementing stochastic gradient descent is the bookkeeping associated with all the nodes in the neural net diagram. We will use matrix differentiation to simplify that process. Let's begin by defining a neural net data type and writing some basic methods for it.

Example 2.4.3

Create data types in Julia to represent affine maps and neural net functions. Write an `architecture` method for neural nets which returns the sequence of dimensions of the domains and codomains of its affine maps.

Solution

We supply a `NeuralNet` with the sequence of affine maps, the activation function, and also the activation function's derivative. We write call methods for the `AffineMap` and `NeuralNet` types so they can be applied as functions to appropriate inputs. (One of these methods refers to a function we will

define in the next example.)

Julia

```
struct AffineMap
    W::Matrix
    b::Vector
end
struct NeuralNet
    maps::Vector{AffineMap}
    K::Function # activation function
    K̇::Function # derivative of K
end
(A::AffineMap)(x) = A.W * x + A.b
(N::NeuralNet)(x) = forwardprop(N,x)[end]
architecture(N::NeuralNet) = [[size(A.W,2) for A in N.maps]; size(last(N.maps).W,1)]
```

Successively applying the maps in the neural network is called **forward propagation**.

Example 2.4.4

Write a Julia function `forwardprop` which calculates the sequence of vectors obtained by applying each successive map in the neural net.

Solution

We store the sequence of values we obtain in an array called `vectors`. The very last map is the identity function rather than $K_\cdot$, so it must be handled separately.

```
function forwardprop(N::NeuralNet,x)
    vectors = [x]
    for (j,A) in enumerate(N.maps)
        push!(vectors,A(vectors[end]))
        push!(vectors,(j < length(N.maps) ? N.K : identity).(vectors[end]))
    end
    vectors
end
```

To adjust the weights and biases of the neural net in a favorable direction, we want to compute for every node v the derivative of the value in the cost node with respect to the value in node v. The node which is easiest to compute the derivative for is the prediction vector node, since the only map between that node and the cost is C_i.

More generally, given the derivative value for any particular node* v, we can calculate the derivative for the node v_{left} immediately to its left, using the chain rule. A small change $d\mathbf{u}$ in the value $\mathbf{u}$ at node v_{left} induces a small change $\frac{\partial S}{\partial \mathbf{u}} d\mathbf{u}$ in the value $\mathbf{v}$ at node v (where S denotes the map between v and v_{left}). This change in turn induces a change $\frac{\partial T}{\partial \mathbf{v}} \frac{\partial S}{\partial \mathbf{u}} d\mathbf{u}$ in the cost value (where T denotes the map from v to the cost node). In other words, the net result of the small change $d\mathbf{u}$ is the product of the two derivative matrices and $d\mathbf{u}$.

* again, the derivative of the *cost* with respect to the value at that node

So we can work from right to left in the diagram and calculate the derivative values for all of the nodes. This is called **backpropagation**. We will just need to calculate the derivatives of the maps between adjacent nodes.

Example 2.4.5

(a) Find the derivative of $C_i(\mathbf{y}) = |\mathbf{y} - \mathbf{y}_i|^2$ with respect to $\mathbf{y}$.

(b) Find the derivative of K. (the map which applies K pointwise).

(c) Find the derivative of $W\mathbf{u} + \mathbf{b}$ with respect to $\mathbf{u}$.

Solution

(a) The derivative of C_i is

$$\frac{\partial}{\partial \mathbf{y}} \left[(\mathbf{y} - \mathbf{y}_i)'(\mathbf{y} - \mathbf{y}_i) \right] = 2(\mathbf{v} - \mathbf{y}_i)',$$

by the product rule.

(b) The derivative with respect to $\mathbf{x}$ has (i, j)th entry $\frac{\partial (K.(\mathbf{x})_i)}{\partial x_j}$, which is equal to 0 if $i \neq j$ and $\dot{K}(x_i)$ if $i = j$. In other words, the derivative of $K.$ with respect to $\mathbf{x}$ is $\operatorname{diag} \dot{K}.(\mathbf{x})$.

(c) The derivative of $W\mathbf{u} + \mathbf{b}$ with respect to $\mathbf{u}$ is $\frac{\partial (W\mathbf{u})}{\partial \mathbf{u}} + \frac{\partial \mathbf{b}}{\partial \mathbf{u}} = W + 0 = W$.

Example 2.4.6

Write a Julia function `backprop` which calculates the for each node the derivative of the cost function with respect to the value at that node. In other words, calculate the derivative of the composition of maps between that node and the cost node at the end.

Solution

We can use all the derivatives we calculated in Example 2.4.5. We define functions which return the index of the jth green node and the jth purple node in the diagram, for convenience.*

```
greennode(j) = 2j-1
purplenode(j) = 2j
function backprop(N::NeuralNet,vectors,yᵢ)
    grads = [2(vectors[end]-yᵢ)']
    push!(grads,grads[end])
    for j = length(N.maps) : -1 : 1
        push!(grads,grads[end] * N.maps[j].W)
        j > 1 && push!(grads,grads[end] * Diagonal(N.K̇.(vectors[purplenode(j-1)])))
    end
    reverse(grads)
end
```

* Note the use of the *short-circuit conditional*: `&&` only evaluates the second expression if the first evaluates to `false`, and we can use this feature for control flow.

With the derivative values computed for all of the nodes on the bottom row, we can calculate the derivative of the cost function with respect to the weights and biases. To find changes in the cost function with respect

to changes in a weight matrix W_j, we need to introduce the idea of differentiating with respect to a matrix.

Exercise 2.4.1

Given $f : \mathbb{R}^{m\times n} \to \mathbb{R}$, we define

$$\frac{\partial}{\partial W} f(W) = \begin{bmatrix} \frac{\partial f}{\partial a_{1,1}} & \cdots & \frac{\partial f}{\partial a_{1,n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial a_{m,1}} & \cdots & \frac{\partial f}{\partial a_{m,n}} \end{bmatrix},$$

where $a_{i,j}$ is the entry in the ith row and jth column of W. Suppose that $\mathbf{u}$ is a $1 \times m$ row vector and $\mathbf{v}$ is an $n \times 1$ column vector. Show that

$$\frac{\partial}{\partial W}(\mathbf{u}W\mathbf{v}) = \mathbf{u}'\mathbf{v}'.$$

If $\mathbf{f}$ is a vector-valued function of a matrix W, then $\frac{\partial}{\partial W}(\mathbf{f}(\mathbf{v}))$ is "matrix" whose (i,j,k)th entry is $\frac{\partial f_i}{\partial a_{j,k}}$. As suggested by the scare quotes, this is not really a matrix, since it has three varying indices instead of two.* This is called a **third-order tensor**, but we will not develop this idea further, since the only property we will need is suggested by the notation and the result of Exercise 2.4.1: differentiating $W\mathbf{v}$ with respect to W and left-multiplying by a row vector $\mathbf{u}$ has the following net effect:

$$\mathbf{u}\frac{\partial}{\partial W}(W\mathbf{v}) = \frac{\partial}{\partial W}(\mathbf{u}W\mathbf{v}) = \mathbf{u}'\mathbf{v}'.$$

* The analogue of the entry-based matrix notation $\left[\begin{smallmatrix}1&2\\3&4\end{smallmatrix}\right]$ would require a 3D box instead of a rectangle.

Example 2.4.7

Write two functions `weight_gradients` and `bias_gradients` which compute the derivatives with respect to W_j and $\mathbf{b}_j$ of $C_i(N(\mathbf{x_i}))$.

Solution

In each case, we calculate the derivative of the map to the next node (either $W_j \mapsto W_j\mathbf{x} + \mathbf{b}_j$ or $\mathbf{b} \mapsto W_j\mathbf{x} + \mathbf{b}_j$) and left-multiply by the derivative of the cost function with respect to the value at that node. The derivative of $W_j\mathbf{x} + \mathbf{b}_j$ with respect to $\mathbf{b}_j$ is the identity matrix, and by Exercise 2.4.1, differentiating $W\mathbf{v}$ with respect to W and left-multiplying by $\mathbf{u}$ yields $\mathbf{u}'\mathbf{v}'$:

```
function weight_gradients(N::NeuralNet,vectors,grads)
    [grads[purplenode(j)]' * vectors[greennode(j)]' for j=1:length(N.maps)]
end

function bias_gradients(N::NeuralNet,vectors,grads)
    [grads[purplenode(j)]' for j=1:length(N.maps)]
end
```

We are now set up to train the neural network.

Example 2.4.8

Write a function **train** which performs stochastic gradient descent: for a randomly chosen subset of the training set, determine the average desired change in weights and biases to reduce the cost function. Update the weights and biases accordingly and perform a specified number of iterations.

Your function should take 7 arguments: (1) desired architecture, (2) the activation function K, (3) the derivative $\dot{K}$ of the activation function K, (4) an array of training samples, (5) the batch size (the number of samples to use in each randomly chosen subset used in a single stochastic gradient descent iteration), (6) the learning rate ϵ, and (7) the desired number of stochastic gradient descent iterations.

Solution

We write a function which calculates the average suggested changes in the weights and biases and call it from inside the **train** function.

```
function averageweightΔ(N::NeuralNet,samples,batch,ε)
    arch = architecture(N)
    layers = length(arch)-1
    sum_Δweight = [zeros(arch[i+1],arch[i]) for i=1:layers]
    sum_Δbias = [zeros(arch[i+1]) for i=1:layers]
    for k in batch
        x, y = samples[k]
        vectors = forwardprop(N,x)
        grads = backprop(N,vectors,y)
        ΔWs = -ε*weight_gradients(N,vectors,grads)
        Δbs = -ε*bias_gradients(N,vectors,grads)
        for i=1:layers
            sum_Δweight[i] += ΔWs[i]
            sum_Δbias[i] += Δbs[i]
        end
    end
    (sum_Δweight, sum_Δbias) ./ length(batch)
end
```

Now we can write **train**. We initialize the biases to 0.1, and the affine map entries are sampled independently from the standard normal distribution.

```
function train(arch,K,K̇,samples,batchsize,ε = 0.1,iterations=1000)
    random_maps = [AffineMap(randn(arch[i+1],arch[i]),
                             fill(0.1,arch[i+1])) for i=1:length(arch)-1]
    N = NeuralNet(random_maps,K,K̇)
    for i=1:iterations
        batch = sample(1:length(samples),batchsize)
        meanΔweight, meanΔbias = averageweightΔ(N,samples,batch,ε)
        N = NeuralNet([AffineMap(A.W .+ ΔW, A.b .+ Δb)
                       for (A,ΔW,Δb) in zip(N.maps,meanΔweight,meanΔbias)],K,K̇)
    end
    N
end
```

Example 2.4.9

Try training your model on some data which are sampled by taking $\mathbf{X}$ uniform in the unit square and setting $Y = 1 - |\mathbf{X}|^2$.

Solution

We begin by defining the ReLU activation and a cost function.

```
K(x) = x > 0 ? x : 0
K̇(x) = x > 0 ? 1 : 0
cost(N::NeuralNet,samples) = mean(norm(N(x)-y)^2 for (x,y) in samples)
```

Next, we sample our data.

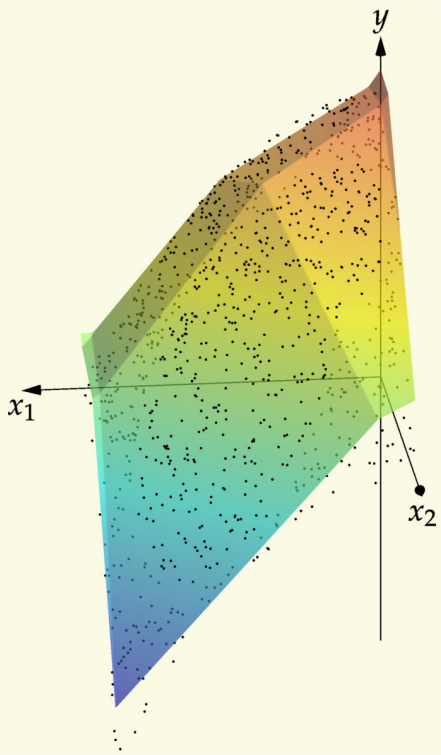
```
using Random; Random.seed!(123)
xs = [rand(2) for i=1:1000]
ys = [[1-norm(x)^2] for x in xs]
samples = collect(zip(xs,ys))
```

Then we choose an architecture, a batch size, and a learning rate, and we train our model on the data:

```
arch = [2,5,1]
batchsize = 100
ε = 0.005
N = train(arch,K,K̇,samples,batchsize,ε,10_000)
```

Finally, we inspect the result.*

```
cost(N,samples) # returns 0.00343
xgrid = 0:1/2^8:1
ygrid = 0:1/2^8:1
zs = [first(N([x,y])) for x=xgrid,y=ygrid]
using Plots; plotly(); surface(xgrid,ygrid,zs)
```



* We switch to the plotly backend, since its surface plotting is much better than that of the default backend GR.

We can see that the resulting graph of N fits the points reasonably well. We can see that the graph is piecewise linear, with the creases between the linear pieces corresponding to points where of the affine maps in the net returns zero.

See here for an animation of the training of this neural network and here for an interactive version of the figure.

2.4.1 Neural nets for classification

So far in this section we have discussed neural networks in a regression context. We can use essentially the same framework for classification, with the following modifications. We encode each sample classification $\mathbf{y}_i$ as a vector in $\mathbb{R}^{|\mathcal{Y}|}$ with a 1 in the position corresponding to its class and with zeros elsewhere. Furthermore, in the neural network diagram, we replace the identity map (whose output is the prediction vector) with the **softmax** function $\mathbf{u} \mapsto \left[\frac{e^{u_j}}{\sum_{k=1}^{n} e^{u_k}}\right]_{j=1}^{|\mathcal{Y}|}$ from $\mathbb{R}^{|\mathcal{Y}|}$ to $\mathbb{R}^{|\mathcal{Y}|}$. By applying this function, we ensure that the entries

of the prediction vector are positive and sum to 1, which means that we can interpret the components of $\mathbf{y}$ as probabilities for each class. A confident prediction will assign a number close to 1 in one position and small numbers elsewhere.

Finally, we perform stochastic gradient descent using the cost functions $C_i(\mathbf{y}) = -\log(\mathbf{y} \cdot \mathbf{y}_i)$. This function returns zero if the output vector assigns probability 1 to the correct class, and it applies a large penalty if the $\mathbf{y}$ assigns a small probability to the correct class.

Example 2.4.10

Find the derivative of $C_i(\text{softmax}(\mathbf{u}))$ respect to $\mathbf{u}$.

Solution

Suppose that j is the correct class for the ith sample. Then

$$-\log\left(\frac{e^{u_j}}{\sum_{k=1}^{n} e^{u_k}}\right) = -u_j + \log\sum_{k=1}^{n} e^{u_k}.$$

Differentiating with respect to u_j yields

$$-1 + \frac{e^{u_j}}{\sum_{k=1}^{n} e^{u_k}},$$

and differentiating with respect to u_ℓ for $\ell \neq j$ yields

$$\frac{e^{u_\ell}}{\sum_{k=1}^{n} e^{u_k}}.$$

Therefore, the derivative with respect to $\mathbf{u}$ is $-\mathbf{y}_i' + \text{softmax}(\mathbf{u})'$.

2.5 Dimension reduction

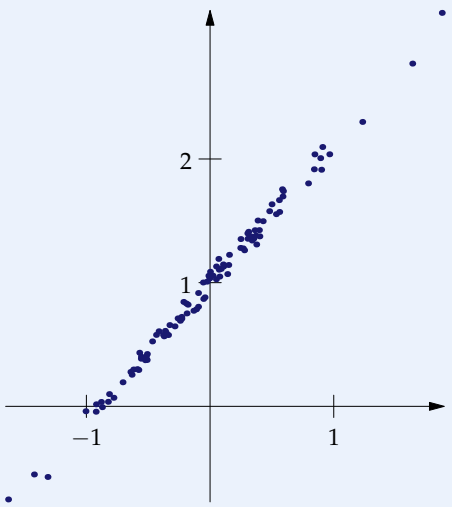
2.5.1 Principal component analysis

Real-world data sets typically have many more than 2 or 3 features, which rules out direct use of the visualizations we've used for the examples we've seen so far. However, we can often apply a map from $\mathbb{R}^p$ to a low-dimensional space which preserves much of the structure of the original data set.

Let's begin with an example where both the original space and the reduced space are low-dimensional.

Example 2.5.1

Finding a map ϕ from $\mathbb{R}^2$ to $\mathbb{R}$ such that each point (x, y) shown can be reasonably accurately located if the value of $\phi(x, y)$ is known. You may assume that the coordinates of the 100 points are stored in a 100×2 matrix A.



Solution

Consider the line $y = mx + b$ which, roughly speaking, runs along the primary axis of the ellipse-shaped point cloud. If we know how far along this line one of the points is, then we know pretty accurately where it's located.

More precisely, for each point, we let $\phi(x, y)$ be the orthogonal projection of (x, y) onto the line $y = mx + b$. We would like for the average squared error associated with the approximation $\phi(x, y) \approx (x, y)$ to be as small as possible.

We solved a very similar problem when we studied the singular value decomposition: the line through the origin which minimizes the sum of squared distances is the first column of V in the singular value decomposition $U\Sigma V'$ of A.

The problem with applying that idea directly is that the best line clearly does not pass through the origin. To deal with this issue, we find the mean of the coordinates of the points and subtract it from the coordinates of each point. Then the transformed point cloud will be centered at the origin, and we can perform SVD on this new matrix to identify the line which minimizes the sum of squared distances to the points. Once the best line's unit vector $\mathbf{v}$ is known, we can determine for each point how far along the line its projection is by taking its dot product with $\mathbf{v}$. In other words, we can project all of the points onto $\mathbb{R}^1$ by right-multiplying the matrix A by $\mathbf{v}$.

The idea we developed in Example 2.5.1 is called **principal component analysis**. We subtract off the means to center the point cloud, apply the singular value decomposition, and consider the first k columns of the resulting V matrix (these columns are called the **principal components** of the feature matrix). The desired dimension reduction map from $\mathbb{R}^p$ to $\mathbb{R}^k$ is represented by the matrix `V[:,1:k]'`.

Example 2.5.2

Apply principal component analysis to project the handwritten digit images in the MNIST dataset to the plane.

Solution

We begin by loading the dataset and reshaping the training data feature matrix.

Julia

```
using MLDatasets, Images, Plots
features, labels = MNIST.traindata(Float64)
A = reshape(features[:],28^2,60_000)'
```

Next, we define a custom function for displaying images. This function accepts a vector of length $28^2 = 784$, it reshapes the entries into a square, and it returns an array of colors for display. If some of the components are negative, then negative and positive values are represented in different colors (red and blue, respectively). Otherwise, the image is displayed in grayscale.

```
function imshow(v)
    if any(v .< 0)
        (x -> x > 0 ? RGB(x,0,0) : RGB(0,0,-x)).(reshape(v./maximum(abs.(v)),(28,28))')
    else
        Gray.(reshape(v./maximum(abs.(v)),(28,28))')
    end
end
```

To perform principal component analysis, we take the column-wise mean with `mean(A,dims=1)` and subtract it from the matrix before performing the singular value decomposition.

```
U, Σ, V = svd(A .- mean(A,dims=1))
```

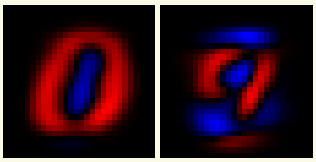


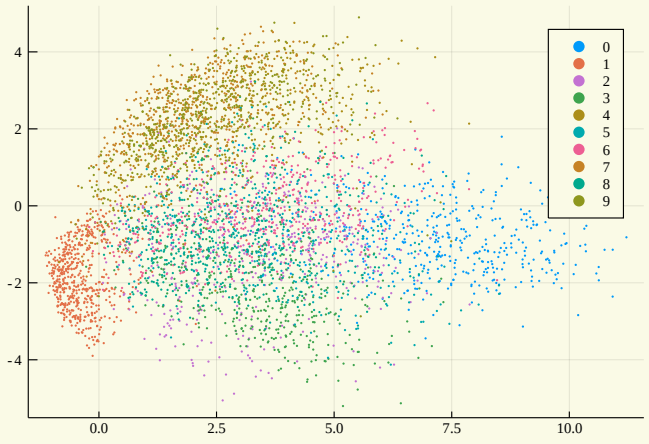
Figure 2.1 The first two principal components.

We can see the first principal component with `imshow(V[:,1])`, and similarly for the second principal component (Figure 2.1).*

* Exercise: reason from the principal component images about which digits should appear in the top/bottom or left/right in the figure below.

Finally, we project onto each of the first two principal components and make a scatter plot of the results. We set the marker size (`ms`) and the marker stroke width (`msw`) so we can see the colors of the points more clearly.

```
n = 5000
scatter(A[1:n,:]*V[:,1],
        A[1:n,:]*V[:,2],
        group=labels[1:n],
        ms=2,
        msw=0)
```



We can see that some digits cluster apart from the other digits (like 1), while others remain heavily overlapping.

2.5.2 Stochastic Neighbor Embedding

In this section we discuss a popular dimensionality reduction technique which is often more effective than principal component analysis. The idea is to choose a map which attempts to preserve *pairwise similarity* of the data points. The version we present is called t-SNE, which is short for *t-distributed stochastic neighbor embedding.*

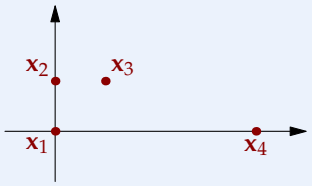
Suppose that $\mathbf{x}_1, \ldots, \mathbf{x}_n$ is a set of points in $\mathbb{R}^p$. We begin by fixing a parameter of the model, called the **perplexity** ρ. Given $\sigma > 0$, we define

$$P_{i,j}(\sigma) = \frac{e^{-|\mathbf{x}_i - \mathbf{x}_j|^2/(2\sigma^2)}}{\sum_{k \neq j} e^{-|\mathbf{x}_k - \mathbf{x}_j|^2/(2\sigma^2)}}$$

for $i \neq j$, and $P_{i,i}(\sigma) = 0$ for all $1 \leq i \leq n$. This quantity measures the similarity of distinct points $\mathbf{x}_i$ and $\mathbf{x}_j$: if $\mathbf{x}_i$ is closer to $\mathbf{x}_j$ (compared to how close other points are to $\mathbf{x}_j$), then $P_{i,j}$ is closer to 1. If $\mathbf{x}_i$ is far from $\mathbf{x}_j$, then $P_{i,j}$ is close to 0.

Example 2.5.3

Consider the points $\mathbf{x}_1 = [0,0]$, $\mathbf{x}_2 = [0,1]$, $\mathbf{x}_3 = [1,1]$, and $\mathbf{x}_4 = [4,0]$. Find $P_{2,1}(\sigma)$ for each value of σ in $\left\{\frac{1}{4}, 1, 2, 100\right\}$.



Solution

We define a function to compute $P_{i,j}(\sigma)$.

```
x = [[0,0],[0,1],[1,1],[4,0]]
f(x,y,σ) = exp(-norm(x-y)^2/(2σ^2))
P(x,i,j,σ) = f(x[i],x[j],σ) / sum(f(x[k],x[j],σ) for k=1:length(x) if k ≠ j)
```

We find that $P_{2,1}(0.25) = 0.9997$, $P_{2,1}(1) = 0.6222$, $P_{2,1}(2) = 0.4912$, and $P_{2,1}(100) = 0.3334$.

We can see from Example 2.5.3 that σ supplies the unit with respect to which proximity is being measured. If σ is very large, then all of the points are effectively close* to $\mathbf{x}_1$, so the values of $P_{i,1}(\sigma)$ are approximately equal for $i \in \{2,3,4\}$. If σ is very small, then $P_{i,1}(\sigma)$ is close to 1 for $\mathbf{x}_1$'s nearest neighbor and is close to 0 for the other points.

* meaning that their distance to $\mathbf{x}_1$ is a small number of σ's

For each j from 1 to n, we define σ_j to be the solution σ of the equation*

$$2^{-\sum_{i:i\neq j} P_{i,j}(\sigma)\log_2 P_{i,j}(\sigma)} = \rho. \tag{2.5.1}$$

* This step makes σ_j smaller for points $\mathbf{x}_j$ in denser regions and larger for points in sparser regions.

This ensures that the function $i \mapsto P_{i,j}(\sigma_j)$ avoids the extremes of too heavily concentrating its values on a small number of i's or spreading out its values too evenly across all of the i's.*

* The sum appearing in the exponent of (2.5.1) is called the **entropy** of the distribution $i \mapsto P_{i,j}(\sigma)$, and it measures how concentrated or diffuse the distribution is.

We also define the following symmetric version of the $P_{i,j}$'s: $p_{i,j} = \frac{1}{2n}(P_{i,j}(\sigma_j) + P_{j,i}(\sigma_i))$.

* This is where the t in t-SNE comes from: the distribution $\frac{1}{\pi}(1+x^2)^{-1}$ is an example of a t-distribution.

We will denote by $\mathbf{y}_1, \ldots, \mathbf{y}_n$ the locations of the *images* of the sample points under some map to a Euclidean space of a lower dimension k (typically 2 or 3). We define*

$$q_{i,j} = \frac{(1+|\mathbf{y}_i - \mathbf{y}_j|^2)^{-1}}{\sum_{k \neq j}(1+|\mathbf{y}_k - \mathbf{y}_j|^2)^{-1}}.$$

These quantities measure the pairwise similarity of the points $\mathbf{y}_1, \ldots, \mathbf{y}_n$, analogously to $P_{i,j}$. We measure how well the similarities $p_{i,j}$ match the similarities $q_{i,j}$ using the cost function

$$C(\mathbf{y}_1, \ldots, \mathbf{y}_n) = \sum_{(i,j):\, i \neq j} p_{i,j} \log_2 \frac{p_{i,j}}{q_{i,j}}.$$

Finally, we use gradient descent to find values for the image points $\mathbf{y}_1, \ldots, \mathbf{y}_n$ which minimize this cost function.

Example 2.5.4

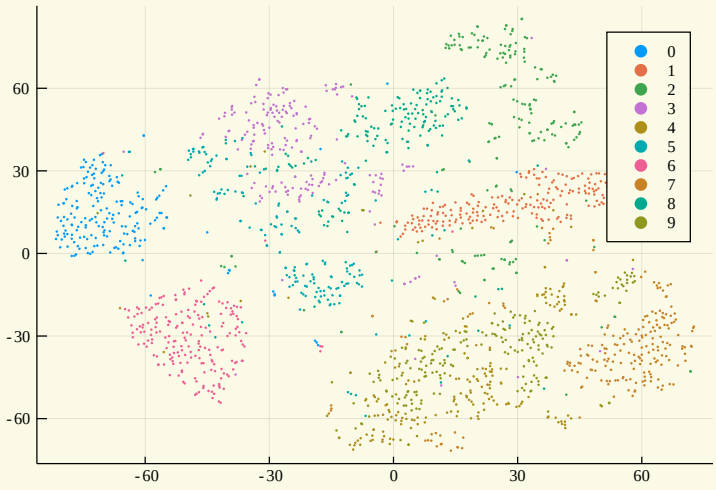
Use the Julia package `TSne` to plot a two-dimensional t-SNE embedding of the first 2000 images in the MNIST data set.

Solution

We call the `tsne` function on the first $k = 2000$ rows of the MNIST matrix `A` defined above.

```
using TSne, Random
Random.seed!(123)
n = 2000
Y = tsne(A[1:n,:])
scatter(Y[:,1],Y[:,2],
        group=labels[1:n],
        ms=2,msw=0)
```

We can see that this approach does a much better job of separating the classes than PCA did for this example.



3 Programming in Python and R

Python and R are the two most commonly used programming environments in data science.

1. **Python**. A popular general purpose language that has also seen widespread adoption in the statistics and machine learning space over the past decade following the introduction of powerful scientific computing packages (NumPy, SciPy, and scikit-learn).
2. **R**. The lingua franca in the statistics community for 15+ years; has extensive tooling available and over 10,000 statistics packages

3.1 Environment and workflow

You can download Python 3 from `anaconda.com/download` and R from `r-project.org`.

Python and R support the same modes of interaction as Julia:

1. *REPL*. Launch a read-eval-print loop from the command line. Any code you enter will be executed immediately, and any values returned by your code will be displayed. To start a session, run `python` or `r` from the terminal.
2. *Script*. Save a file called `example.py` or `example.r` and run `python example.py` or `rscript example.r` from the command line to execute all the code in the script.
3. *Notebook*. Like a REPL, but allows inserting text and math expressions, grouping code into blocks, etc. **Jupyter notebooks*** support Julia, Python, R and many other languages.
4. *IDE*. RStudio is an excellent IDE for R. There are many IDEs for Python, including PyCharm, Visual Studio Code, and Atom.

* Jupyter is a portmanteau of these three language names

3.2 Syntax differences from Julia

The syntax for variable assignent is slightly different in R:

Python

```
age = 41
```

R

```
age <- 41
```

Numerical values are floats by default in R; integer literals are specified with an L suffix, as in `10L`. Python behaves the same as Julia in this respect:

Python

```python
type(2.0) # float
type(2) # int
```

R

```r
typeof(2) # float
typeof(2L) # integer
```

The exponentiation operator is `^` in R and `**` in Python.

To find the length of a string: `nchar("hello")` in R; `len("hello")` in Python,

Concatenating strings: `"Hello " + "World"` , `paste("Hello ","World")`

In Python the boolean values are `True` and `False`, and R they are `TRUE` and `FALSE`. Boolean operators are the same in R as in Julia (`&&` `||` `!`), while in Python they are the corresponding English words: `and` `or` `not`

Blocks in Python are delineated using indentation, while R uses curly braces. Also, conditions in `if` statements are parenthesized in R.

Python

```python
if x > 0:
    print("x is positive")
elif x == 0:
    print("x is zero")
else:
    print("x is negative")
```

R

```r
if (x > 0) {
  print("x is positive")
}
else if (x == 0) {
  print("x is zero")
}
else {
  print("x is negative")
}
```

Python uses the keyword `def` for function definitions. Unlike in Julia, the `return` keyword is required in Python for the function to return a value.

Python

```python
def f(x,y):
    x2 = x * x
    return x2 + (y*x2+1)**(1/2)
```

In R, functions are defined by writing an *anonymous* function like `function(x) {x^2}` and binding it to a name:

R

```
f  <- function(x,y) {
  x2 <- x * x
  x2 + sqrt(y*x2+1)
}
```

In Python, lists are indexed from *0*.

Python

```
myList = [1,2,"a",[10,8,9]]
myList[2] # evaluates to "a"
myList[3][2] # evaluates to 9
2 in myList # evaluates to True
```

In R, the more commonly used data type for list-type operations is a *vector*, but R does have lists as well:

R

```
myList <-list(1,2,"a",list(10,8,9))
myList[3] # evaluates to "a"
myList[4][2] # evaluates to 8
2 %in% myList # evaluates to true
```

NumPy arrays may be defined by calling `np.array` on a Python list, and in R you use the function `c` to define a vector, as in `c(1,2,3)`.

Python

```
import numpy as np
import timeit
A = np.random.randn(10**6)
L = list(A)
n = 10**3
timeit.timeit("sum(L)","from __main__ import L",number=n)/n # 0.04 seconds
timeit.timeit("np.sum(L)","from __main__ import A, np",number=n)/n  # 0.0005 seconds
```

Python has list comprehensions just like Julia:

Python

```
perfect_squares = [n**2 for n in range(1,101)]
```

R doesn't have list comprehensions, but you can achieve a similar effect using `sapply`:

R

```
sapply(1:100,function(x) {return(x*x)})
```

1. Ranges: `range(1,11)` in Python, `1:10` in R. If we want only the odd integers, then we use `range(1,11,2)` in Python and `seq(1,10,2)` in R.
2. Concatenation: `[1,2,3] + [8,9,10]` in Python `c(c(1,2,3),c(8,9,10))` in R
3. Appending: `myList.append(3)` in Python, `myVector <- c(myVector,3)` in R.
4. Slicing*: `A[-3:]` in Python, `A[(length(A)-2):length(A)]` in R.
5. Finding the indices where a certain condition holds: `np.which(A > 2)` in Python, `which(A > 2)` in R.
6. Setting multiple array values: `A[:5] = 1` in Python, `A[1:5] <- 1` in R.

* These examples select the last three elements of `A`

The syntax for `while` and `for` blocks is entirely analogous to syntax for conditional blocks:

Python

```python
def isprime(n):
    for j in range(2,n):
        if n % j == 0:
            return False
    return True
```

R

```r
isprime <- function(n) {
  for (j in 2:(n-1)) {
    if (n %% j == 0) {
      return(FALSE)
    }
  }
  return(TRUE)
}
```

3.3 Package Ecosystems

The scientific computing tools in Python are organized around a relatively small number of fine-tuned and extensive packages. Some of the most important ones are:

1. **NumPy**. Fast multidimensional arrays; the basis for most scientific computing packages in Python.
2. **SciPy**. General purpose tools for scientific computing in Python.
3. **Matplotlib**. Standard plotting package in Python.
4. **Seaborn**. Data visualization library built on matplotlib.
5. **plotnine**. Clone of R's ggplot2.
6. **Pandas**. Data frames: structures for storing and manipulating tabular data).
7. **Scikit-Learn**. Machine learning.

The most commonly used suite of data analysis packages in R is Hadley Wickham's **tidyverse**.

1. **ggplot2**, for data visualisation. Graphics are built as a sum of *layers*, which consist of a data frame, a **geom**, a **stat**, and a mapping from the data to the geom's **aesthetics** (like `x`, `y`, `color`, or `size`). The appearance of the plot can be customized with `scale`s, `coord`s, and `theme`s.
2. **dplyr**, for data manipulation. The primary functions in this package filter rows, sort rows, select columns, add columns, group, and aggregate the columns of a grouped data frame:

```
flights %>%
  filter(month == 1, day < 5) %>%
  arrange(day, distance) %>%
  select(month, day, distance, air_time) %>%
  mutate(speed = distance / air_time * 60) %>%
  group_by(day) %>%
  summarise(avgspeed = mean(speed,na.rm=TRUE))
```

3. **tibble**, a variant of `data.frame` designed to work optimally with tidyverse packages

See *R for Data Science* (`r4ds.had.co.nz`) by Garrett Grolemund and Hadley Wickham to learn more about the tidyverse packages.

In Python (with the Anaconda installation), you can install a module using `conda install modulename` from the command line. A module (let's say NumPy) may be imported into a Python session or script using `import numpy as np`. Then functions and variables from that package can be referenced with the given abbreviation (for example, `np.array` makes an array).

In R, do `install.packages("modulename")` from an R session, and then `library(modulename)` to load the package. For example, you can run `install.packages("tidyverse")` to intall all of the tidyverse packages.

3.4 Data structures

Python's approach to data types is quite different from Julia's. The idea is to combine types and related functions into a single object called a **class**. For example:

PYTHON

```
class Album(object):
    def __init__(self,name,artist,year,songs):
        self.name = name
        self.artist = artist
        self.year = year
        self.length = length
```

The `__init__` function is called whenever a new `Album` is created:

JULIA/PYTHON

```
A = Album("Abbey Road","The Beatles",1969,"47:23")
```

The **attributes** of `A` may be then be accessed via `A.name`, `A.artist`, etc. We can also define functions for

custom types in the definition of the class:

Python

```python
class Album(object):
    def __init__(self,name,artist,year,songs):
        self.name = name
        self.artist = artist
        self.year = year
        self.length = length

def numYearsAgo(self):
    from datetime import datetime
    return datetime.now().year - self.year
```

You can define new types in R using the function `setClass`, but user-defined types do not feature as prominently in idiomatic R as they do in Julia or Python.

3.5 File I/O

3.5.1 Text files

To read or write text files in Python, it's good practice to use a `with` block in conjunction with the function `open`. This method ensures that the file is properly closed at the end.

Python

```python
# Reading a file:
with open("workfile.txt") as f:
    read_data = f.read()
# Writing a file:
with open("workfile.txt","w") as f:
    f.write("new file contents")
```

If the file is too large to read it all at once, you can read it one line at a time using `readline`. Make sure to close the file when you're done.

Python

```python
f = open("workfile.txt")
first_line = f.readline()
...
f.close()
```

3.5.2 CSV files

Julia, Python, and R all use their tabular data storage type a *data frame*. Data frames in Python are supplied by `pandas`, and data frame functionality does not have to be imported in R because it's part of the base language. To create a data frame from a local CSV file (and then write it back to disk):

Python

```python
import pandas
myDataFrame = pandas.read_csv("data.csv")
pandas.write_csv(myDataFrame)
```

R

```r
myDataFrame = read.csv("data.csv")
write.csv(myDataFrame)
```

3.6 Speed

3.6.1 Vectorized operations

Loops written explicitly in Python and R are very slow compared to Julia's loops. As a result, the idiom in Python and R is to disguise loops in vector operations whenever possible. This allows the loop to be performed using optimized code written by package developers, rather than directly in the Python or R interpreter. Vectorizing code can also help with clarity in many cases.

For example, if we have two long lists `A` and `B` and want to calculate the vector sum `A+B`, we could use the list comprehension `[A[k] + B[k] for k in range(n)]`. However, it is preferable to work with NumPy arrays and use the built-in addition operator:

Python

```python
import numpy as np
n = 10000
A = np.random.rand(n)
B = np.random.rand(n)
np.array(A) + np.array(B)
```

If we wanted to calculate the running sum of the elements of A, rather than

Python

```python
[sum(A[:i]) for i in range(len(A))]
```

we should use the NumPy function `cumsum`, as in

Python

```python
import numpy as np
np.cumsum(A)
```

In R we can add vectors directly using the `+` operator, and we can also find the cumulative sum of an array using `cumsum`.

R

```r
A <- runif(10000) # 10000 uniform-[0,1] random samples
cumsum(A)
```

Exercise 3.6.1

In R and in Python, define a random vector of length 1000 and set to zero all of the entries of the vector which are below the median of the entries of the vector.

Do this without writing any loops, using the Python functions `np.where` and `np.median` and the R functions `which` and `median`.

3.6.2 Compilation

In Julia you can sum the first billion integer square reciprocals in a few seconds with a basic loop:

Julia

```julia
function pisum()
    sum = 0.0
    for k = 1:10^9
        sum += 1.0/(k*k)
    end
    sum
end
```

You could write an analogous function in Python or R, but it would take a long time to run. Vectorization doesn't help with this problem, because it's time consuming just to populate an array with a billion entries. Fortunately, there is a Python package which helps achieve some of the same performance benefits you have in Julia: *Numba*. If we load Numba and place a `@jit` decorator before our function, then Numba will compile it into fast machine code for us. For example:

Python

```python
import numba
@numba.jit
def pisum():
    sum = 0.0
    for k in range(1,10**9+1):
        sum += 1.0/(k*k)
    return sum
```

JIT stands for *just-in-time* compilation, which refers to the technique of compiling code on the fly as needed. *Ahead-of-time* compilation saves time when the program runs, but it is less flexible since we have to determine in advance which types we'll need to call the function on. There is an ahead-of-time compilation system for Python: *Cython*. However, using Cython is rather more complex than using Numba, since Cython is a different language with fewer features than regular Python, and extra syntax for communicating information about types to the compiler.

Exercise 3.6.2

Compare the run time for the jitted `pisum` function and that of the regular `pisum` function.

3.7 Comparison of languages: a fireside chat

If Python and R are the most popular data science languages, then why use Julia at all? The short answer is that Python was designed as a general purpose language, R was designed for statistics, and Julia was designed for scientific computing. Most of what we have done in the course (running simulations, implementing numerically intensive models, etc.) is much closer to scientific computing than to statistics or general scripting.

Using a language whose strengths align with the task at hand offers material advantages to the learner. For example, in the case of Julia, one doesn't have to learn strategies for overcoming the language's intrinsic performance problems (which famously tends to consume considerable mindshare for those working extensively in MATLAB, Python, or R). Also, the availability of conveniences like multidimensional array comprehensions eliminates whole classes of programming challenges. Most notably for intermediate and advanced users, Julia's user-defined types are first-class. So you can—without a performance penalty—make full use of types to model your problem in a conceptually clear way. Experienced Python and R users have a rule of thumb to never use classes in performance-sensitive code, and this conflict between clarity and speed is a constant source of tension.

Likewise, R's tidyverse packages create an unparalleled experience for data cleaning and visualization. You can read in a data frame from an Excel spreadsheet, manipulate it by piping it through a few **`dplyr`** functions, and make a publication-ready **`ggplot`** in a few lines of code which are clear enough to read and modify effortlessly. These tasks are also supported in Python and Julia, of course, but the tidyverse is widely regarded as the gold standard for convenience of basic tidy-data statistics tasks.

To cite similar examples for Python, **`scikitlearn`** offers a superbly well-organized interface to a wide variety of machine learning models. The centralized organization allows the user to avoid having to learn the nuances of a variety of disjoint packages. And **`Keras`** eliminates the hassle of doing deep learning on the GPU.

Some knowledge of the broader ecosystem can be an advantage even if you predominantly work in one language, because excellent interop tools have been developed (Julia's **`PyCall`** and **`RCall`**, Python's **`rpy2`** and **`pyjulia`**, and R's **`reticulate`** and **`JuliaCall`**). For example, Julia can be used as a much more convenient alternative to C or FORTRAN when you're looking to address a perfomance bottleneck by re-writing a particular function in a faster language.

4 Statistics

4.1 Point estimation

In Section 1.1.1.1, we discussed the problem of estimating a distribution given a list of independent samples it. Now we turn to the simpler task of **point estimation**: estimating a single real-valued feature (such as the mean, variance, or maximum) of a distribution. We begin by formalizing the notion of a real-valued feature of a distribution.

Definition 4.1.1: Statistical functional

A **statistical functional** is any function T from the set of distributions to $[-\infty, \infty]$.

For example, if we define $T_1(\nu)$ to be the mean of ν, then T_1 is a statistical functional. Similarly, consider the *maximum* functional $T_2(\nu) = F^{-1}(1)$ where F is the CDF of ν. To give a more complicated example, we define $T_3(\nu)$ to be the expected value of the difference between the greatest and least of 10 independent random variables with common distribution ν. Then T_3 also a statistical functional*.

* Note that we could estimate T_3 by simulation, though it may be difficult to calculate analytically.

Given a statistical functional, our goal will be to use a list of independent samples from ν to estimate $T(\nu)$.

Definition 4.1.2: Estimator

An **estimator** $\widehat{\theta}$ is a random variable which is a function of n i.i.d. random variables.

Example 4.1.1

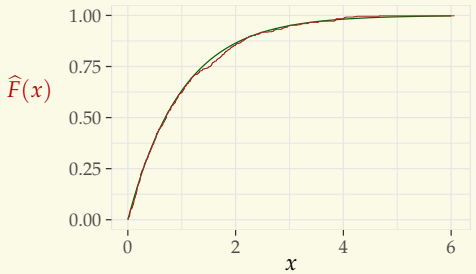
Draw 500 independent samples from an exponential distribution with parameter 1. Plot the function $\widehat{F}$ which maps x to the proportion of samples at or to the left of x on the number line. Compare this graph to the graph of the CDF of the exponential distribution with parameter 1.

Solution

We use the step geom to graph $\widehat{F}$:

```
n <- 500
xvals = seq(0,8,length=100)

ggplot() +
  geom_line(aes(x=xvals,y=1-exp(-xvals))) +
  geom_step(aes(x=sort(rexp(n)),y=(1:n)/n))
```



Example 4.1.1 suggests an idea for estimating $\widehat{\theta}$: since the unknown distribution ν is typically close* to the measure $\widehat{\nu}$ which places mass $\frac{1}{n}$ at each of the observed samples. Then we can build an estimator of $T(\nu)$ by plugging $\widehat{\nu}$ into T.

* In the sense of proximity of CDFs; see Section 4.3.

Definition 4.1.3: Plug-in estimator

The plug-in estimator of $\theta = T(\nu)$ is $\widehat{\theta} = T(\widehat{\nu})$.

Example 4.1.2

Find the plug-in estimator of the mean of a distribution. Find the plug-in estimator of the variance.

Solution

The plug-in estimator of the mean is the mean of the empirical distribution, which is the average of the locations of the samples. We call this the **sample mean**:

$$\overline{X} = \frac{X_1 + \cdots + X_n}{n}.$$

Likewise, the plug-in estimator of the variance is **sample variance**

$$S^2 = \frac{1}{n}\left((X_1 - \overline{X})^2 + (X_2 - \overline{X})^2 + \cdots + (X_n - \overline{X})^2\right).$$

Ideally, an estimator $\widehat{\theta}$ is close to θ with high probability. We will see that we can decompose the question of whether $\widehat{\theta}$ is close to θ into two sub-questions: is the *mean* of $\widehat{\theta}$ close to θ, and is $\widehat{\theta}$ close to its mean with high probability?

Definition 4.1.4: Bias

The **bias** of an estimator $\widehat{\theta}$ is

$$\mathbb{E}[\widehat{\theta}] - \theta.$$

An estimator is said to be **biased** if its bias is nonzero and **unbiased** if its bias is zero.

Example 4.1.3

Consider the estimator

$$\widehat{\theta} = \max(X_1, \ldots, X_n)$$

of the maximum functional. Assuming that the distribution has a density function, show that $\widehat{\theta}$ is biased.

Solution

If ν is a continuous distribution, then the probability of the event $\{X_i < T(\nu)\}$ is 1 for all $i = 1, 2, \ldots, n$. This implies that $\widehat{\theta} < T(\nu)$ with probability 1. Taking expectation of both sides, we find

that $\mathbb{E}[\widehat{\theta}] < T(\nu)$. Therefore, this estimator has negative bias.

Zero or small bias is a desirable property of an estimator: it means that the estimator is accurate *on average*. The second desirable property of an estimator is for the probability mass of its distribution to be concentrated near its mean:

Definition 4.1.5: Standard error

The standard error $\mathrm{se}(\widehat{\theta})$ of an estimator $\widehat{\theta}$ is its standard deviation.

Example 4.1.4

Find the standard error of the sample mean if the distribution ν with variance σ^2.

Solution

We have

$$\operatorname{Var}\left(\frac{X_1 + X_2 + \cdots + X_n}{n}\right) = \frac{1}{n^2}(n \operatorname{Var} X_1) = \frac{\sigma^2}{n}.$$

Therefore, the standard error is $\sigma/\sqrt{n}$.

If the expectation of an estimator of θ is close to θ and if the estimator close to its average with high probability, then it makes sense that $\widehat{\theta}$ and θ are close to each other with high probability. We can measure the discrepancy between $\widehat{\theta}$ and θ directly by computing their average squared difference:

Definition 4.1.6: Mean squared error

The mean squared error of an estimator $\widehat{\theta}$ is $\mathbb{E}[(\widehat{\theta} - \theta)^2]$.

As advertised, the mean squared error decomposes as a sum of *squared bias* and *squared standard error*:

Theorem 4.1.1

The mean squared error of an estimator θ is equal to its variance plus its squared bias:

$$\mathbb{E}[(\widehat{\theta} - \mathbb{E}[\widehat{\theta}])^2] + (\mathbb{E}[\widehat{\theta}] - \theta)^2.$$

Proof

The idea is to add and subtract the mean of $\widehat{\theta}$. We find that

$$\begin{aligned}\mathbb{E}[(\widehat{\theta} - \theta)^2] &= \mathbb{E}[(\widehat{\theta} - \mathbb{E}[\widehat{\theta}] + \mathbb{E}[\widehat{\theta}] - \theta)^2] \\ &= \mathbb{E}[(\widehat{\theta} - \mathbb{E}[\widehat{\theta}])^2] + 2\mathbb{E}[(\widehat{\theta} - \mathbb{E}[\widehat{\theta}])(\mathbb{E}[\widehat{\theta}] - \theta)] + (\mathbb{E}[\widehat{\theta}] - \theta)^2.\end{aligned}$$

The middle term is zero by linearity of expectation.

If the bias and standard error of an estimator both converge to 0, then the estimator is *consistent*:

Definition 4.1.7: Consistent

An estimator is **consistent** if $\widehat{\theta}$ converges to θ in probability as $n \to \infty$.

Example 4.1.5

Show that the plug-in maximum estimator $\widehat{\theta}_n = \max(X_1, \ldots, X_n)$ of $\theta = T(v) = F^{-1}(1)$ is consistent, assuming that the distribution belongs to the parametric family $\{\text{Unif}([0, b]) : b \in \mathbb{R}\}$.

Solution

The probability that $\widehat{\theta}_n$ is more than ϵ units from θ is equal to the probability that every sample is less than $\theta - \epsilon$, which by independence is equal to

$$\left(\frac{\theta - \epsilon}{\theta}\right)^n.$$

This converges to 0 as $n \to \infty$, since $\frac{\theta-\epsilon}{\theta} < 1$.

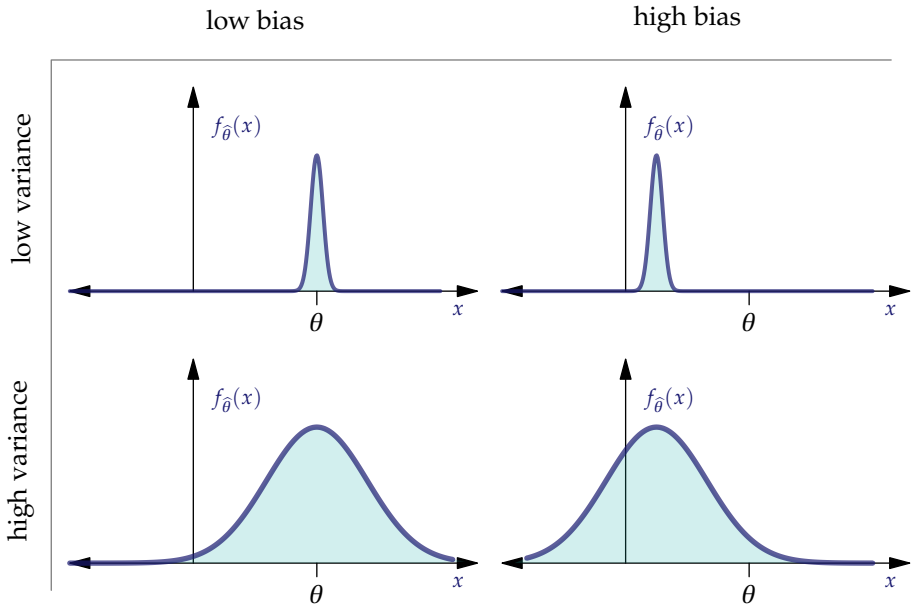


Figure 4.1 An estimator of θ has high or low bias depending on whether its mean is far from or close to θ. It has high or low variance depending on whether its mass is spread out or concentrated.

Example 4.1.6

Show that the sample variance $S^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \overline{X})^2$ is biased.

Solution

We will perform the calculation for $n = 3$. It may be generalized to other values of n by replacing 3 with n and 2 with $n-1$. We have

$$\mathbb{E}[S^2] = \frac{1}{3}\mathbb{E}\left[\left(\frac{2}{3}X_1 - \frac{1}{3}X_2 - \frac{1}{3}X_3\right)^2 + \left(\frac{2}{3}X_2 - \frac{1}{3}X_3 - \frac{1}{3}X_1\right)^2 + \left(\frac{2}{3}X_3 - \frac{1}{3}X_1 - \frac{1}{3}X_2\right)\right]^2$$

Squaring out each trinomial, we get $\frac{4}{9}X_1^2$ from the first term and $\frac{1}{9}X_1^2$ from each of the other two. So altogether the X_1^2 term is $\frac{6}{9}X_1^2$. By symmetry, the same is true of X_2^2 and X_3^2. For cross-terms, we get $-\frac{4}{9}X_1X_2$ from the first squared expression, $-\frac{4}{9}X_1X_2$ from the second, and $\frac{2}{9}X_1X_2$ from the third. Altogether, we get $-\frac{6}{9}X_1X_2$. By symmetry, the remaining two terms are $-\frac{6}{9}X_1X_3 - \frac{6}{9}X_2X_3<$.

Recalling that $\mathrm{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$ for any random variable X, we have $\mathbb{E}[X_1^2] = \mu^2 + \sigma^2$, where μ and σ are the mean and standard deviation of the distribution of X_1 (and similarly for X_2 and X_3. So we have

$$\mathbb{E}[S^2] = \frac{1}{3}\left(\frac{6}{9}(X_1^2 + X_2^2 + X_3^2) - \frac{6}{9}(X_1X_2 + X_1X_3 + X_2X_3)\right) \quad = \frac{1}{3}\cdot\frac{6}{9}(3(\sigma^2+\mu^2) - 3\mu^2) = \frac{2}{3}\sigma^2.$$

If we repeat the above calculation with n in place of 3, we find that the resulting expectation is $\frac{n-1}{n}\sigma^2$.

Motivated by Example 4.1.6, we define the **unbiased sample variance**

$$\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2.$$

4.2 Confidence intervals

It is often of limited use to know the value of an estimator given an observed collection of samples, since the single value does not indicate how close we should expect θ to be to $\widehat{\theta}$. For example, if a poll estimates that a randomly selected voter has 46% probability of being a supporter of candidate A and a 42% probability of being a supporter of candidate B, then knowing more information about the distributions of the estimators is essential if we want to know the probability of winning for each candidate. Thus we introduce the idea of a *confidence interval*.

Definition 4.2.1: Confidence interval

Consider an unknown probability distribution ν from which we get n independent samples $X_1, \ldots, X_n$, and suppose that θ is the value of some statistical functional of ν. A **confidence interval** for θ is an interval-valued function of the sample data $X_1, \ldots, X_n$. A confidence interval has **confidence level** $1-\alpha$ if it contains θ with probability at least $1-\alpha$.

Exercise 4.2.1

Use Chebyshev's inequality to show that if $\widehat{\theta}$ is unbiased, then $(\widehat{\theta} - k\,\mathrm{se}(\widehat{\theta}), \widehat{\theta} + k\,\mathrm{se}(\widehat{\theta}))$ is a $1 - \frac{1}{k^2}$ confidence interval.

If $\widehat{\theta}$ is approximately normally distributed, then we can give tighter confidence intervals using the normal approximation:

Exercise 4.2.2

Show that if $\widehat{\theta}$ is approximately normally distributed, then $(\widehat{\theta} - k\,\mathrm{se}(\widehat{\theta}), \widehat{\theta} + k\,\mathrm{se}(\widehat{\theta}))$ is a $1 - 2(1 - \Phi(k))$ confidence interval, where Φ is the CDF of the standard normal distribution.

If we are estimating a *function*-valued feature of v rather than a single number (for example, a regression function), then we might want to provide a confidence *band* which traps the whole graph of the function with specified probability (see Theorem 4.3.2 for an example).

Definition 4.2.2: Confidence band

Let $I \subset \mathbb{R}$, and suppose that T is a function from the set of distributions to the set of real-valued functions on I. A $1 - \alpha$ **confidence band** for $T(v)$ is pair of random functions $y_{\min}$ and $y_{\max}$ from I to $\mathbb{R}$ defined in terms of n independent samples from v and having $y_{\min} \leq T(v) \leq y_{\max}$ everywhere on I with probability at least $1 - \alpha$.

4.3 Empirical CDF convergence

Let's revisit Example 4.1.1, where we observed that the CDF of the empirical distribution of an independent list of samples from a distribution tends to be close to the CDF of the distribution itself. The **Glivenko-Cantelli theorem** is a mathematical formulation of the idea that these two functions are indeed close.

Theorem 4.3.1: Glivenko-Cantelli

If F is the CDF of a distribution v and $\widehat{F}_n$ is the CDF of the empirical distribution $\widehat{v}_n$ of n samples from v, then F_n converges to F along the whole number line:

$$\max_{x \in \mathbb{R}} |F(x) - \widehat{F}_n(x)| \to 0 \quad \text{as } n \to \infty,$$

in probability.

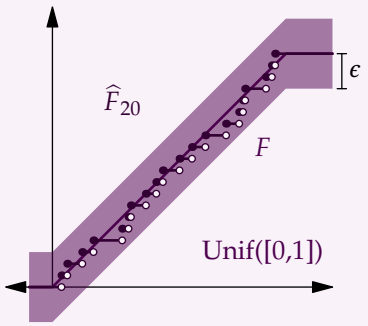
The **Dvoretzky-Kiefer-Wolfowitz inequality** quantifies this result by providing a confidence band.

Theorem 4.3.2: DKW inequality

If $X_1, X_2, \ldots$ are independent random variables with common CDF F, then for all $\epsilon > 0$, we have

$$\mathbb{P}\left(\max_x |F(x) - \widehat{F}_n(x)| \geq \epsilon\right) \leq 2e^{-2n\epsilon^2}.$$

In other words, the probability that the graph of $\widehat{F}_n$ lies in the ϵ-band around F (or vice versa) is at least $1 - 2e^{-2n\epsilon^2}$.



Exercise 4.3.1

Show that if $\epsilon_n = \sqrt{\frac{1}{2n}\log(\frac{2}{\alpha})}$, then with probability at least $1 - \alpha$, we have $|F(x) - F_n(x)| \leq \epsilon$ for all $x \in \mathbb{R}$.

4.4 Bootstrapping

Bootstrapping is the use of simulation to approximate the value of the plug-in estimator of a statistical functional T which is expressed in terms of independent samples from the input distribution ν. The key idea is that drawing k samples from $\widehat{\nu}$ is the same as drawing k times with replacement from the list of samples.

Example 4.4.1

Consider the statistical functional $T(\nu) =$ the expected difference between the greatest and least of 10 independent samples from ν. Suppose that 50 samples $X_1, \ldots, X_{50}$ from ν are observed, and that $\widehat{\nu}$ is the associated empirical CDF. Explain how $T(\widehat{\nu})$ may be estimated with arbitrarily small error.

Solution

The value of $T(\widehat{\nu})$ is defined to be the expectation of a distribution that we have instructions for how to sample from. So we sample 10 times with replacement from $X_1, \ldots, X_{50}$, identify the largest and smallest of the 10 samples, and record the difference. We repeat B times for some large integer B, and we return the sample mean of these B values.

By the law of large numbers, the result can be made arbitrarily close to $T(\widehat{\nu})$ with arbitrarily high probability by choosing B sufficiently large.

Although Example 4.4.1 might seem a bit contrived, bootstrapping is useful in practice because of a common source of statistical functionals that fit the bootstrap form: *standard errors*.

Example 4.4.2

Suppose that we estimate the median θ of a distribution using the plug-in estimator $\widehat{\theta}$ for 75 observed samples, and we want to produce a confidence interval for θ. Show how to use bootstrapping to estimate the standard error of the estimator.

Solution

By definition, the standard error of $\widehat{\theta}$ is the square root of the variance of the median of 75 independent draws from ν. Therefore, the plug-in estimator of the standard error is the square root of the variance of the median of 75 independent draws from $\widehat{\nu}$. This can be readily simulated. If the samples are stored in a vector `X`, then

```
sd(sapply(1:10^5,function(n) {median(sample(X,75,replace=TRUE))}))
```

returns a very accurate approximation of $T(\widehat{\nu})$.

Exercise 4.4.1

Suppose that ν is the uniform distribution on $[0,1]$. Generate 75 samples from ν, store them in a vector X, and compute the bootstrap estimate of $T(\widehat{\nu})$. Use Monte Carlo simulation to directly estimate $T(\nu)$. Can the gap between your approximations of $T(\widehat{\nu})$ and $T(\nu)$ be made arbitrarily small by using more bootstrap samples?

4.5 Maximum likelihood estimation

So far we've only had one idea for building an estimator, which is to plug $\widehat{\nu}$ into the statistical functional. In this section, we'll learn another approach which is quite general and has some compelling properties.

Consider a parametric family $\{f_{\boldsymbol{\theta}}(x) : \boldsymbol{\theta} \in \mathbb{R}^d\}$ of PDFs or PMFs. Given $\mathbf{x} \in \mathbb{R}^n$, the **likelihood** $\mathcal{L}_{\mathbf{x}} : \mathbb{R}^d \to \mathbb{R}$ is defined by

$$\mathcal{L}_{\mathbf{x}}(\boldsymbol{\theta}) = f_{\boldsymbol{\theta}}(x_1) f_{\boldsymbol{\theta}}(x_2) \cdots f_{\boldsymbol{\theta}}(x_n).$$

The idea is that if $\mathbf{X}$ is a vector of n independent samples drawn from $f_{\boldsymbol{\theta}}(x)$, then $\mathcal{L}_{\mathbf{X}}(\boldsymbol{\theta})$ is small or zero when $\boldsymbol{\theta}$ is not in concert with the observed data.

Example 4.5.1

Suppose $x \mapsto f(x;\theta)$ is the density of a uniform random variable on $[0,\theta]$. We observe four samples drawn from this distribution: 1.41, 2.45, 6.12, and 4.9. Find $\mathcal{L}(5)$, $\mathcal{L}(10^6)$, and $\mathcal{L}(7)$.

Solution

The likelihood at 5 is zero, since $f_5(x_3) = 0$. The likelihood at 10^6 is very small, since $\mathcal{L}(10^6) = (1/10^6)^4 = 10^{-24}$. The likelihood at 7 is larger: $(1/7)^4 = 1/2401$.

We can see from Example 4.5.1 that likelihood has the property that it is zero or small at implausible values of $\boldsymbol{\theta}$, and larger at more reasonable values. Thus we propose the **maximum likelihood estimator**

$$\widehat{\boldsymbol{\theta}}_{\text{MLE}} = \text{argmax}_{\boldsymbol{\theta}\in\mathbb{R}^d} \mathcal{L}_{\mathbf{X}}(\boldsymbol{\theta}).$$

Example 4.5.2

Suppose that $x \mapsto f(x;\mu,\sigma^2)$ is the normal density with mean μ and variance σ^2. Find the maximum likelihood estimator for μ and σ^2.

Solution

The maximum likelihood estimator is the minimizer of the logarithm of the likelihood function, which is

$$-\frac{n}{2}\log 2\pi - n\log\sigma - \frac{n}{2}\log 2\pi - n\log\sigma - \frac{(X_1-\mu)^2}{2\sigma^2} - \cdots - \frac{(X_n-\mu)^2}{2\sigma^2}$$

Setting the derivatives with respect to μ and σ^2 equal to zero, we find $\mu = \overline{X} = \frac{1}{n}(X_1+\cdots+X_n)$ and $\sigma^2 = \frac{1}{n}((X_1-\overline{X})^2+\cdots+(X_n-\overline{X})^2)$. So the maximum likelihood estimator agrees with the plug-in estimator for μ and σ^2.

MLE enjoys several nice properties: under certain regularity conditions, we have

1. **Consistency**: $\mathbb{E}[(\widehat{\theta}_{\text{MLE}}-\theta)^2] \to 0$ as the number of samples goes to ∞.
2. **Asymptotic normality**: $(\widehat{\theta}_{\text{MLE}}-\theta)/\sqrt{\text{Var}\,\widehat{\theta}_{\text{MLE}}}$ converges to $\mathcal{N}(0,1)$ as the number of samples goes to ∞.
3. **Asymptotic optimality**: the MSE of the MLE converges to 0 approximately as fast as the MSE of any other consistent estimator.

Potential difficulties with MLE:

1. **Computational difficulties**. It might be difficult to work out where the maximum of the likelihood occurs, either analytically or numerically.
2. **Misspecification**. The MLE may be inaccurate if the distribution of the samples is not in the specified parametric family.
3. **Unbounded likelihood**. If the likelihood function is not bounded, then $\widehat{\theta}_{\text{MLE}}$ is not well-defined.

4.6 Hypothesis testing

Hypothesis testing is a disciplined framework for adjudicating whether observed data do not support a given hypothesis.

Consider an unknown distribution from which we will observe n samples $X_1,\ldots X_n$.

1. We state a hypothesis H_0—called the **null hypothesis**—about the distribution.

2. We come up with a **test statistic** T, which is a function of the data $X_1, \ldots X_n$, for which we can evaluate the distribution of T assuming the null hypothesis.
3. We give an **alternative hypothesis** H_a under which T is expected to be significantly different from its value under H_0.
4. We give a significance level α (like 5% or 1%), and based on H_a we determine a set of values for T—called the *critical region*—which T would be in with probability at most α under the null hypothesis.
5. **After setting H_0, H_a, α, T, and the critical region**, we run the experiment, evaluate T on the samples we get, and record the result as t_{obs}.
6. If t_{obs} falls in the critical region, we reject the null hypothesis. The corresponding ***p*-value** is defined to be the minimum α-value which would have resulted in rejecting the null hypothesis, with the critical region chosen in the same way*.

Example 4.6.1

Muriel Bristol claims that she can tell by taste whether the tea or the milk was poured into the cup first. She is given eight cups of tea, four poured milk-first and four poured tea-first.

We posit a null hypothesis that she isn't able to discern the pouring method, and an alternative hypothesis that she can tell the difference. How many cups does she have to identify correctly to reject the null hypothesis with 95% confidence?

Solution

Under the null hypothesis, the number of cups identified correctly is 4 with probability $1/\binom{8}{4} \approx 1.4\%$ and at least 3 with probability $17/70 \approx 24\%$. Therefore, at the 5% significance level, only a correct identification of all the cups would give us grounds to reject the null hypothesis. The p-value in that case would be 1.4%.

Failure to reject the null hypothesis is not necessarily evidence *for* the null hypothesis. The **power** of a hypothesis test is the conditional probability of rejecting the null hypothesis given that the alternative hypothesis is true. A p-value may be low either because the null hypothesis is true or because the test has low power.*

* For example, reducing the number of samples used for a test decreases its power.

Definition 4.6.1

The **Wald test** is based on the normal approximation. Consider a null hypothesis $\theta = 0$ and the alternative hypothesis $\theta \neq 0$, and suppose that $\widehat{\theta}$ is approximately normally distributed. The Wald test rejects the null hypothesis at the 5% significance level if $|\widehat{\theta}| > 1.96\,\text{se}(\widehat{\theta})$.

Example 4.6.2

Consider the alternative hypothesis that 8-cylinder engines have lower fuel economy than 6-cylinder engines (with null hypothesis that they are the same). Apply the Wald test, using the `mtcars` dataset available in R.

Solution

We frame the problem as a question about whether the *difference in means* between the distribution of 8-cylinder `mpg` values and the distribution of 6-cylinder `mpg` values is zero. We use the difference between the sample means $\overline{X}$ and $\overline{Y}$ of the two populations as an estimator of the difference in means. If we think of the records in the data frame as independent, then $\overline{X}$ and $\overline{Y}$ are independent. Since each is approximately normally distributed by the central limit theorem, their difference is therefore also approximately normal.*

*...because linear combinations of independent normal random variables are normal.

So, let's calculate the sample mean and sample variance for the 8-cylinder cars and for the 6-cylinder cars.

```
library(tidyverse)

stats <- mtcars %>%
  group_by(cyl) %>%
  filter(cyl %in% c(6,8)) %>%
  summarise(m = mean(mpg), S2 = var(mpg), n = n(), se = sqrt(S2/n))
```

Given that the distribution of 8-cylinder `mpg` values has variance σ^2_{eight}, the variance of the sample mean $\overline{X}$ is $\sigma^2_{\text{eight}}/n_{\text{eight}}$, where n_{eight} is the number of 8-cylinder vehicles (and similarly for $\overline{Y}$). Therefore, we estimate the variance of the difference in sample means as

$$\text{Var}(\overline{X} - \overline{Y}) = \text{Var}(\overline{X}) + \text{Var}(\overline{Y}) = \sigma^2_{\text{eight}}/n_{\text{eight}} + \sigma^2_{\text{six}}/n_{\text{six}}.$$

Under the null hypothesis, therefore, $\overline{X} - \overline{Y}$ has mean zero and standard error $\sqrt{\sigma^2_{\text{eight}}/n_{\text{eight}} + \sigma^2_{\text{six}}/n_{\text{six}}}$. We therefore reject the null hypothesis with 95% confidence if the value of $\overline{X} - \overline{Y}$ divided by its estimated standard error exceeds 1.96. We find that

```
z <- (stats$m[1] - stats$m[2]) / sqrt(sum(stats$se^2))
```

returns 5.29, so we do reject the null hypothesis at the 95% confidence level. The p-value of this test is `1 - pnorm(z)` $= 6.08 \times 10^{-6}\%$.

The following test is more flexible than the Wald test, since it doesn't rely on the normal approximation. It's based on a simple idea: if there's no difference in labels, the data shouldn't look very different if we shuffle them around.

Definition 4.6.2

The **random permutation test** is applicable when the null hypothesis is that two distributions are the same.

1. We compute the difference between the sample means for the two groups.
2. We randomly re-assign the group labels and compute the resulting sample mean differences. Repeat many times.
3. We check where the original difference falls in the sorted list of re-sampled differences.

Example 4.6.3

Suppose the heights of the Romero sons are 72, 69, 68, and 66 inches, and the heights of the Larsen sons are 70, 65, and 64 inches. Consider the null hypothesis that the height distributions for the two families are the same, with the alternative hypothesis that they are not. Determine whether a random permutation test applied to the absolute sample mean difference rejects the null hypothesis at significance level $\alpha = 5\%$.

Solution

We find that the absolute sample mean difference of about 2.4 inches is larger than only about 68% of the mean differences obtained by resampling many times.

```
set.seed(123)
romero <- c(72, 69, 68, 66)
larsen <- c(70, 65, 64)
actual.diff <- abs(mean(romero) - mean(larsen))

resample.diff <- function(n) {
  shuffled <- sample(c(romero,larsen))
  abs(mean(shuffled[1:4]) - mean(shuffled[5:7]))
}

sum(sapply(1:10000,resample.diff) < actual.diff)
```

Since 68% < 95%, we retain the null hypothesis.

4.6.1 Multiple hypothesis testing

If we conduct many hypothesis tests, then the probability of obtaining some false rejections is high*. This is called the **multiple testing problem**.

* xkcd.com/882

The **Bonferroni method** is to reject the null hypothesis only for those tests whose p-values are less than α divided by the number of hypothesis tests being run. This ensures that the probability of having even one false rejection is less than α, so it is very conservative.

Example 4.6.4

Suppose that 10 different genes are tested to determine whether they have an affect on heart disease. The 10 p-values resulting from these hypothesis tests are (rounded to the nearest hundredth of a percent):

$$0.89\%, 2.71\%, 9.11\%, 2.18\%, 9.17\%, 7.48\%, 5.0\%, 2.02\%, 5.22\%, 9.46\%$$

Which results are reported as significant at the 5% level, according to the Bonferroni method?

Solution

At the 5% level, only p values less than 5%/10 = 0.5% are reported as significant (since we ran ten hypothesis tests). Since none of the p values are below 0.5%, none of the genes will be considered significant.