

Logistic Regression

Last modified: 2026-10-28

```
# The Python chunks on this page import these packages; reticulate provisions them.
library(reticulate)
py_require(c("numpy", "pandas", "statsmodels", "scipy", "scikit-learn", "matplotlib"))
```

i Source

Adapted from Brian Hutchinson's Fall 2025 lecture notes, reorganized as exercises. It also draws on Kameron Decker Harris' 2022 weekly notes, for the log-odds view, the margin form of the loss and the hand-drawn figures, and on Logan Sizemore's Spring 2025 lectures on classification and on multinomial logistic regression, for why thresholding a linear fit fails, where cross-entropy comes from, and the redundant weight vector.

Hutchinson's version introduces gradient descent as a side topic partway through. That material now has a chapter of its own, so we keep the side topic as an outtake rather than repeat it here. Solution 12.7 corrects the spurious $1/n$ in the source's multinomial gradient and flags the correction rather than fixing it silently.

1 Predicting a category

Linear regression predicts a number. A great many questions are not about a number: whether a patient recovers, whether an email is spam, which of ten digits a scan shows.

Let the data be $\mathcal{X} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ with $x_i \in \mathbb{R}^p$ and y_i a label rather than a measurement. Two labels is binary classification, which most of this chapter covers; more than two is multi-class classification, which Section 12 covers.

We could predict the label directly. Predicting $P(Y = 1 \mid x)$ is more informative: whoever uses the prediction can decide how to act on it, because a probability states how confident the model is and a class label alone does not. A model of $P(Y \mid x)$ is discriminative: it models the conditional distribution¹ of the label given the features, without modelling the distribution of the features.

The simplest model of anything is a constant. The next simplest is built from a linear function, which is where this chapter starts.

i Watch

Hutchinson teaches classification and decision boundaries^a (6 min) [1]. The video site requires a login, which enrolled students find in the syllabus.

^ahttps://facultyweb.cs.wvu.edu/~hutchib2/video_lectures/data371/#classification_overview

Exercise 1.1 (Fitting a line to a yes-or-no outcome). The Western Collaborative Group Study followed 3154 middle-aged men for coronary heart disease [2]. In `data/wcgs.csv`^a, the column `chd` holds each man's outcome^b $y_i \in \{0, 1\}$: 1 if man i had an incident case and 0 if not. To predict `chd`, we try one

¹<https://morrison-lab.github.io/pds/random-variables.html#def-cdf>

feature: man i 's body mass index x_i , from the column `bmi`. We do not know that `bmi` is related to `chd`. Later WCGS examples use age at enrollment (`age`) as the main covariate. This exercise uses `bmi` instead, to show what a straight line does to a yes-or-no outcome. The column `dibpat` records each man's behavior pattern, type A or type B. This exercise pools the men of both patterns, so it ignores that column. We already know how to fit $\hat{y} = \tilde{\beta} \cdot \tilde{x}$ by least squares. Treat the two values of `chd` as the numbers 0 and 1, and fit linear regression in R or Python.

Fit a line of `chd` on `bmi`. Report its slope and intercept, the range of the fitted $\hat{y}$, and how many men have a fitted $\hat{y}$ outside $[0, 1]$. Then plot the data and the fitted line. Hint: because `chd` takes only the values 0 and 1, plotted points overlap at the two vertical coordinates $y = 0$ and $y = 1$. Instead, consider a rug^c, with one tick per man along the bottom edge for `chd` = 0 and along the top edge for `chd` = 1, and/or two histograms^d of `bmi`, one along the line $y = 0$ for `chd` = 0 and one extending downward from the line $y = 1$ for `chd` = 1.

^a<https://morrison-lab.github.io/lds/chapters/data/wcgs.csv>

^b<https://morrison-lab.github.io/sds/model-validation.html#def-outcome-vector>

^c<https://morrison-lab.github.io/vds/chapters/binary-outcomes.html#exm-rug-plot>

^d<https://morrison-lab.github.io/vds/chapters/binary-outcomes.html#exm-two-histograms>

Solution

Solution 1.1. Read the data and fit the line by least squares.

1.1 R

```
wcgs <- read.csv("data/wcgs.csv")
fit_bmi <- lm(chd ~ bmi, data = wcgs)
fitted_bmi <- fitted(fit_bmi)
outside <- sum(fitted_bmi < 0 | fitted_bmi > 1)
c(coef(fit_bmi), range(fitted_bmi), outside = outside)
```

(Intercept)	bmi	outside
-0.081181996	0.006634395	-0.006943111 0.177227706 1.000000000

```
library(ggplot2)
ggplot(wcgs, aes(bmi, chd)) +
  geom_rug(data = subset(wcgs, chd == 0), sides = "b", alpha = 0.3) +
  geom_rug(data = subset(wcgs, chd == 1), sides = "t", alpha = 0.3) +
  geom_hline(yintercept = c(0, 1), linetype = "dashed") +
  geom_smooth(method = "lm", formula = y ~ x, se = FALSE) +
  labs(x = "Body mass index (bmi)", y = "Fitted chd")
```

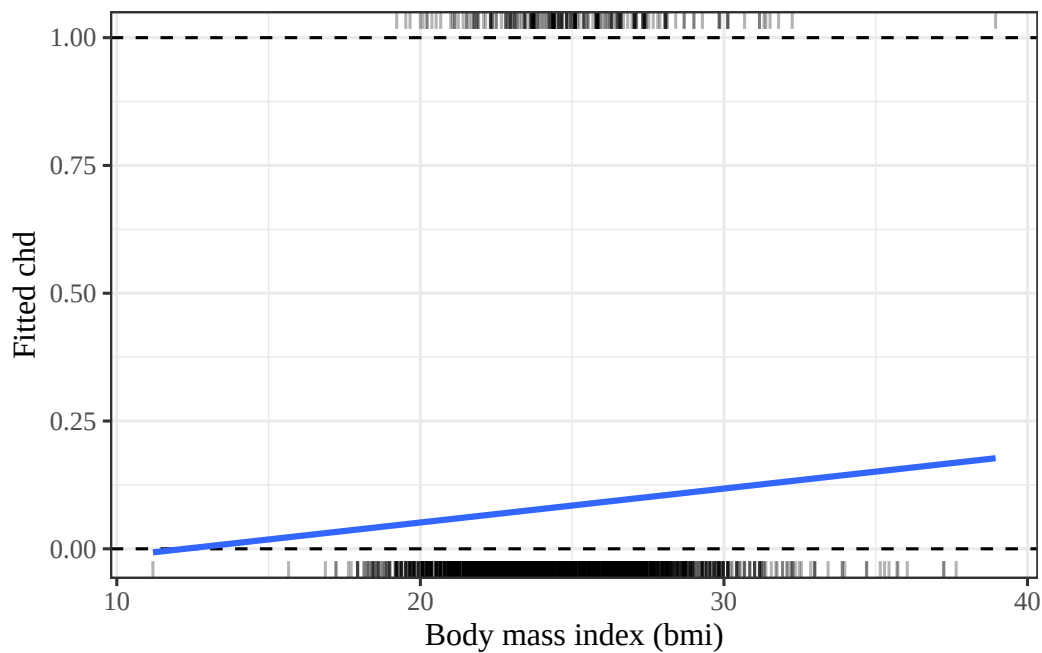


Figure 1: Fitted line of chd on bmi in the WCGS data, with a rug of the men with chd = 1 along the top and chd = 0 along the bottom.

```
breaks <- seq(floor(min(wcgs$bmi)), ceiling(max(wcgs$bmi)), by = 1)
bars <- function(chd_value) {
  counts <- hist(wcgs$bmi[wcgs$chd == chd_value], breaks = breaks, plot = FALSE)$counts
  data.frame(xmin = head(breaks, -1), xmax = tail(breaks, -1),
             height = 0.3 * counts / max(counts))
}
bars0 <- bars(0)
bars1 <- bars(1)
ggplot(wcgs, aes(bmi, chd)) +
  geom_rect(data = bars0, aes(xmin = xmin, xmax = xmax, ymin = 0, ymax = height),
            inherit.aes = FALSE, alpha = 0.4) +
  geom_rect(data = bars1, aes(xmin = xmin, xmax = xmax, ymin = 1 - height, ymax = 1),
            inherit.aes = FALSE, alpha = 0.4) +
  geom_smooth(method = "lm", formula = y ~ x, se = FALSE) +
  labs(x = "Body mass index (bmi)", y = "Fitted chd")
```

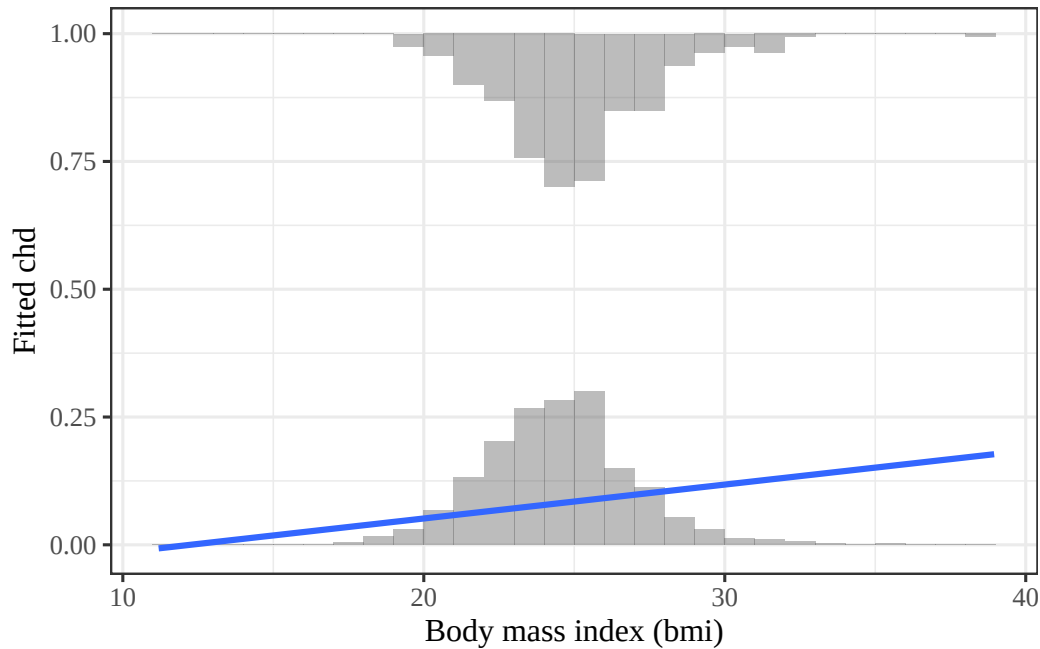


Figure 2: The same line, with a histogram of `bmi` for the men with `chd` = 0 standing on the line $y = 0$ and one for the men with `chd` = 1 hanging from the line $y = 1$. Each histogram is scaled so its tallest bar is 0.3 high.

1.2 Python

```
import numpy as np
import pandas as pd

wcgs = pd.read_csv("data/wcgs.csv")
slope, intercept = np.polyfit(wcgs["bmi"], wcgs["chd"], 1)
fitted_bmi = intercept + slope * wcgs["bmi"]
print(intercept, slope, fitted_bmi.min(), fitted_bmi.max(),
      ((fitted_bmi < 0) | (fitted_bmi > 1)).sum())

-0.08118199613838391 0.00663439543322767 -0.006943111240566277 0.17722770598583387 1

import matplotlib.pyplot as plt

fig, ax = plt.subplots()
for chd, side in [(0, 0.0), (1, 1.0)]:
    bmi_chd = wcgs.loc[wcgs["chd"] == chd, "bmi"]
    ax.plot(bmi_chd, [side] * len(bmi_chd), "|", color="gray", alpha=0.3);
ax.axhline(0, linestyle="--", color="black");
ax.axhline(1, linestyle="--", color="black");
ax.plot(wcgs["bmi"], fitted_bmi);
ax.set_xlabel("Body mass index (bmi)");
ax.set_ylabel("Fitted chd");
plt.show()
```

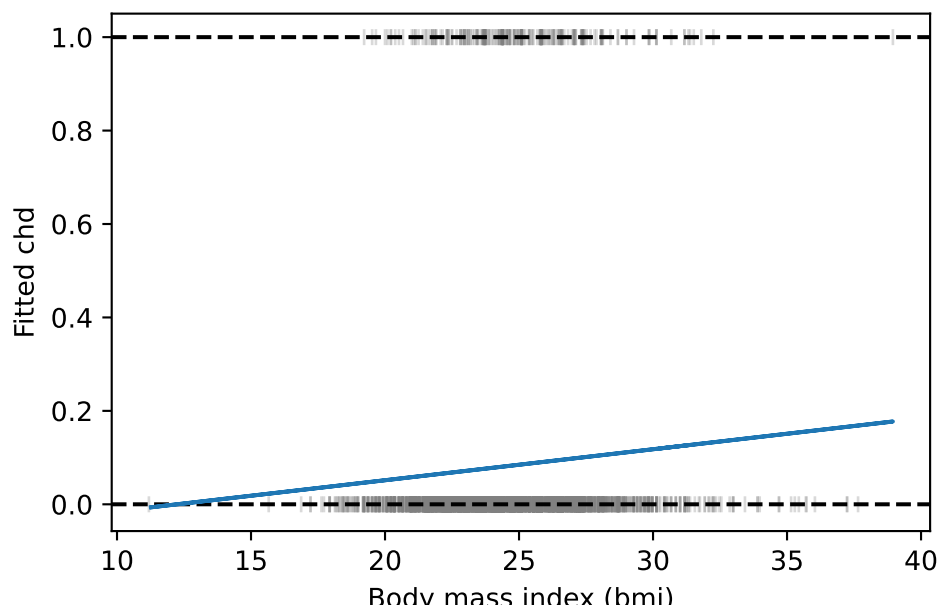


Figure 3: Fitted line of chd on bmi in the WCGS data, with a rug of the men with chd = 1 along the top and chd = 0 along the bottom.

```
breaks = np.arange(np.floor(wcgs["bmi"].min()), np.ceil(wcgs["bmi"].max()) + 1)

fig, ax = plt.subplots()
for chd, base, sign in [(0, 0.0, 1), (1, 1.0, -1)]:
    counts, _ = np.histogram(wcgs.loc[wcgs["chd"] == chd, "bmi"], bins=breaks)
    ax.bar(breaks[:-1], sign * 0.3 * counts / counts.max(), width=1,
           bottom=base, align="edge", alpha=0.4);
ax.plot(wcgs["bmi"], fitted_bmi);
ax.set_xlabel("Body mass index (bmi)");
ax.set_ylabel("Fitted chd");
plt.show()
```

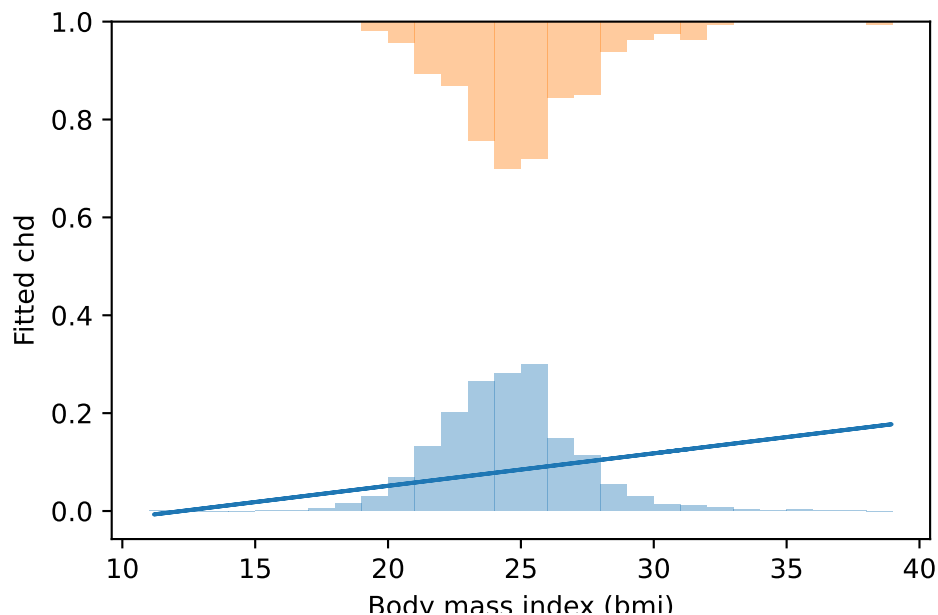


Figure 4: The same line, with a histogram of `bmi` for the men with `chd` = 0 standing on the line $y = 0$ and one for the men with `chd` = 1 hanging from the line $y = 1$. Each histogram is scaled so its tallest bar is 0.3 high.

The fitted line is $\hat{y} = -0.081 + 0.0066 \times \text{bmi}$. Its fitted values range from -0.007 to 0.177, so 1 of the 3154 men (the one with the lowest `bmi`) has a fitted value outside $[0, 1]$. A value outside $[0, 1]$ cannot be a valid probability. More generally, a nonconstant affine function of $\tilde{x}$ is unbounded in both directions: here the fitted line exceeds 1 above a `bmi` of 163.

Exercise 1.2 (Squared error on probabilities). A patient has $\hat{y} = 0.7$ and another has $\hat{y} = 0.9$. Under the squared-error loss, what does the difference of 0.2 imply about the two patients? State whether that implication is appropriate for probabilities.

Solution

Solution 1.2. Under squared-error loss, a difference of 0.2 between 0.7 and 0.9 is weighted identically to a difference of 0.2 between 0.3 and 0.5, because squared error treats equal differences in probability as equal regardless of where the probabilities are in $[0, 1]$. However, moving from probability 0.7 to 0.9 more than triples the odds^a (from $7/3 \approx 2.33$ to 9), whereas moving from 0.3 to 0.5 increases the odds by a factor of $7/3$ (from $3/7 \approx 0.43$ to 1). Equal differences in probability do not correspond to equal multiplicative changes in odds.

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-odds>

Exercise 1.3 (A correctly classified point far from the others). Suppose the data are one-dimensional and perfectly ordered: every point with $x < 0$ has $y = 0$, and every point with $x > 0$ has $y = 1$. Add a single new point far out at $x = 100$, also with $y = 1$, a point the model already classifies correctly. Say what happens to the least-squares fit, and why that change is undesirable when the prediction is correct.

Solution

Solution 1.3. Adding the outlying point tilts the least-squares line toward that point and shifts the [decision boundary](#).

For the six points $x = -3, -2, -1, 1, 2, 3$ with labels $y = 0, 0, 0, 1, 1, 1$, least squares gives $\hat{y} = 0.5 + 0.2143x$, which classifies all six observations using the threshold $\hat{y} > 0.5$ without error.

Now add a seventh observation at $x = 100$ with $y = 1$ — an observation that the six-point line already places far on the $y = 1$ side of the boundary. The least-squares line becomes $\hat{y} = 0.4903 + 0.005681x$. The slope^a decreases by a factor of about 38, and the six original fitted values are now between 0.473 and 0.507. At $x = 1$, the fitted value is $0.4959 < 0.5$, so the observation at $x = 1$ is misclassified as 0 even though its label is 1.

Adding an observation that is far on the correct side of the boundary causes the model to misclassify an observation it previously classified accurately.

This failure occurs because squared error penalizes $(\hat{y}_i - y_i)^2$, and at $x = 100$ the six-point line predicts $\hat{y} = 21.9$, which contributes about 438 to the sum of squared residuals. Reducing the slope reduces that large squared [residual](#) at $x = 100$ much more than it increases the squared residuals of the six points near the origin. Least squares therefore penalizes predictions that exceed 1 even when $y_i = 1$, which distorts the classification boundary [3, Sec. 4.1.3].

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope>

Exercise 1.4 (Requirements for a classifier's output). From Exercise 1.1, Exercise 1.2 and Exercise 1.3, state the two requirements that a classifier's output should satisfy and that $\tilde{\beta} \cdot \tilde{x}$ does not.

Solution

Solution 1.4. Two properties are required:

The output must be a valid probability. It must be in $[0, 1]$ for every x by construction, so that it can be interpreted as a class probability and combined with decision thresholds.

The loss must saturate on the side of the true label. Once an observation is classified accurately with high confidence, increasing its margin further should reduce the loss by almost nothing, so that points far on the correct side of the boundary do not distort the boundary near contested points.

The first requirement is satisfied by mapping the linear score into $(0, 1)$ with the logistic sigmoid^a function (Exercise 2.1). The second requirement is satisfied by fitting the parameters by maximum Bernoulli likelihood^b ([log loss](#)) rather than least squares (Exercise 5.5). Both follow from modeling the log-odds^c as a linear function of x in Exercise 3.1.

^a<https://morrison-lab.github.io/mds/functions.html#def-logistic-function>

^b<https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik>

^c<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

Exercise 1.5 (Fitting the thresholded line). Check the first part of Solution 1.3 by computer, in Python, R or Julia.

Fit the least-squares line to the six points $x = -3, -2, -1, 1, 2, 3$ with labels $y = 0, 0, 0, 1, 1, 1$, and report its slope^a, its intercept^b, the input^c at which it crosses 0.5, and which of the six points the rule $\hat{y} > 0.5$ misclassifies.

Compute the slope and intercept directly from the least-squares formulas rather than calling a regression function.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope>

^b<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

^c<https://morrison-lab.github.io/sds/model-validation.html#def-data-set>

Solution

Solution 1.5.

1.3 Python

```
x = [-3, -2, -1, 1, 2, 3]
y = [0, 0, 0, 1, 1, 1]

def fit(xs, ys):
    n = len(xs)
    xbar, ybar = sum(xs) / n, sum(ys) / n
    sxy = sum((xi - xbar) * (yi - ybar) for xi, yi in zip(xs, ys))
    sxx = sum((xi - xbar) ** 2 for xi in xs)
    m = sxy / sxx
    return m, ybar - m * xbar

def report(label, xs, ys):
    m, b = fit(xs, ys)
    cut = (0.5 - b) / m
    bad = [xi for xi, yi in zip(x, y) if (m * xi + b > 0.5) != (yi == 1)]
    print("%s slope %.6f intercept %.4f boundary %.2f wrong: %s"
          % (label, m, b, cut, ", ".join(map(str, bad)) if bad else "none"))

report("without:", x, y)

without: slope 0.214286 intercept 0.5000 boundary 0.00 wrong: none
```

1.4 R

```
x <- c(-3, -2, -1, 1, 2, 3)
y <- c(0, 0, 0, 1, 1, 1)

fit <- function(xs, ys) {
  m <- sum((xs - mean(xs)) * (ys - mean(ys))) / sum((xs - mean(xs))^2)
  c(m, mean(ys) - m * mean(xs))
}

report <- function(label, xs, ys) {
  mb <- fit(xs, ys)
  cut <- (0.5 - mb[2]) / mb[1]
  bad <- x[(mb[1] * x + mb[2] > 0.5) != (y == 1)]
  cat(sprintf("%s slope %.6f intercept %.4f boundary %.2f wrong: %s\n",
              label, mb[1], mb[2], cut,
              if (length(bad) == 0) "none" else paste(bad, collapse = ", ")))
}

report("without:", x, y)

without: slope 0.214286 intercept 0.5000 boundary 0.00 wrong: none
```

1.5 Julia

```

using Printf, Statistics

x = [-3, -2, -1, 1, 2, 3];
y = [0, 0, 0, 1, 1, 1];

function fit(xs, ys)
    m = sum((xs .- mean(xs)) .* (ys .- mean(ys))) / sum((xs .- mean(xs)) .^ 2)
    (m, mean(ys) - m * mean(xs))
end;

function report(label, xs, ys)
    m, b = fit(xs, ys)
    cut = (0.5 - b) / m
    bad = x[(m .* x .+ b .> 0.5) .!= (y .== 1)]
    @printf("%s slope %.6f intercept %.4f boundary %.2f wrong: %s\n",
        label, m, b, cut, isempty(bad) ? "none" : join(bad, ", "))
end;

report("without:", x, y)

without: slope 0.214286 intercept 0.5000 boundary 0.00 wrong: none

without <- fit(x, y)
cut_without <- (0.5 - without[2]) / without[1]

```

The fit crosses 0.5 at $x = 0$, midway between the classes, and classifies all six points correctly.

Exercise 1.6 (Watching an extreme point move the thresholded line). Check the second part of Solution 1.3 by computer, in Python, R or Julia, using the six points of Exercise 1.5. Refit with the seventh point $x = 100$, $y = 1$ added, and report the same four quantities, still evaluating classification accuracy on the original six points. Compute the slope and intercept directly from the least-squares formulas rather than calling a regression function.

Solution

Solution 1.6. The code reuses `x`, `y`, `fit` and `report` from Solution 1.5.

1.6 Python

```
report("with :", x + [100], y + [1])
with : slope 0.005681 intercept 0.4903 boundary 1.71 wrong: 1
```

1.7 R

```
report("with :", c(x, 100), c(y, 1))
with : slope 0.005681 intercept 0.4903 boundary 1.71 wrong: 1
```

1.8 Julia

```
report("with :", [x; 100], [y; 1])
with : slope 0.005681 intercept 0.4903 boundary 1.71 wrong: 1
```

```
with_far <- fit(c(x, 100), c(y, 1))
cut_with <- (0.5 - with_far[2]) / with_far[1]
flipped <- x[(with_far[1] * x + with_far[2] > 0.5) != (y == 1)]
```

The second fit crosses 0.5 at $x \approx 1.71$, past the class-1 point at $x = 1$, which it now classifies as class 0. Adding a point that the classifier already classified correctly causes it to misclassify a different point. Although this example is one-dimensional and uses a small sample, neither property is essential. The same vulnerability to extreme points applies to every coefficient of a multi-feature least-squares fit, and the further the point is from the boundary, the larger its influence.

2 From a line to a probability

To keep the predictions in $[0, 1]$, we apply the **logistic function**²

$$\text{expit}(\eta) = \frac{1}{1 + e^{-\eta}} \in (0, 1),$$

to $\vec{\beta} \cdot \tilde{x}$, which ranges over the whole real line. Setting the probability that the label is 1 to the result gives a **logistic regression** model.

The curve is also called a *sigmoid*³, for its S-shape.

²<https://morrison-lab.github.io/mds/functions.html#def-logistic-function>

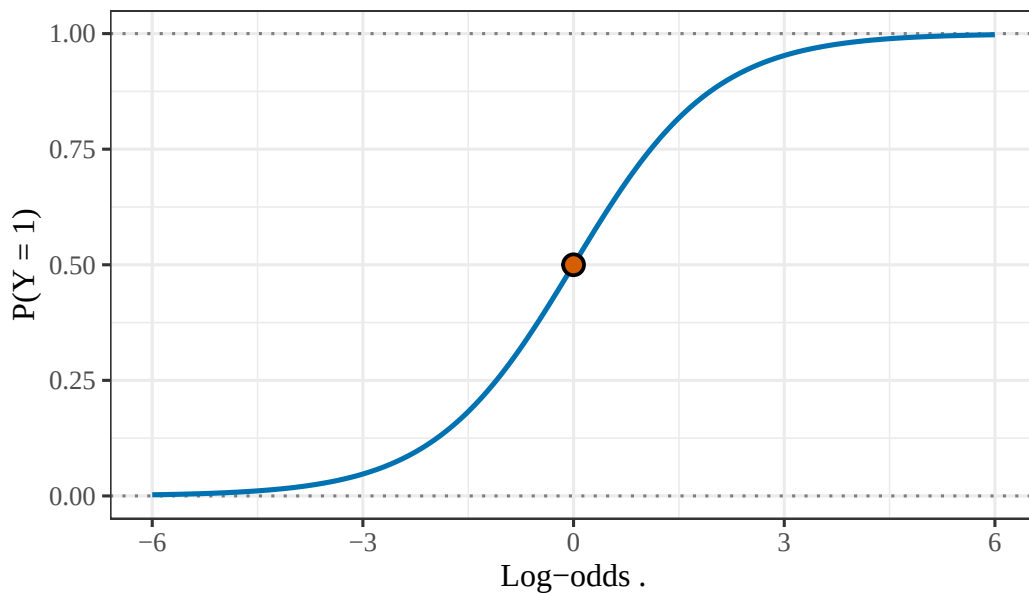
³<https://morrison-lab.github.io/mds/functions.html#def-logistic-function>

2.1 R

```
library(ggplot2)

sigmoid_df <- data.frame(eta = seq(-6, 6, length.out = 241))
sigmoid_df$p <- plogis(sigmoid_df$eta)
midpoint <- data.frame(eta = 0, p = plogis(0))

ggplot(sigmoid_df) +
  aes(eta, p) +
  geom_hline(yintercept = c(0, 1), linetype = "dotted", colour = "grey50") +
  geom_line(linewidth = 0.9, colour = "#0072B2") +
  geom_point(
    data = midpoint, shape = 21, size = 3, stroke = 1,
    fill = "#D55E00", colour = "black"
  ) +
  labs(x = "Log-odds ", y = "P(Y = 1)")
```



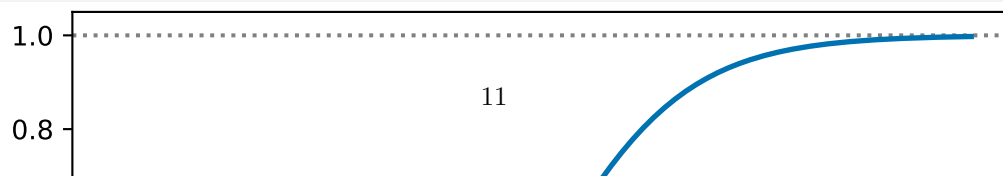
(a)

2.2 Python

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.special import expit

f_grid = np.linspace(-6, 6, 241)

fig, ax = plt.subplots(figsize=(5.5, 3.2), layout="constrained")
ax.axhline(0, color="grey", linestyle=":");
ax.axhline(1, color="grey", linestyle=":");
ax.plot(f_grid, expit(f_grid), color="#0072B2", linewidth=2);
ax.plot(0, expit(0), "o", markerfacecolor="#D55E00", markeredgecolor="black",
        markersize=8);
ax.set_xlabel("Log-odds ");
ax.set_ylabel("P(Y = 1)");
plt.show()
```



11

Definition 2.1 (Logistic regression). For a binary label $Y \in \{0, 1\}$ and feature vector $\tilde{x}$, a **logistic regression** model sets the probability that the label is 1 to the logistic function^a of a linear score with coefficient vector $\vec{\beta}$:

$$\Pr(Y = 1 \mid \tilde{x}) = \text{expit}(\vec{\beta} \cdot \tilde{x}).$$

It is a linear regression model whose output passes through the logistic function.

^a<https://morrison-lab.github.io/mds/functions.html#def-logistic-function>

Example 2.1 (Logistic regression of coronary heart disease on age at enrollment, stratified by behavior pattern). In the WCGS data (`data/wcgs.csv`^a), the column `dibpat` records each man's behavior pattern, type A or type B. The two patterns have different rates of coronary heart disease (`chd`), so we stratify: we fit a separate **logistic regression** of `chd` on age at enrollment (`age`) within each pattern, rather than combining the men. Each fit gives the fitted probability of disease at each age for its pattern:

```
wcgs_exm <- read.csv("data/wcgs.csv")
wcgs_fits <- lapply(
  split(wcgs_exm, wcgs_exm$dibpat),
  function(d) glm(chd ~ age, family = binomial, data = d)
)
exm_ages <- data.frame(age = c(40, 45, 50, 55))
cbind(
  exm_ages,
  sapply(wcgs_fits, predict, newdata = exm_ages, type = "response")
)
```

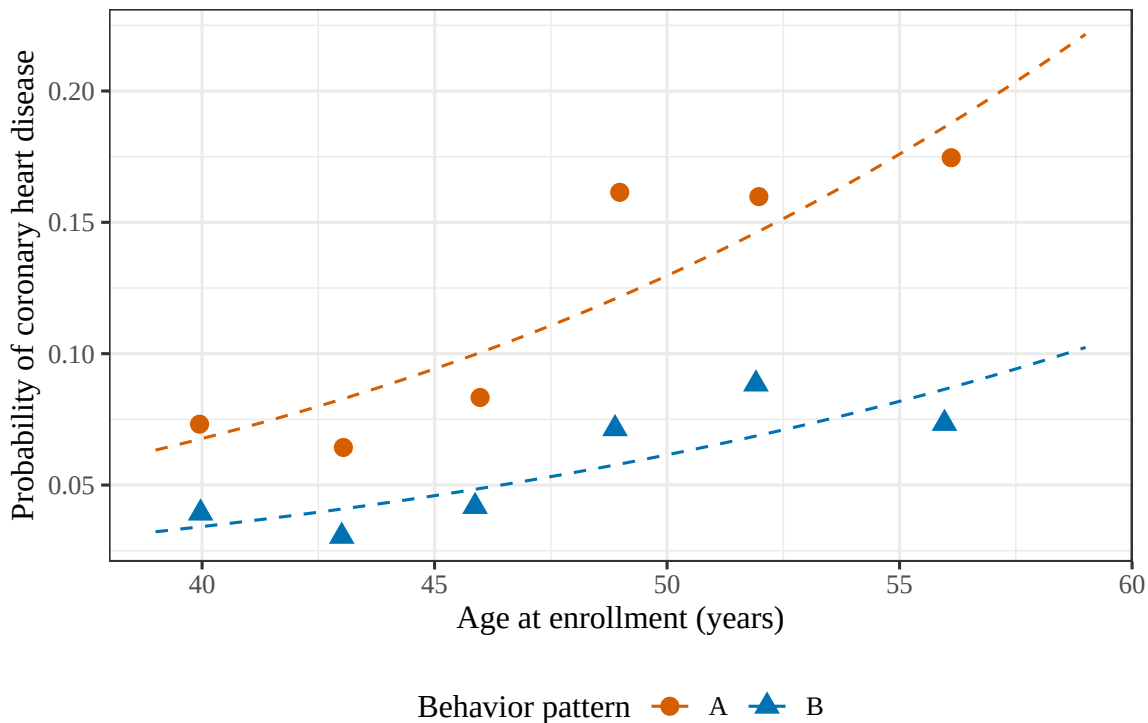
```
# A tibble: 4 x 3
  age      A      B
<dbl> <dbl> <dbl>
1   40 0.0677 0.0342
2   45 0.0942 0.0460
3   50 0.130  0.0615
4   55 0.176  0.0819
```

In both strata the fitted probability rises with age at enrollment, and it stays between 0 and 1 at every age. At every age shown, type A has the higher fitted probability.

^a<https://morrison-lab.github.io/lds/chapters/data/wcgs.csv>

2.3 R

```
library(ggplot2)
lrf_bins <- cut(wcgs_exm$age, breaks = c(38, 41, 44, 47, 50, 53, 59))
lrf_points <- aggregate(
  cbind(age, chd) ~ lrf_bins + dibpat, data = wcgs_exm, FUN = mean
)
lrf_curve <- do.call(rbind, lapply(names(wcgs_fits), function(type) {
  d <- data.frame(age = seq(min(wcgs_exm$age), max(wcgs_exm$age), length.out = 100))
  d$prob <- predict(wcgs_fits[[type]], newdata = d, type = "response")
  d$dibpat <- type
  d
}))
ggplot(lrf_curve, aes(age, prob, colour = dibpat)) +
  geom_line(linetype = "dashed") +
  geom_point(data = lrf_points, aes(y = chd, shape = dibpat), size = 3) +
  scale_colour_manual(values = c(A = "#D55E00", B = "#0072B2")) +
  labs(x = "Age at enrollment (years)", y = "Probability of coronary heart disease",
       colour = "Behavior pattern", shape = "Behavior pattern")
```



(a)

2.4 Python

```
import matplotlib.pyplot as plt

points_lrf = r.lrf_points
curve_lrf = r.lrf_curve
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
for pattern, colour, marker in [("A", "#D55E00", "o"), ("B", "#0072B2", "^")]:
    curve = curve_lrf[curve_lrf["dibpat"] == pattern]
    points = points_lrf[points_lrf["dibpat"] == pattern]
    ax.plot(curve["age"], curve["prob"], color=colour, linestyle="--",
            label=f"type {pattern}, fitted logistic regression");
    ax.plot(points["age"], points["chd"], color=colour, marker=marker, markersize=8,
            linestyle="", label=f"type {pattern}, proportion in each age group");
ax.set_xlabel("Age at enrollment (years)");
ax.set_ylabel("Probability of coronary heart disease");
```

We separate two questions. The next two exercises ask whether this particular curve is a reasonable choice. Exercise 3.1 asks why it is the *inevitable* choice, rather than one S-shaped curve among many.

Exercise 2.1 (What the logistic function does). $\text{expit}(\eta) = \frac{1}{1 + e^{-\eta}}$. Establish its properties directly, because every one of them is used later. Evaluate $\text{expit}(0)$, and find $\lim_{\eta \rightarrow \infty} \text{expit}(\eta)$ and $\lim_{\eta \rightarrow -\infty} \text{expit}(\eta)$.

Solution

Solution 2.1.

$$\begin{aligned}\text{expit}(0) &= 1/(1 + e^0) \\ &= 1/2.\end{aligned}$$

As $\eta \rightarrow \infty$, $e^{-\eta} \rightarrow 0$, so $\text{expit}(\eta) \rightarrow 1$. As $\eta \rightarrow -\infty$, $e^{-\eta} \rightarrow \infty$, so $\text{expit}(\eta) \rightarrow 0$.

It is strictly increasing — Solution 2.3 gives a derivative that is positive everywhere — so expit is a bijection from $(-\infty, \infty)$ onto $(0, 1)$, which is what Exercise 3.1 needed. The interval is open: expit never reaches 0 or 1, so the model is never completely certain.

Exercise 2.2 (What the logistic function does (part 2)). Use the setup of Exercise 2.1. Show that $\text{expit}(-\eta) = 1 - \text{expit}(\eta)$. Say what this says about the model's treatment of the two classes.

Solution

Solution 2.2.

$$\begin{aligned}\text{expit}(-\eta) &= \frac{1}{1 + e^{\eta}} \\ &= \frac{e^{-\eta}}{e^{-\eta}(1 + e^{\eta})} \\ &= \frac{e^{-\eta}}{e^{-\eta} + 1} \\ &= \frac{1 + e^{-\eta} - 1}{1 + e^{-\eta}} \\ &= 1 - \text{expit}(\eta)\end{aligned}$$

This says the model is **symmetric^a in the two classes**. Since

$$\begin{aligned}\Pr[Y = 0 \mid \tilde{x}] &= 1 - \text{expit}(\vec{\beta} \cdot \tilde{x}) \\ &= \text{expit}(-\vec{\beta} \cdot \tilde{x}),\end{aligned}$$

fitting the model to the relabelled data $y \mapsto 1 - y$ returns exactly $-\vec{\beta}$ and the same fitted probabilities. Which class you call 1 is a matter of labeling, not of modelling.

^a<https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-symmetric-matrix>

Exercise 2.3 (What the logistic function does (part 3)). Use the setup of Exercise 2.1. Show that $\text{expit}'(\eta) = \text{expit}(\eta)(1 - \text{expit}(\eta))$.

Solution

Solution 2.3. Expressing $\text{expit}(\eta)$ as $(1 + e^{-\eta})^{-1}$ and applying the chain rule gives

$$\begin{aligned}
\text{expit}'(\eta) &= -(1 + e^{-\eta})^{-2} \cdot \frac{d}{d\eta}(1 + e^{-\eta}) \\
&= -(1 + e^{-\eta})^{-2} \cdot (-e^{-\eta}) \\
&= \frac{e^{-\eta}}{(1 + e^{-\eta})^2}
\end{aligned}$$

Rewriting the numerator as $e^{-\eta} = (1 + e^{-\eta}) - 1$ and splitting the fraction gives

$$\begin{aligned}
\text{expit}'(\eta) &= \frac{(1 + e^{-\eta}) - 1}{(1 + e^{-\eta})^2} \\
&= \frac{1}{1 + e^{-\eta}} - \frac{1}{(1 + e^{-\eta})^2} \\
&= \text{expit}(\eta) - \text{expit}(\eta)^2 \\
&= \text{expit}(\eta)(1 - \text{expit}(\eta))
\end{aligned}$$

This identity simplifies the gradient of [logistic regression](#) in Exercise 6.1: the derivative of the model is expressed directly in terms of the model's output $\text{expit}(\eta)$.

Exercise 2.4 (What the logistic function does (part 4)). Use the setup of Exercise 2.1.

Where is expit' largest, and what is its value there? Give $\text{expit}'(4)$ to three significant figures, and say what the comparison implies for a point the model already classifies confidently.

Solution

Solution 2.4. The [derivative](#) $\text{expit}' = \text{expit}(1 - \text{expit})$ is a downward-opening parabola in expit , maximized at $\text{expit} = 1/2$, which corresponds to $\eta = 0$. Its maximum value is $1/4$.

At $\eta = 4$, $\text{expit}(4) = 0.98201$, so

$$\begin{aligned}
\text{expit}'(4) &= 0.98201 \times 0.01799 \\
&= 0.0177
\end{aligned}$$

to three significant figures, which is about $1/14$ of the maximum. A point with large $|\eta|$ therefore is in a nearly flat region of expit : a small change in $\vec{\beta}$ changes that point's predicted probability by very little. For a point that is confidently *correct*, this flatness provides the saturation that Exercise 1.4 required and that a linear model lacks. Because $\text{expit}(\eta)$ is already close to the true label, its [residual](#) is near zero and it contributes almost nothing to the gradient with respect to $\vec{\beta}$, so a point with a large positive margin does not outweigh contested points near the boundary.

For a point that is confidently *wrong*, the flatness of expit makes squared-error loss applied to $\text{expit}(\vec{\beta} \cdot \tilde{x})$ stall during optimization, because the expit' factor in the gradient of squared error shrinks when the prediction is furthest from the label. Exercise 6.1 shows that the [log loss](#) of Exercise 5.3 cancels the expit' factor, so a confidently wrong point contributes a gradient of magnitude close to $\|\tilde{x}_i\|$. Log loss preserves saturation for confidently correct predictions while removing it for confidently wrong predictions.

Exercise 2.5 (What the logistic function does (part 5)). Use the setup of Exercise 2.1.

There is a second sigmoid^a in common use, $\tanh(\eta) = \frac{e^{\eta} - e^{-\eta}}{e^{\eta} + e^{-\eta}}$. Prove $\text{expit}(\eta) = \frac{1}{2}(1 + \tanh(\eta/2))$.

^a<https://morrison-lab.github.io/mds/functions.html#def-logistic-function>

Solution

Solution 2.5. Starting from the right-hand side, write $\tanh(\eta/2)$ in exponentials and multiply numerator and denominator by $e^{\eta/2}$:

$$\begin{aligned}\tanh(\eta/2) &= \frac{e^{\eta/2} - e^{-\eta/2}}{e^{\eta/2} + e^{-\eta/2}} \\ &= \frac{e^{\eta} - 1}{e^{\eta} + 1}\end{aligned}$$

Then add 1, halve, and simplify:

$$\begin{aligned}\frac{1}{2}(1 + \tanh(\eta/2)) &= \frac{1}{2} \cdot \frac{(e^{\eta} + 1) + (e^{\eta} - 1)}{e^{\eta} + 1} \\ &= \frac{e^{\eta}}{e^{\eta} + 1} \\ &= \frac{1}{1 + e^{-\eta}} \\ &= \text{expit}(\eta)\end{aligned}$$

Thus the two sigmoid^a functions are related by a rescaling of the input^b and an affine transformation of the output. The function $\tanh$ has range $(-1, 1)$, which matches the $y \in \{+1, -1\}$ coding of Exercise 5.6 and is often used in hidden layers of neural networks where a zero-centered output is convenient. The logistic function^c expit has range $(0, 1)$, which matches probabilities.

^a<https://morrison-lab.github.io/mds/functions.html#def-logistic-function>

^b<https://morrison-lab.github.io/sds/model-validation.html#def-data-set>

^c<https://morrison-lab.github.io/mds/functions.html#def-logistic-function>

The logistic function⁶, also called the sigmoid⁷, is defined in the math prerequisites.

Example 2.2 (Values of the logistic function in the WCGS data). Fit the logistic model of `chd` on `bmi` to the WCGS data of Exercise 1.1, with $\pi(x) = \text{expit}(\beta_0 + \beta_1 x)$.

```
wcgs_logit <- glm(chd ~ bmi, family = binomial, data = wcgs)
coef(wcgs_logit)
```

```
(Intercept)      bmi
-4.50856101  0.08427102
```

For a man with `bmi = 25`, the linear score is $\vec{\beta} \cdot \vec{x} = -2.402$, and the logistic function^a maps it to $\text{expit}(-2.402) = 0.083$. The model therefore gives about a 8.3% probability that a man with a body mass index of 25 has an incident case of coronary heart disease.

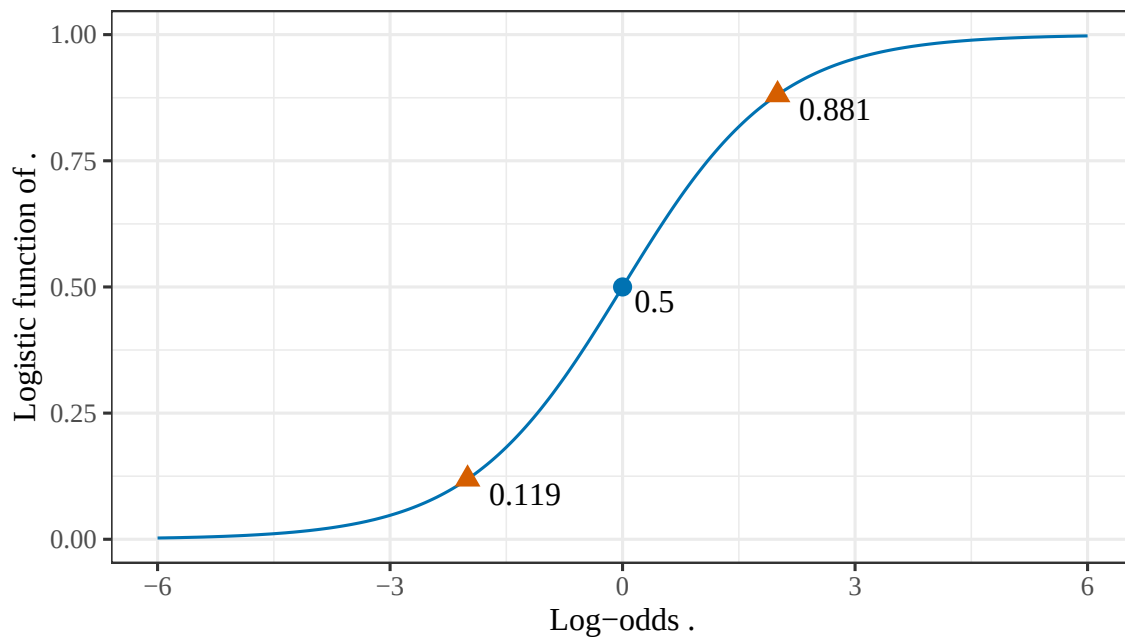
^a<https://morrison-lab.github.io/mds/functions.html#def-logistic-function>

⁶<https://morrison-lab.github.io/mds/functions.html#def-logistic-function>

⁷<https://morrison-lab.github.io/mds/functions.html#def-logistic-function>

2.5 R

```
library(ggplot2)
lfv_curve <- data.frame(z = seq(-6, 6, length.out = 241))
lfv_curve$p <- plogis(lfv_curve$z)
lfv_pts <- data.frame(z = c(-2, 0, 2))
lfv_pts$p <- plogis(lfv_pts$z)
lfv_pts$kind <- ifelse(lfv_pts$z == 0, " = 0", " = -2 and 2")
ggplot(lfv_curve, aes(z, p)) +
  geom_line(colour = "#0072B2") +
  geom_point(data = lfv_pts, aes(colour = kind, shape = kind), size = 3) +
  geom_text(data = lfv_pts, aes(label = round(p, 3)), hjust = -0.3, vjust = 1.2) +
  scale_colour_manual(values = c(" = 0" = "#0072B2", " = -2 and 2" = "#D55E00")) +
  scale_shape_manual(values = c(" = 0" = 16, " = -2 and 2" = 17)) +
  labs(x = "Log-odds ", y = "Logistic function of ", colour = NULL, shape = NULL)
```



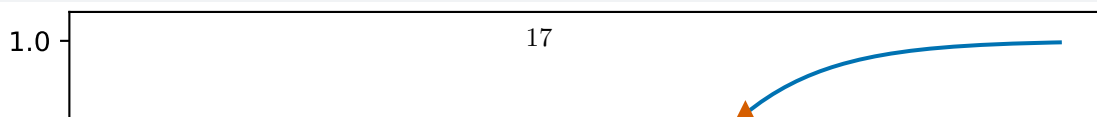
▲ . = -2 and 2 ● . = 0

(a)

2.6 Python

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.special import expit

grid_lfv = np.linspace(-6, 6, 241)
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.plot(grid_lfv, expit(grid_lfv), color="#0072B2");
for z_lfv, colour, marker in [(-2, "#D55E00", "^"), (0, "#0072B2", "o"), (2, "#D55E00", "^")]:
    ax.plot(z_lfv, expit(z_lfv), color=colour, marker=marker, markersize=8, linestyle="");
    ax.annotate(f"{expit(z_lfv):.3f}", (z_lfv, expit(z_lfv)), xytext=(8, -14),
               textcoords="offset points");
ax.set_xlabel("Log-odds ");
ax.set_ylabel("Logistic function of ");
plt.show()
```



i Remark (Intercept convention)

Remark 2.1 (Intercept convention). Logistic regression models the probability of the label as $\Pr(Y = 1 \mid \tilde{x}) = \text{expit}(\beta_0 + \vec{\beta} \cdot \tilde{x})$, where β_0 is the intercept and $\vec{\beta}$ holds the slopes. A formula that shows β_0 separately, as the [decision boundary](#) does, uses $\vec{\beta}$ for the vector of slopes only. The notation choices page^a explains this convention and how to read a formula that absorbs the intercept into $\vec{\beta}$.

^a<https://morrison-lab.github.io/macros/notation-choices.html#sec-intercept>

Theorem 2.1 (Symmetry of the logistic function). *For every $\eta \in \mathbb{R}$,*

$$\text{expit}(-\eta) = 1 - \text{expit}(\eta).$$

Hence relabelling the classes ($y \mapsto 1 - y$) negates every coefficient, including the intercept^a, and leaves the fitted probabilities unchanged.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

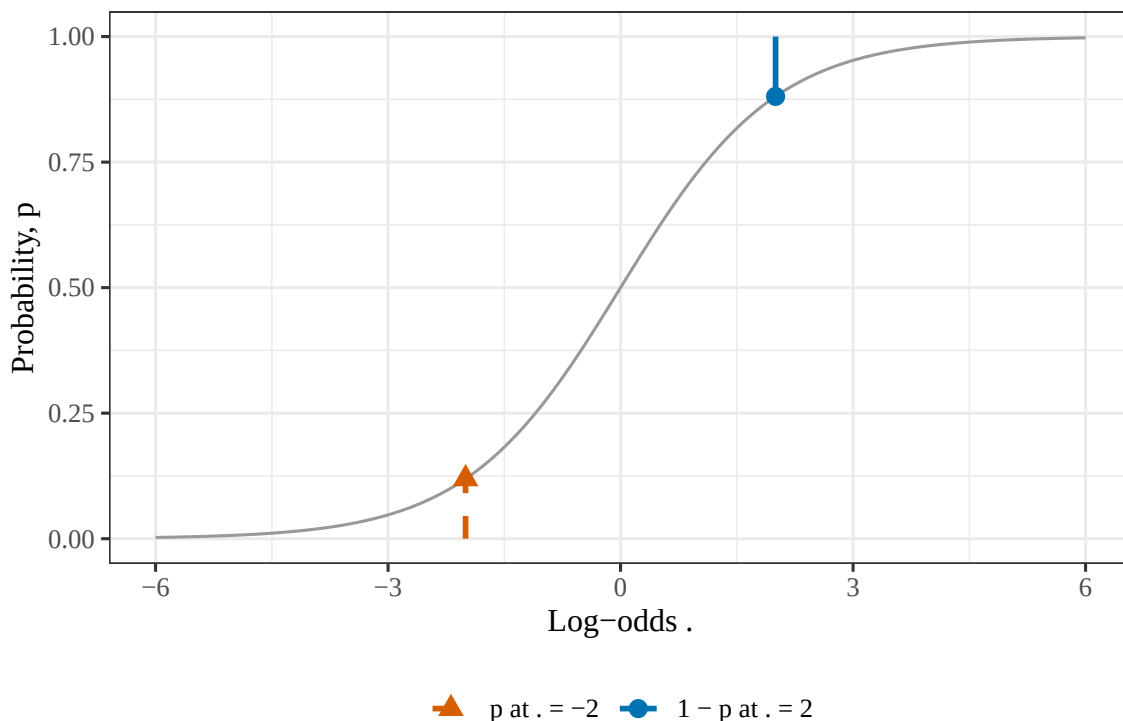
i Proof

Proof. By Solution 2.2. □

Example 2.3 (Symmetry for a man with a body mass index of 25). In the model of Example 2.2, a man with $\text{bmi} = 25$ has the score $\eta = -2.402$. Here $\text{expit}(-\eta) = 0.917$ and $1 - \text{expit}(\eta) = 0.917$ agree. The model gives $\Pr(Y = 1 \mid \tilde{x}) = \text{expit}(\eta) = 0.083$ to this man, and therefore $\Pr(Y = 0 \mid \tilde{x}) = \text{expit}(-\eta) = 0.917$ to him.

2.7 R

```
library(ggplot2)
lsy_curve <- data.frame(z = seq(-6, 6, length.out = 241))
lsy_curve$p <- plogis(lsy_curve$z)
lsy_seg <- data.frame(
  z = c(-2, 2),
  from = c(0, plogis(2)),
  to = c(plogis(-2), 1),
  kind = factor(c("p at = -2", "1 - p at = 2"), levels = c("p at = -2", "1 - p at = 2"))
)
ggplot(lsy_curve, aes(z, p)) +
  geom_line(colour = "grey60") +
  geom_segment(
    data = lsy_seg,
    aes(x = z, xend = z, y = from, yend = to, colour = kind, linetype = kind),
    linewidth = 1
  ) +
  geom_point(
    data = data.frame(z = c(-2, 2), p = plogis(c(-2, 2))), kind = lsy_seg$kind,
    aes(colour = kind, shape = kind), size = 3
  ) +
  scale_colour_manual(values = c("p at = -2" = "#D55E00", "1 - p at = 2" = "#0072B2")) +
  scale_linetype_manual(values = c("p at = -2" = "dashed", "1 - p at = 2" = "solid")) +
  scale_shape_manual(values = c("p at = -2" = 17, "1 - p at = 2" = 16)) +
  labs(x = "Log-odds ", y = "Probability, p", colour = NULL, linetype = NULL, shape = NULL)
```



(a)

2.8 Python

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.special import expit

grid_lsy = np.linspace(-6, 6, 241)
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.plot(grid_lsy, expit(grid_lsy), color="grey");
```

Theorem 2.2 (Derivative of the logistic function). *For every $\eta \in \mathbb{R}$,*

$$\text{expit}'(\eta) = \text{expit}(\eta)(1 - \text{expit}(\eta)).$$

i Proof

Proof. By Solution 2.3 and Solution 2.4. □

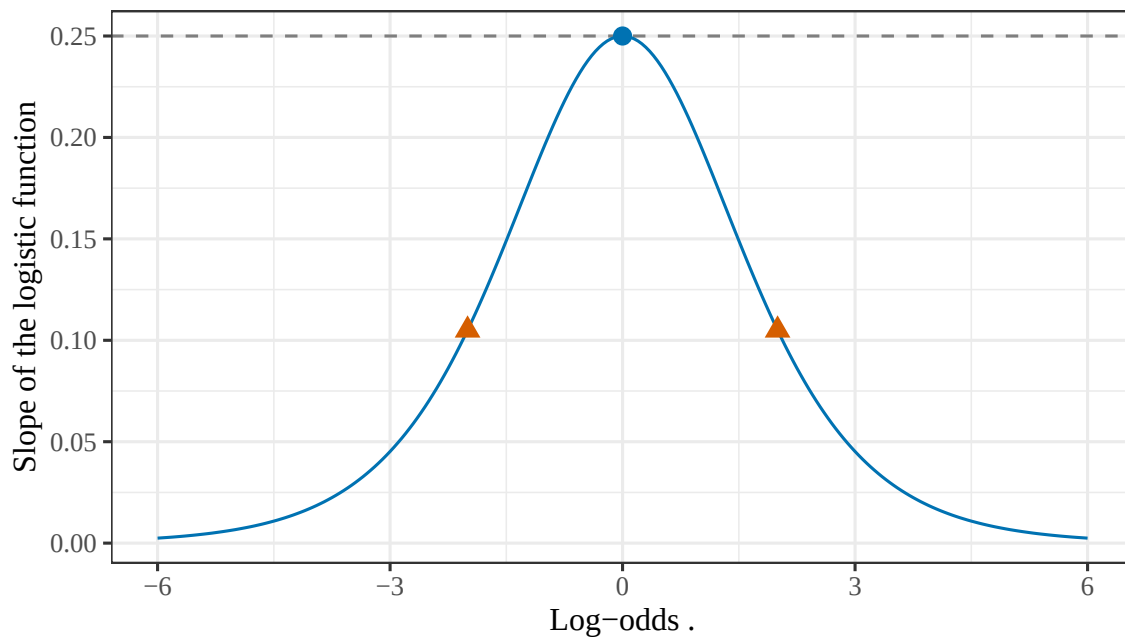
Example 2.4 (Slope of the logistic function for a man with a body mass index of 25). At the score $\eta = -2.402$ of the man with $\text{bmi} = 25$ in Example 2.3, the slope^a of the sigmoid^b is $\text{expit}'\{\eta\} = \text{expit}(\eta)(1 - \text{expit}(\eta)) = 0.0761$. It is the same at $-\eta$, because Theorem 2.1 swaps the two factors $\text{expit}(\eta)$ and $1 - \text{expit}(\eta)$. At $\eta = 0$ the slope is $\text{expit}'\{0\} = 0.25$, 3.3 times as large.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope>

^b<https://morrison-lab.github.io/mds/functions.html#def-logistic-function>

2.9 R

```
library(ggplot2)
lde_slope <- function(z) plogis(z) * (1 - plogis(z))
lde_pts <- data.frame(z = c(-2, 0, 2))
lde_pts$slope <- lde_slope(lde_pts$z)
lde_pts$kind <- factor(ifelse(lde_pts$z == 0, " = 0", " = -2 and 2"), levels = c(" = 0", " = -2 and 2"))
ggplot(data.frame(z = c(-6, 6)), aes(z)) +
  geom_hline(yintercept = lde_slope(0), linetype = "dashed", colour = "grey50") +
  geom_function(fun = lde_slope, colour = "#0072B2", n = 241) +
  geom_point(data = lde_pts, aes(y = slope, colour = kind, shape = kind), size = 3) +
  scale_colour_manual(values = c(" = 0" = "#0072B2", " = -2 and 2" = "#D55E00")) +
  scale_shape_manual(values = c(" = 0" = 16, " = -2 and 2" = 17)) +
  labs(x = "Log-odds ", y = "Slope of the logistic function", colour = NULL, shape = NULL)
```



● . = 0 ▲ . = -2 and 2

(a)

2.10 Python

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.special import expit

def slope_lde(z):
    return expit(z) * (1 - expit(z))

grid_lde = np.linspace(-6, 6, 241)
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.axhline(slope_lde(0), color="grey", linestyle="--");
ax.plot(grid_lde, slope_lde(grid_lde), color="#0072B2");
ax.plot(0, slope_lde(0), color="#0072B2", marker="o", markersize=8, linestyle="",
        label=" = 0");
ax.plot([-2, 2], slope_lde(np.array([-2, 2])), color="#D55E00", marker="^", markersize=8,
        linestyle="", label=" = -2 and 2");
ax.set_xlabel("Log-odds ");
ax.set_ylabel("Slope of the logistic function");
fig.legend(loc="outside lower center", ncol=2);
plt.show()
```

Theorem 2.3 (The logistic function is steepest at zero). *For every $\eta \in \mathbb{R}$, $\text{expit}'(\eta) \leq 1/4$, with equality if and only if $\eta = 0$.*

i Proof

Proof. By Theorem 2.2, $\text{expit}'(\eta) = \text{expit}(\eta)(1 - \text{expit}(\eta))$. Completing the square in $\text{expit}(\eta)$ gives

$$\text{expit}(\eta)(1 - \text{expit}(\eta)) = \frac{1}{4} - \left(\text{expit}(\eta) - \frac{1}{2}\right)^2.$$

The square is nonnegative, so $\text{expit}'(\eta) \leq 1/4$, with equality if and only if $\text{expit}(\eta) = 1/2$. Since the logistic function^a is strictly increasing with $\text{expit}(0) = 1/2$, this happens only at $\eta = 0$. $\square$

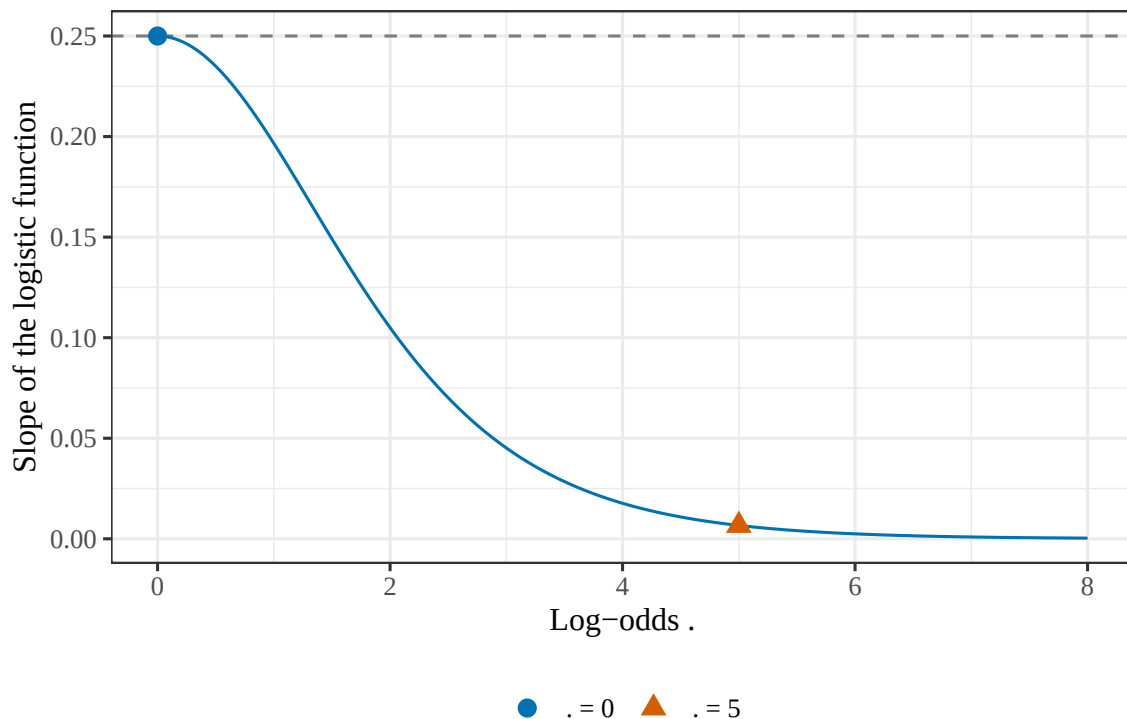
^a<https://morrison-lab.github.io/mds/functions.html#def-logistic-function>

Example 2.5 (Slope across the body mass indices in the WCGS data). The slope^a at $\eta = 0$ is 0.25, the bound of Theorem 2.3. In the model of Example 2.2, the body mass indices in the WCGS data run from 11.19 to 38.95, with scores from -3.57 to -1.23 . Over these men the slope runs from 0.0267 to 0.1754, below the bound because no man has a score of 0, and smaller the further a score is from 0.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope>

2.11 R

```
library(ggplot2)
ldb_slope <- function(z) plogis(z) * (1 - plogis(z))
ldb_pts <- data.frame(z = c(0, 5))
ldb_pts$slope <- ldb_slope(ldb_pts$z)
ldb_pts$kind <- factor(c(" = 0", " = 5"), levels = c(" = 0", " = 5"))
ggplot(data.frame(z = c(0, 8)), aes(z)) +
  geom_hline(yintercept = ldb_slope(0), linetype = "dashed", colour = "grey50") +
  geom_function(fun = ldb_slope, colour = "#0072B2", n = 241) +
  geom_point(data = ldb_pts, aes(y = slope, colour = kind, shape = kind), size = 3) +
  scale_colour_manual(values = c(" = 0" = "#0072B2", " = 5" = "#D55E00")) +
  scale_shape_manual(values = c(" = 0" = 16, " = 5" = 17)) +
  labs(x = "Log-odds ", y = "Slope of the logistic function", colour = NULL, shape = NULL)
```



(a)

2.12 Python

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.special import expit

def slope_ldb(z):
    return expit(z) * (1 - expit(z))

grid_ldb = np.linspace(0, 8, 161)
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.axhline(slope_ldb(0), color="grey", linestyle="--");
ax.plot(grid_ldb, slope_ldb(grid_ldb), color="#0072B2");
ax.plot(0, slope_ldb(0), color="#0072B2", marker="o", markersize=8, linestyle="", label=" = 0");
ax.plot(5, slope_ldb(5), color="#D55E00", marker="^", markersize=8, linestyle="", label=" = 5");
ax.set_xlabel("Log-odds ");
ax.set_ylabel("Slope of the logistic function");
fig.legend(loc="outside lower center", ncol=2);
plt.show()
```



Watch

- Hastie and Tibshirani introduce logistic regression^a [4], video 4.2.
- Hutchinson teaches logistic regression^b (17 min) [1].
- Starmer introduces logistic regression^c [5].

^ahttps://www.youtube.com/watch?v=kr_Be9NVXOM

^bhttps://facultyweb.cs.wvu.edu/~hutchib2/video_lectures/data371/#logistic_regression

^c<https://www.youtube.com/watch?v=yYKR4sgzI8>

Read

GeeksforGeeks explains:

- logistic regression^a [6]
- the sigmoid function^b [7]

^a<https://www.geeksforgeeks.org/machine-learning/understanding-logistic-regression/>

^b<https://www.geeksforgeeks.org/machine-learning/derivative-of-the-sigmoid-function/>

Read

The University of Washington's CSE 446 slides cover classification and logistic regression^a [8], Lecture 13.

^a<https://courses.cs.washington.edu/courses/cse446/26sp/schedule/lectures/13-classification%20and%20logistic%20regression.pdf>

Exercise 2.6 (What the coefficients do to the surface). Set $\beta_0 = 0$, so the surface $p(\tilde{x}) = \text{expit}(\vec{\beta} \cdot \tilde{x})$ passes through 1/2 at the origin, and let $\vec{\beta} \neq \vec{0}$. Compare the surfaces for $\vec{\beta}$ and $10\vec{\beta}$. What is the same about them?

Solution

Solution 2.6. Same shape, steeper. Both surfaces rise from 0 to 1 along the same direction $\vec{\beta}$ and are constant on the same family of parallel hyperplanes $\vec{\beta} \cdot \tilde{x} = c$, because $10\vec{\beta} \cdot \tilde{x} = 10c$ is constant exactly when $\vec{\beta} \cdot \tilde{x}$ is.

Exercise 2.7 (What the coefficients do to the surface (part 2)). Use the setup of Exercise 2.6. What is different about them?

Solution

Solution 2.7. The surfaces differ in how fast they rise. Going from $p = 0.02$ to $p = 0.98$ corresponds to going from $\eta = \log(0.02/0.98)$ to $\eta = \log(0.98/0.02)$, an interval of length $2 \log 49 \approx 7.78$ in η . Since $\eta = \vec{\beta} \cdot \tilde{x}$, that range is covered by a distance $7.78/\|\vec{\beta}\|$ along $\vec{\beta}$ — so $10\vec{\beta}$ covers it in one-tenth the distance required by $\vec{\beta}$.

Exercise 2.8 (What the coefficients do to the surface (part 3)). Use the setup of Exercise 2.6. Give the set of points where the two agree.

Solution

Solution 2.8. They agree exactly on the hyperplane $\vec{\beta} \cdot \tilde{x} = 0$, where both read $1/2$, and nowhere else: for $\eta \neq 0$, $\text{expit}(10\eta) \neq \text{expit}(\eta)$ since expit is injective.

Exercise 2.9 (What the coefficients do to the surface (part 4)). Use the setup of Exercise 2.6. Compare the surfaces for $\vec{\beta}$ and $\vec{\beta}/10$.

Solution

Solution 2.9. Same shape, flatter, by the same argument with the factor inverted. As the scale falls toward zero the surface flattens toward the constant^a $1/2$, which is the model that ignores $\tilde{x}$ altogether and predicts the class prevalence.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

Exercise 2.10 (What the coefficients do to the surface (part 5)). Use the setup of Exercise 2.6. Compare the surfaces for $\vec{\beta}$ and $-\vec{\beta}$.

Solution

Solution 2.10. The surface is reflected. $\text{expit}(-\eta) = 1 - \text{expit}(\eta)$ from Exercise 2.1, so the surface for $-\vec{\beta}$ is 1 minus the surface for $\vec{\beta}$: it falls where the other rises, at the same rate, about the same hyperplane. This is the class relabelling of Exercise 2.2 seen geometrically.

Exercise 2.11 (What the coefficients do to the surface (part 6)). Use the setup of Exercise 2.6. From Exercise 2.6 to Exercise 2.10, separate what $\vec{\beta}$ encodes into a *direction* and a *magnitude*, and say in one sentence what each controls.

Solution

Solution 2.11. Direction. $\vec{\beta}/\|\vec{\beta}\|$ fixes which way the probability increases, and so fixes the orientation of the [decision boundary](#) — the boundary is perpendicular to it (Exercise 4.1).

Magnitude. $\|\vec{\beta}\|$ fixes how sharply the probability turns over as $\tilde{x}$ crosses that boundary, and so fixes the model's confidence: large $\|\vec{\beta}\|$ is a model that commits to 0 or 1 just outside the boundary, small $\|\vec{\beta}\|$ a model that gives intermediate probabilities.

Because the magnitude is a *separate* quantity from the direction, Exercise 11.1 is possible, where the direction is determined and the magnitude grows without bound, and ℓ_2 regularization restricts it, as Solution 11.5 of that exercise works out.

Exercise 2.12 (What the coefficients do to the surface (part 7)). Use the setup of Exercise 2.6. Now allow a nonzero intercept^a β_0 . State how the surface changes, and give the point along the direction $\vec{\beta}$ at which the surface equals $1/2$.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

Solution

Solution 2.12. Nothing about direction or steepness changes. β_0 shifts the surface along $\vec{\beta}$ without rotating or scaling it, so the family of parallel hyperplanes is unchanged and only the hyperplane on

which the probability equals $1/2$ moves.

That hyperplane is $\beta_0 + \vec{\beta} \cdot \tilde{x} = 0$. Along the direction $\vec{\beta}$, setting $\tilde{x} = t \vec{\beta} / \|\vec{\beta}\|$, the surface equals $1/2$ at $t = -\beta_0 / \|\vec{\beta}\|$.

So β_0 is a signed distance in units of $1/\|\vec{\beta}\|$, which is why it is reported alongside the other coefficients but interpreted differently: it is the log odds^a at $\tilde{x} = \vec{0}$, which is a meaningful quantity only if $\tilde{x} = \vec{0}$ is a case that could occur. Centring the features makes it one.

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

i Watch

- Andrew Ng derives logistic regression, from 46:18^a [9], Lecture 3.
- Weinberger introduces logistic regression, from 26:12^b [10], Lecture 11.

^a<https://www.youtube.com/watch?v=het9HFqo1TQ&t=2778s>

^b<https://www.youtube.com/watch?v=GnkDzIOxfzI&t=1572s>

3 What the coefficients mean

A coefficient in a linear regression is a change in the response per unit change in a predictor. Here the response is a probability, and expit is not linear, so a unit change in a predictor does not change the probability by a fixed amount.

The log odds are the quantity the model makes linear. Deriving which transformation of a probability a linear model can act on yields the sigmoid¹⁴ rather than assuming it.

Exercise 3.1 (Deriving the sigmoid instead of choosing it). Exercise 1.4 requires a function that maps $\vec{\beta} \cdot \tilde{x}$ into $(0, 1)$. Many functions map $\mathbb{R}$ into $(0, 1)$, so choosing one requires a modelling criterion. Rather than choosing a function that maps into $(0, 1)$ and then asking how to interpret the parameters, we specify what the linear score $\vec{\beta} \cdot \tilde{x}$ represents and solve for the function.

Let $p(\tilde{x}) = \Pr[Y = 1 \mid \tilde{x}]$.

The odds^a of an event of probability p are $p/(1-p)$. State the range of the odds as p varies over $(0, 1)$, and the range of the log-odds^b $\log \frac{p}{1-p}$.

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-odds>

^b<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

Solution

Solution 3.1. As p varies over $(0, 1)$, the odds^a $p/(1-p)$ range over $(0, \infty)$: the numerator approaches 0 as $p \rightarrow 0$, and the denominator approaches 0 as $p \rightarrow 1$. The logarithm maps $(0, \infty)$ bijectively onto $(-\infty, \infty)$.

So the log-odds^b transform maps a probability in $(0, 1)$ onto the entire real line $(-\infty, \infty)$.

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-odds>

^b<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

Exercise 3.2 (Deriving the sigmoid instead of choosing it (part 2)). Use the setup of Exercise 3.1.

A linear score $\vec{\beta} \cdot \tilde{x}$ ranges over $(-\infty, \infty)$. Given Exercise 3.1, state the modelling assumption that equates the linear score to a function of p with the same range $(-\infty, \infty)$, without truncation or clipping.

¹⁴<https://morrison-lab.github.io/mds/functions.html#def-logistic-function>

Solution

Solution 3.2. The assumption is that the **log odds^a** are linear in the features:

$$\log \frac{p(\tilde{x})}{1 - p(\tilde{x})} = \vec{\beta} \cdot \tilde{x}$$

Both sides have range $(-\infty, \infty)$, so the equation can hold for every $\tilde{x} \in \mathbb{R}^p$ without clipping or restricting $\vec{\beta}$. This linearity of the log odds is the defining modeling assumption of **logistic regression**; the sigmoid^b formula for $p(\tilde{x})$ follows by algebra.

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

^b<https://morrison-lab.github.io/mds/functions.html#def-logistic-function>

Exercise 3.3 (Deriving the sigmoid instead of choosing it (part 3)). Use the setup of Exercise 3.1.

Solve $\log \frac{p}{1-p} = \eta$ for p . Show every step.

Solution

Solution 3.3. Exponentiate, then clear the denominator:

$$\begin{aligned}\frac{p}{1-p} &= e^\eta \\ p &= (1-p)e^\eta \\ p &= e^\eta - p e^\eta \\ p + p e^\eta &= e^\eta \\ p(1 + e^\eta) &= e^\eta \\ p &= \frac{e^\eta}{1 + e^\eta}\end{aligned}$$

Divide numerator and denominator by e^η :

$$\begin{aligned}p &= \frac{1}{1 + e^{-\eta}} \\ &= \text{expit}(\eta)\end{aligned}$$

Both expressions^a for $\text{expit}(\eta)$ appear in the literature; the second form $1/(1 + e^{-\eta})$ avoids floating-point overflow for large positive η , where e^η overflows.

^a<https://morrison-lab.github.io/mds/expressions-algorithms.html#def-expression>

Exercise 3.4 (Deriving the sigmoid instead of choosing it (part 4)). Use the setup of Exercise 3.1.

The function you obtained is the logistic function^a, expit . State in one sentence what the coefficient β_j therefore represents.

^a<https://morrison-lab.github.io/mds/functions.html#def-logistic-function>

Solution

Solution 3.4. β_j is the change in the log odds^a of $Y = 1$ per one-unit increase in x_j , holding the other features fixed. Equivalently, on the odds^b scale, increasing x_j by one unit multiplies the odds by e^{β_j} (Theorem 3.2).

A coefficient of $0.693 \approx \log 2$ multiplies the odds by 2; a coefficient of 0 leaves them unchanged; a negative coefficient multiplies them by a factor in $(0, 1)$.

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>
^b<https://morrison-lab.github.io/pds/probability-basics.html#def-odds>

Exercise 3.5 (Deriving the sigmoid instead of choosing it (part 5)). Use the setup of Exercise 3.1. The earlier parts, Exercise 3.1 to Exercise 3.4, do not show that the assumption in Exercise 3.2 holds for any given dataset. Describe how to check this assumption empirically on a dataset, and give one example in which you would expect it to fail.

Solution

Solution 3.5. It is a modelling assumption that can be checked against data. The assumption states that the log odds^a are *linear* in each feature, so plotting the empirical log odds against x_j — binning x_j , computing the proportion $\hat{p}$ of observations with $y = 1$ in each bin, and plotting $\log \frac{\hat{p}}{1-\hat{p}}$ against the bin midpoint — should yield points scattered around a straight line. Systematic curvature indicates that the log odds are not linear in x_j .

This assumption fails whenever a feature has an interior optimum. For blood pressure and mortality risk, both very low and very high blood pressure are associated with elevated risk, so the empirical log odds are U-shaped and no linear function of blood pressure alone fits. Adding a quadratic feature x_j^2 restores linearity in the parameters, because the model assumes linearity in the *features* rather than in the raw measurements.

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

The odds¹⁵ of an event are defined in the probability prerequisites.

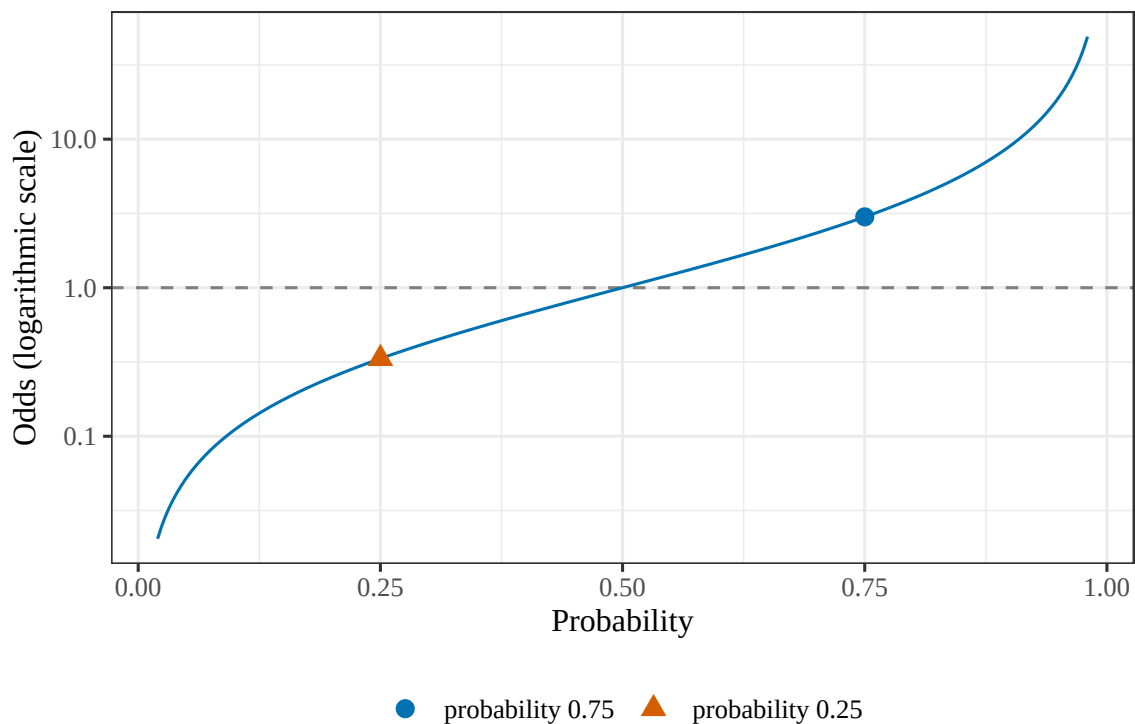
Example 3.1 (Odds of coronary heart disease in the WCGS data). In the WCGS data of Exercise 1.1, the proportion of men with an incident case of coronary heart disease is $\pi = 0.081$. The odds^a are $\omega = \pi/(1 - \pi) = 0.089$, about 1 to 11.3.

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-odds>

¹⁵<https://morrison-lab.github.io/pds/probability-basics.html#def-odds>

3.1 R

```
library(ggplot2)
od_curve <- data.frame(prob = seq(0.02, 0.98, length.out = 241))
od_curve$odds <- od_curve$prob / (1 - od_curve$prob)
od_pts <- data.frame(prob = c(0.25, 0.75))
od_pts$odds <- od_pts$prob / (1 - od_pts$prob)
od_pts$kind <- factor(c("probability 0.25", "probability 0.75"),
                      levels = c("probability 0.75", "probability 0.25"))
ggplot(od_curve, aes(prob, odds)) +
  geom_hline(yintercept = 1, linetype = "dashed", colour = "grey50") +
  geom_line(colour = "#0072B2") +
  geom_point(data = od_pts, aes(colour = kind, shape = kind), size = 3) +
  scale_y_log10() +
  scale_colour_manual(values = c("probability 0.75" = "#0072B2", "probability 0.25" = "#D55E00")) +
  scale_shape_manual(values = c("probability 0.75" = 16, "probability 0.25" = 17)) +
  labs(x = "Probability", y = "Odds (logarithmic scale)", colour = NULL, shape = NULL)
```



(a)

3.2 Python

```
import matplotlib.pyplot as plt
import numpy as np

prob_od = np.linspace(0.02, 0.98, 241)
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.axhline(1, color="grey", linestyle="--");
ax.plot(prob_od, prob_od / (1 - prob_od), color="#0072B2");
ax.plot(0.75, 0.75 / 0.25, color="#0072B2", marker="o", markersize=8, linestyle="",
        label="probability 0.75");
ax.plot(0.25, 0.25 / 0.75, color="#D55E00", marker="^", markersize=8, linestyle="",
        label="probability 0.25");
ax.set_yscale("log");
ax.set_xlabel("Probability");
ax.set_ylabel("Odds (logarithmic scale)"); 29
fig.legend(loc="outside lower center", ncol=2);
plt.show()
```

The log-odds¹⁷, also called the logit¹⁸, are defined there too.

Example 3.2 (Log-odds of coronary heart disease in the WCGS data). In the WCGS data of Exercise 1.1, the proportion with coronary heart disease is $\pi = 0.081$ and the odds^a are 0.089 (Example 3.1). The log-odds^b are $\eta = -2.422$. The log-odds are negative because the probability is below 1/2.

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-odds>

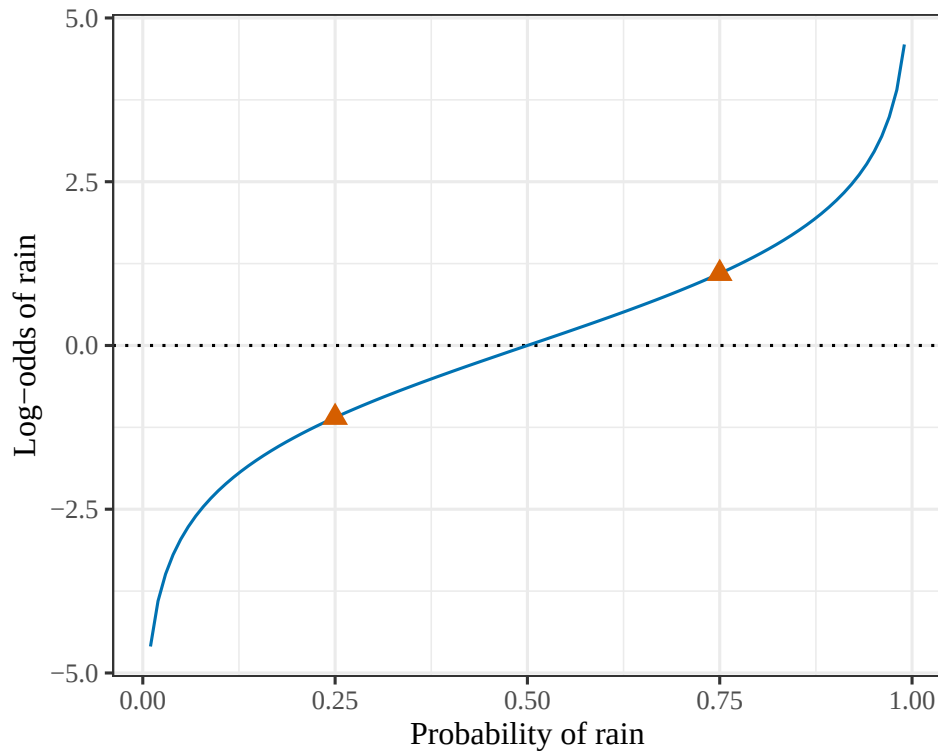
^b<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

¹⁷<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

¹⁸<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

3.3 R

```
library(ggplot2)
lo_marks <- data.frame(prob = c(0.25, 0.75))
lo_marks$logodds <- log(lo_marks$prob / (1 - lo_marks$prob))
ggplot(data.frame(prob = c(0.01, 0.99)), aes(prob)) +
  geom_hline(yintercept = 0, linetype = "dotted") +
  geom_function(fun = function(p) log(p / (1 - p)), colour = "#0072B2") +
  geom_point(data = lo_marks, aes(prob, logodds), colour = "#D55E00", shape = 17, size = 3) +
  labs(x = "Probability of rain", y = "Log-odds of rain")
```



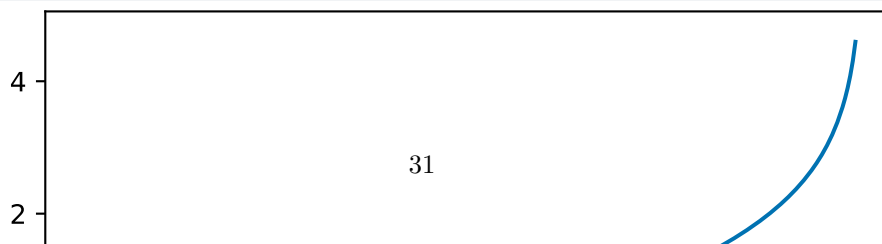
(a)

3.4 Python

```
import matplotlib.pyplot as plt
import numpy as np

def log_odds_lo(prob):
    return np.log(prob / (1 - prob))

grid_lo = np.linspace(0.01, 0.99, 300)
marks_lo = np.array([0.25, 0.75])
fig, ax = plt.subplots(figsize=(5, 4), layout="constrained")
ax.axhline(0, color="black", linestyle=":");
ax.plot(grid_lo, log_odds_lo(grid_lo), color="#0072B2");
ax.plot(marks_lo, log_odds_lo(marks_lo), color="#D55E00", marker="^", markersize=8, linestyle="");
ax.set_xlabel("Probability of rain");
ax.set_ylabel("Log-odds of rain");
plt.show()
```



Theorem 3.1 (Probability from the log-odds). *If the log-odds^a of an event are η , its probability is*

$$\pi = \text{expit}(\eta) = \frac{1}{1 + e^{-\eta}}.$$

Hence the logistic regression model $\log \frac{\pi(\tilde{x})}{1 - \pi(\tilde{x})} = \vec{\beta} \cdot \tilde{x}$ is the same as $\pi(\tilde{x}) = \text{expit}(\vec{\beta} \cdot \tilde{x})$.

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

i Proof

Proof. By Solution 3.3 and Solution 3.4. □

Example 3.3 (From log-odds to probability in the WCGS data). The log-odds^a of coronary heart disease in the WCGS data are $\eta = -2.422$ (Example 3.2). These log-odds give $\pi = \text{expit}(\eta) = 0.081$, the proportion of Example 3.1, as they must.

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

Theorem 3.2 (The odds multiplier of a coefficient). *In the logistic regression model $\pi(\tilde{x}) = \text{expit}(\vec{\beta} \cdot \tilde{x})$, increasing x_j by one unit, holding the other predictors fixed, multiplies the odds^a by e^{β_j} .*

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-odds>

i Proof

Proof. By Theorem 3.1 the log-odds^a are $\eta = \vec{\beta} \cdot \tilde{x}$, so the odds^b are e^η . Increasing x_j by one unit, with the other predictors fixed, changes the score from η to $\eta + \beta_j$. The odds therefore change from e^η to $e^{\eta + \beta_j} = e^\eta e^{\beta_j}$, a factor of e^{β_j} . □

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

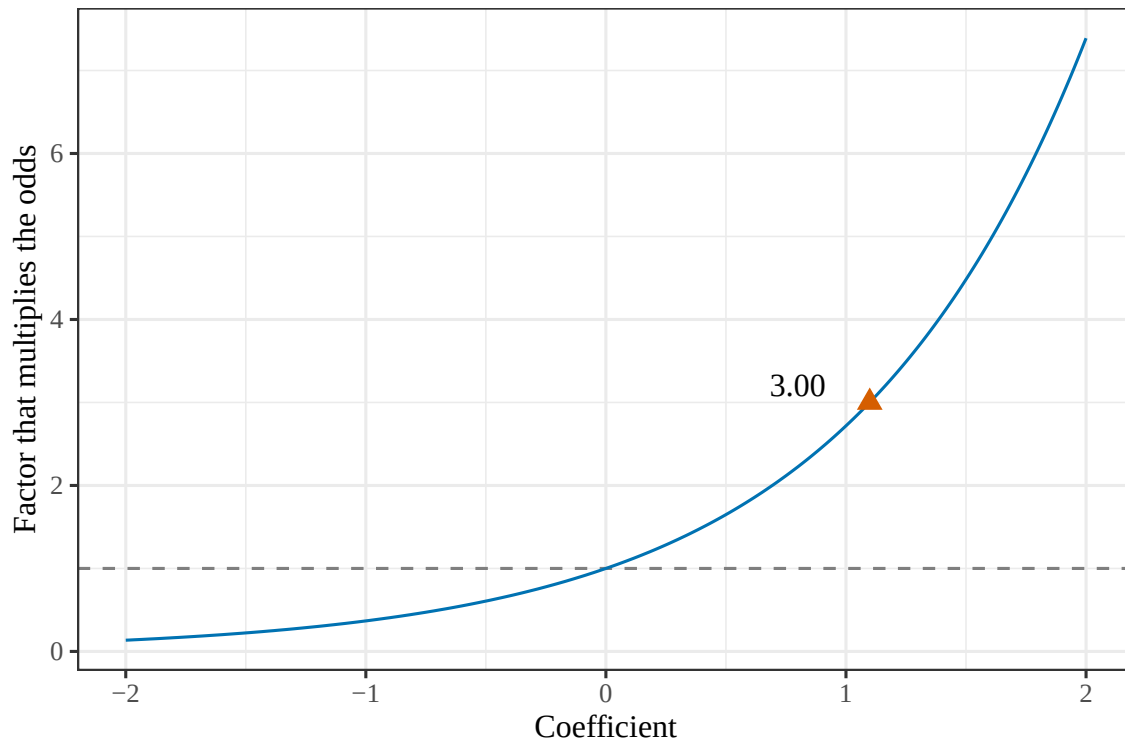
^b<https://morrison-lab.github.io/pds/probability-basics.html#def-odds>

Example 3.4 (The odds multiplier of body mass index). In the model of Example 2.2, the coefficient of `bmi` is $\beta_1 = 0.084$. One more unit of body mass index multiplies the odds^a of coronary heart disease by $e^{\beta_1} = 1.088$, an increase of about 8.8%.

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-odds>

3.5 R

```
library(ggplot2)
om_curve <- data.frame(coef = seq(-2, 2, length.out = 241))
om_curve$factor <- exp(om_curve$coef)
om_pt <- data.frame(coef = 1.099, factor = exp(1.099))
ggplot(om_curve, aes(coef, factor)) +
  geom_hline(yintercept = 1, linetype = "dashed", colour = "grey50") +
  geom_line(colour = "#0072B2") +
  geom_point(data = om_pt, colour = "#D55E00", shape = 17, size = 3) +
  geom_text(data = om_pt, aes(label = sprintf("%.2f", factor)), nudge_x = -0.3, nudge_y = 0.2) +
  labs(x = "Coefficient", y = "Factor that multiplies the odds")
```

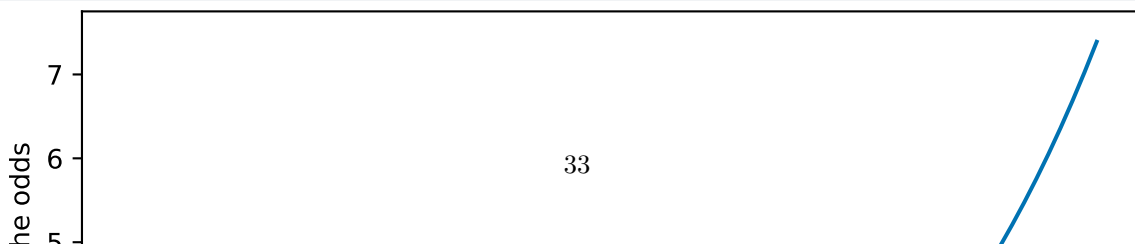


(a)

3.6 Python

```
import matplotlib.pyplot as plt
import numpy as np

coef_om = np.linspace(-2, 2, 241)
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.axhline(1, color="grey", linestyle="--");
ax.plot(coef_om, np.exp(coef_om), color="#0072B2");
ax.plot(1.099, np.exp(1.099), color="#D55E00", marker="^", markersize=8, linestyle="");
ax.annotate(f"{np.exp(1.099):.2f}", (1.099, np.exp(1.099)), xytext=(-34, 2),
           textcoords="offset points");
ax.set_xlabel("Coefficient");
ax.set_ylabel("Factor that multiplies the odds");
plt.show()
```



Corollary 3.1 (The odds ratio of a coefficient). *Let:*

- $\tilde{x}$ be a vector of predictors in the logistic model $\pi(\tilde{x}) = \text{expit}(\vec{\beta} \cdot \tilde{x})$.
- $\tilde{x}'$ be the same vector with x_j increased by one unit.

Then the odds ratio^a of $\tilde{x}'$ compared with $\tilde{x}$ is

$$\omega = e^{\beta_j}.$$

^a<https://morrison-lab.github.io/sds/categorical-tests.html#def-odds-ratio>

i Proof

Proof. By Theorem 3.2 the odds^a at $\tilde{x}'$ are e^{β_j} times the odds at $\tilde{x}$. The odds ratio^b is the odds at $\tilde{x}'$ divided by the odds at $\tilde{x}$. That quotient is e^{β_j} . $\square$

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-odds>

^b<https://morrison-lab.github.io/sds/categorical-tests.html#def-odds-ratio>

Example 3.5 (The odds ratio for body mass index). In the model of Example 2.2, take $\tilde{x}$ to be a man with bmi = 25 and $\tilde{x}'$ a man with bmi = 26.

```
wcgs_odds <- function(bmi) {  
  p <- unname(predict(wcgs_logit, data.frame(bmi = bmi), type = "response"))  
  p / (1 - p)  
}  
c(odds_25 = wcgs_odds(25), odds_26 = wcgs_odds(26), ratio = wcgs_odds(26) / wcgs_odds(25))
```

```
      odds_25    odds_26      ratio  
0.09055613 0.09851816 1.08792371
```

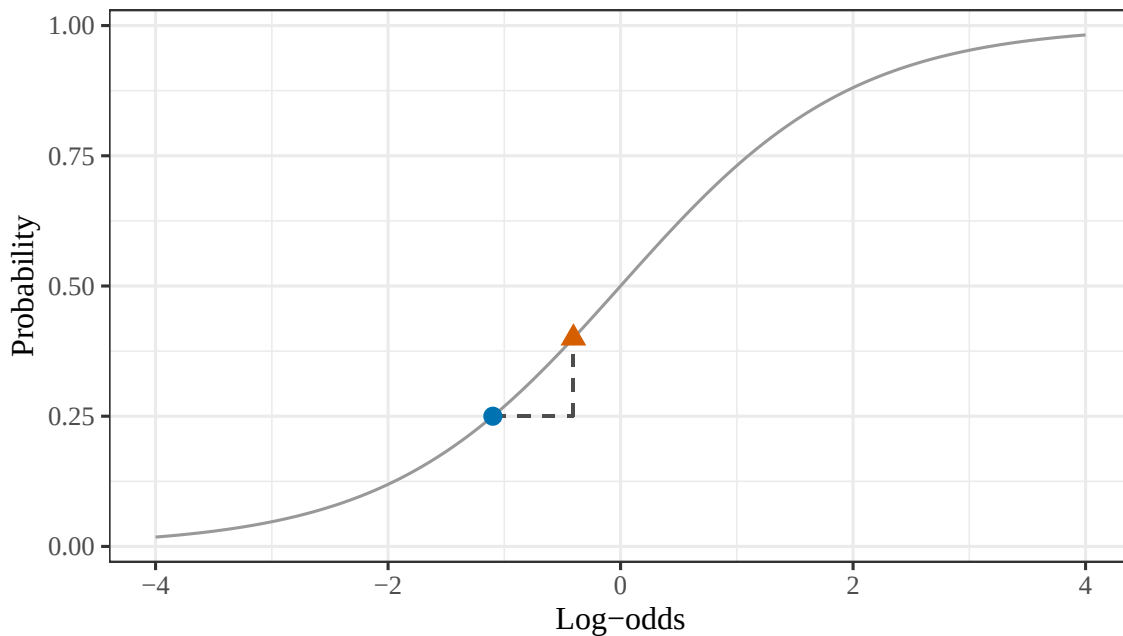
Their odds^a are 0.0906 and 0.0985, so the odds ratio^b is $\omega = 1.088$. This equals $e^{\beta_1} = 1.088$, as Corollary 3.1 states.

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-odds>

^b<https://morrison-lab.github.io/sds/categorical-tests.html#def-odds-ratio>

3.7 R

```
library(ggplot2)
orc_curve <- data.frame(logodds = seq(-4, 4, length.out = 241))
orc_curve$prob <- plogis(orc_curve$logodds)
orc_pts <- data.frame(logodds = c(log(1 / 3), log(1 / 3) + 0.693))
orc_pts$prob <- plogis(orc_pts$logodds)
orc_pts$kind <- factor(c("at the first point", "at the second point"),
                      levels = c("at the first point", "at the second point"))
ggplot(orc_curve, aes(logodds, prob)) +
  geom_line(colour = "grey60") +
  geom_segment(
    x = orc_pts$logodds[1], xend = orc_pts$logodds[2], y = orc_pts$prob[1], yend = orc_pts$prob[1],
    linetype = "dashed", colour = "grey30"
  ) +
  geom_segment(
    x = orc_pts$logodds[2], xend = orc_pts$logodds[2], y = orc_pts$prob[1], yend = orc_pts$prob[2],
    linetype = "dashed", colour = "grey30"
  ) +
  geom_point(data = orc_pts, aes(colour = kind, shape = kind), size = 3) +
  scale_colour_manual(values = c("at the first point" = "#0072B2", "at the second point" = "#D55E00")) +
  scale_shape_manual(values = c("at the first point" = 16, "at the second point" = 17)) +
  labs(x = "Log-odds", y = "Probability", colour = NULL, shape = NULL)
```



● at the first point ▲ at the second point

(a)

3.8 Python

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.special import expit

logodds_orc = np.linspace(-4, 4, 241)
first_orc = np.log(1 / 3)
second_orc = first_orc + 0.693
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.plot(logodds_orc, expit(logodds_orc), color="grey");
ax.hlines(expit(first_orc), first_orc, second_orc, color="#4d4d4d", linestyle="--");
```

Theorem 3.3 (Linear log odds give the sigmoid). *If the log odds^a of $Y = 1$ are η , that is, $\log \frac{p}{1-p} = \eta$ for a probability $p \in (0, 1)$, then*

$$p = \frac{e^\eta}{1 + e^\eta} = \frac{1}{1 + e^{-\eta}} = \text{expit}(\eta).$$

In particular, if the log odds are linear in the features, $\eta = \vec{\beta} \cdot \vec{x}$, then $\Pr[Y = 1 \mid \vec{x}] = \text{expit}(\vec{\beta} \cdot \vec{x})$.

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

i Proof

Proof. This is the solution to Exercise 3.3: see Solution 3.3. □

Example 3.6 (From log-odds back to a probability). For the man with bmi = 25 of Example 2.2, the odds^a of coronary heart disease are $\omega = 0.0906$ (Example 3.5), so the log-odds^b are $\eta = \log \omega = -2.402$.

```
sg_z <- log(wcgs_odds(25))
c(log_odds = sg_z, sigmoid = plogis(sg_z), direct = wcgs_odds(25) / (1 + wcgs_odds(25)))
```

log_odds	sigmoid	direct
-2.40178544	0.08303665	0.08303665

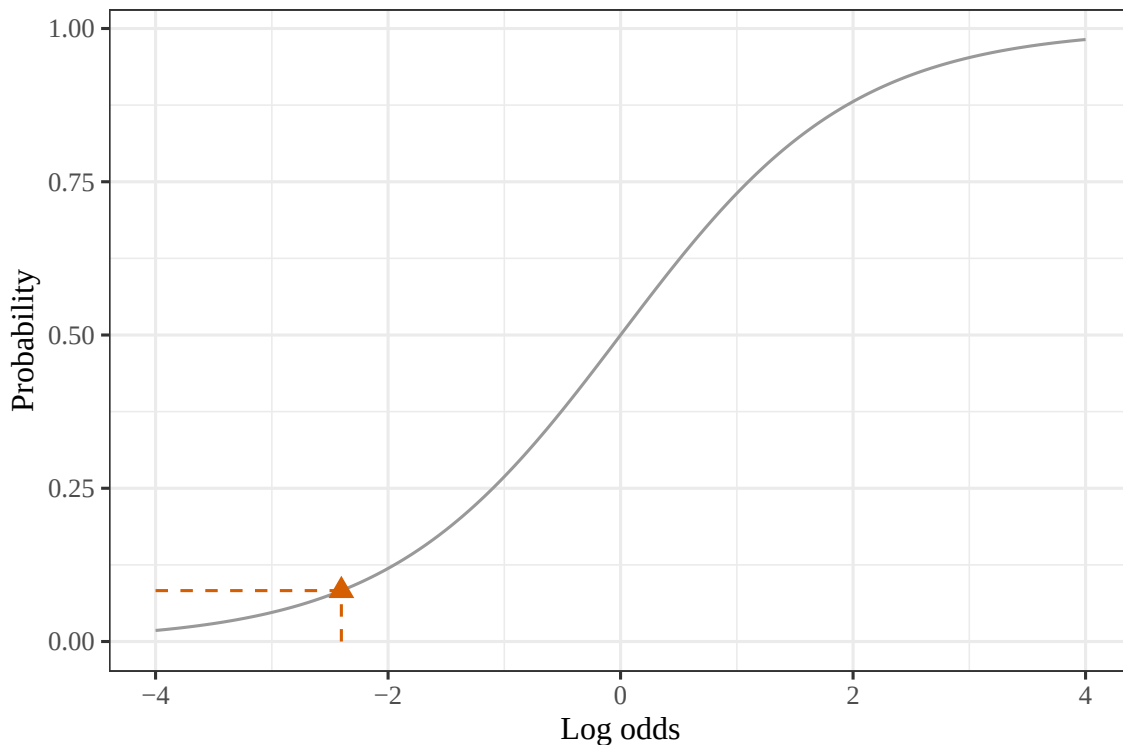
By Theorem 3.3, $\pi = \text{expit}(\eta) = 1/(1 + e^{-\eta}) = 0.083$, which equals the probability $\omega/(1 + \omega)$ that those odds always give.

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-odds>

^b<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

3.9 R

```
library(ggplot2)
sgf_curve <- data.frame(z = seq(-4, 4, length.out = 241))
sgf_curve$p <- plogis(sgf_curve$z)
sgf_pt <- data.frame(z = sgf_curve$z, p = plogis(sgf_curve$z))
ggplot(sgf_curve, aes(z, p)) +
  geom_line(colour = "grey60") +
  geom_segment(
    data = sgf_pt, aes(x = z, xend = z, y = 0, yend = p),
    colour = "#D55E00", linetype = "dashed"
  ) +
  geom_segment(
    data = sgf_pt, aes(x = -4, xend = z, y = p, yend = p),
    colour = "#D55E00", linetype = "dashed"
  ) +
  geom_point(data = sgf_pt, colour = "#D55E00", shape = 17, size = 3) +
  labs(x = "Log odds", y = "Probability")
```



(a)

3.10 Python

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.special import expit

z_sgf = float(r.sg_z)
grid_sgf = np.linspace(-4, 4, 241)
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.plot(grid_sgf, expit(grid_sgf), color="grey");
ax.vlines(z_sgf, 0, expit(z_sgf), color="#D55E00", linestyle="--");
ax.hlines(expit(z_sgf), -4, z_sgf, color="#D55E00", linestyle="--");
ax.plot(z_sgf, expit(z_sgf), color="#D55E00", marker="^", markersize=8, linestyle="");
ax.set_xlabel("Log odds");
ax.set_ylabel("Probability");
plt.show()
```

i Watch

- Hastie and Tibshirani fit logistic regression with several predictors^a [4], video 4.3.
- Starmer explains odds and log odds^b [11], odds ratios^c [12], and what the coefficients mean^d [13].

^a<https://www.youtube.com/watch?v=1uJVE8bkabc>

^b<https://www.youtube.com/watch?v=ARfXDSkQf1Y>

^c<https://www.youtube.com/watch?v=8nm0G-1uJzA>

^d<https://www.youtube.com/watch?v=vN5cNN2-HWE>

i Read

GeeksforGeeks explains log odds^a [14].

^a<https://www.geeksforgeeks.org/machine-learning/log-odds/>

4 Where the boundary is

Turning a probability into a decision means choosing a threshold. The set of points where the model is exactly at the threshold is its **decision boundary** (Definition 4.1), and for **logistic regression** it is a hyperplane.

Definition 4.1 (Decision boundary). The **decision boundary** of a classifier is the set of feature vectors where it is exactly undecided between two predictions. For a probabilistic classifier with threshold θ , it is the set where the predicted probability equals θ ; for logistic regression at $\theta = 1/2$, it is $\{\tilde{x} : \beta_0 + \vec{\beta} \cdot \tilde{x} = 0\}$.

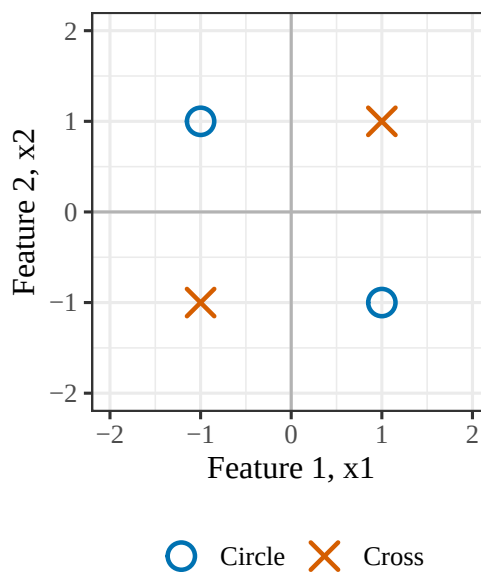
Example 4.1 (A boundary in two features). Let $\beta_0 = -1$, $\beta_1 = 2$, and $\beta_2 = 1$. The boundary $-1 + 2x_1 + x_2 = 0$ is the line $x_2 = 1 - 2x_1$. The point $(1, 1)$ has score 2 and is on the side predicted 1, while $(0, 0)$ has score -1 and is on the other side.

4.1 R

```
library(ggplot2)

xor_df <- expand.grid(x1 = c(-1, 1), x2 = c(-1, 1))
xor_df$class <- ifelse(xor_df$x1 * xor_df$x2 > 0, "Cross", "Circle")

ggplot(xor_df) +
  aes(x1, x2, shape = class, colour = class) +
  geom_hline(yintercept = 0, colour = "grey70") +
  geom_vline(xintercept = 0, colour = "grey70") +
  geom_point(size = 4, stroke = 1.2) +
  scale_shape_manual(values = c(Circle = 1, Cross = 4)) +
  scale_colour_manual(values = c(Circle = "#0072B2", Cross = "#D55E00")) +
  coord_fixed(xlim = c(-2, 2), ylim = c(-2, 2)) +
  labs(x = "Feature 1, x1", y = "Feature 2, x2", shape = NULL, colour = NULL)
```



(a)

4.2 Python

```
import itertools

import matplotlib.pyplot as plt
import numpy as np

corners = np.array(list(itertools.product([-1, 1], repeat=2)))
is_cross = corners[:, 0] * corners[:, 1] > 0

fig, ax = plt.subplots(figsize=(3.8, 3.2), layout="constrained")
ax.axhline(0, color="lightgrey");
ax.axvline(0, color="lightgrey");
ax.scatter(*corners[~is_cross].T, marker="o", facecolors="none", s=70,
          edgecolors="#0072B2", linewidths=1.5, label="Circle");
ax.scatter(*corners[is_cross].T, marker="x", s=70, color="#D55E00",
          linewidths=1.5, label="Cross");
ax.set_xlim(-2, 2);
ax.set_ylim(-2, 2);
ax.set_aspect("equal");
ax.set_xlabel("Feature 1, x1");
ax.set_ylabel("Feature 2, x2");
fig.legend(loc="outside lower center", ncol=2, frameon=False);
plt.show()
```

Exercise 4.1 (The boundary is a hyperplane). A classifier must return a class label rather than a probability, so we apply a threshold to the predicted probability. Using the threshold $1/2$, predict $\hat{y} = 1$ when $p(\tilde{x}) = \text{expit}(\beta_0 + \vec{\beta} \cdot \tilde{x}) > 1/2$. Show that this rule is equivalent to a condition that is *linear* in $\tilde{x}$, and state that condition.

Solution

Solution 4.1. expit is strictly increasing and $\text{expit}(0) = 1/2$, so $\text{expit}(\eta) > 1/2 \iff \eta > 0$. Hence

$$\hat{y} = 1 \iff \beta_0 + \vec{\beta} \cdot \tilde{x} > 0$$

which is linear in $\tilde{x}$. The sigmoid has vanished entirely: it decides *how confident* the model is, and contributes nothing to *which* class is predicted.

Exercise 4.2 (The boundary is a hyperplane (part 2)). Use the setup of Exercise 4.1. The set of $\tilde{x}$ where the predicted probability equals the threshold is the [decision boundary](#). Identify this geometric set, and give its normal vector^a.

^a<https://morrison-lab.github.io/mds/linear-algebra-matrices.html#def-normal-vector>

Solution

Solution 4.2. A hyperplane: the set $\{\tilde{x} : \beta_0 + \vec{\beta} \cdot \tilde{x} = 0\}$. In two dimensions it is a line, in three a plane, in p dimensions a flat $(p-1)$ -dimensional sheet.

Its normal vector^a is $\vec{\beta}$. That is the geometric content of Exercise 2.11: the direction of $\vec{\beta}$ is the direction perpendicular to the boundary, which is why rescaling $\vec{\beta}$ leaves the boundary alone.

^a<https://morrison-lab.github.io/mds/linear-algebra-matrices.html#def-normal-vector>

Exercise 4.3 (The boundary is a hyperplane (part 3)). Use the setup of Exercise 4.1. Suppose instead the cost of a false negative is nine times the cost of a false positive, so you threshold at $p > 0.1$ rather than $p > 0.5$. Give the new boundary, and say precisely how it differs from the old one.

Solution

Solution 4.3. Threshold at $p > t$ rather than $p > 1/2$:

$$\text{expit}(\eta) > t \iff \eta > \text{expit}^{-1}(t) = \log \frac{t}{1-t}$$

so the boundary is $\beta_0 + \vec{\beta} \cdot \tilde{x} = \log \frac{t}{1-t}$.

At $t = 0.1$ that is $\log(1/9) = -2.197$. The boundary is the *same hyperplane translated*: same normal vector^a, same orientation, shifted along $\vec{\beta}$ by $2.197/\|\vec{\beta}\|$ in the direction that enlarges the region predicted 1.

Two consequences follow. First, changing the threshold does not require refitting the model: one set of fitted probabilities supports every threshold, so two users with different misclassification costs can apply different thresholds to the same fitted model. Second, the cost ratio enters the boundary equation only through the constant^b term $\log \frac{t}{1-t}$, so changing the threshold shifts the intercept^c of the boundary without changing $\vec{\beta}$.

^a<https://morrison-lab.github.io/mds/linear-algebra-matrices.html#def-normal-vector>

^b<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

^c<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

Exercise 4.4 (The boundary is a hyperplane (part 4)). Use the setup of Exercise 4.1.

A colleague says **logistic regression** is “a linear model, so it can only draw straight boundaries, which makes it useless for the data in Figure 16”. The first clause is true and the conclusion does not follow. Explain why, and say what you would do to the *features* to separate those points.

Solution

Solution 4.4. The boundary is linear in the **features**, not in the raw measurements. Those coincide only when no nonlinear features are constructed from the measurements.

For Figure 16, the classes are separated by the sign of x_1x_2 , so adding a third feature $x_3 = x_1x_2$ makes the data linearly separable in the three-dimensional feature space — the plane $x_3 = 0$ separates the classes — while its intersection with the (x_1, x_2) plane is the pair of coordinate axes, which no single line in two dimensions can represent.

This is the same construction linear regression uses when fitting a polynomial^a, and that neural networks automate by *learning* feature transformations from data.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-polynomial>

Exercise 4.5 (The boundary is a hyperplane (part 5)). Use the setup of Exercise 4.1.

Logistic regression and linear regression are both linear in $\vec{\beta} \cdot x$. State what is linear in each, and say which of the two has a linear relationship between features and the quantity actually reported.

Solution

Solution 4.5. In linear regression the *prediction itself* is linear in the features: $\hat{y} = \vec{\beta} \cdot x$, and that is the number reported.

In **logistic regression** the **log odds**^a are linear in the features (Exercise 3.1), and the number reported is expit of that — which is not linear in anything.

So only linear regression has a linear relationship between features and the reported quantity. Calling logistic regression a linear model is a statement about its *boundary* and its log odds, and readers often misread it as a claim about its predictions.

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

Theorem 4.1 (The logistic regression boundary is a hyperplane). *Let β_0 be the intercept and $\vec{\beta} \neq \vec{0}$ the vector of slopes of a logistic regression model. At threshold $\theta \in (0, 1)$, the decision boundary of the model is the hyperplane*

$$\beta_0 + \vec{\beta} \cdot x = \log \frac{\theta}{1 - \theta}$$

with normal vector $\vec{\beta}$. Changing the threshold translates the hyperplane and does not rotate it.

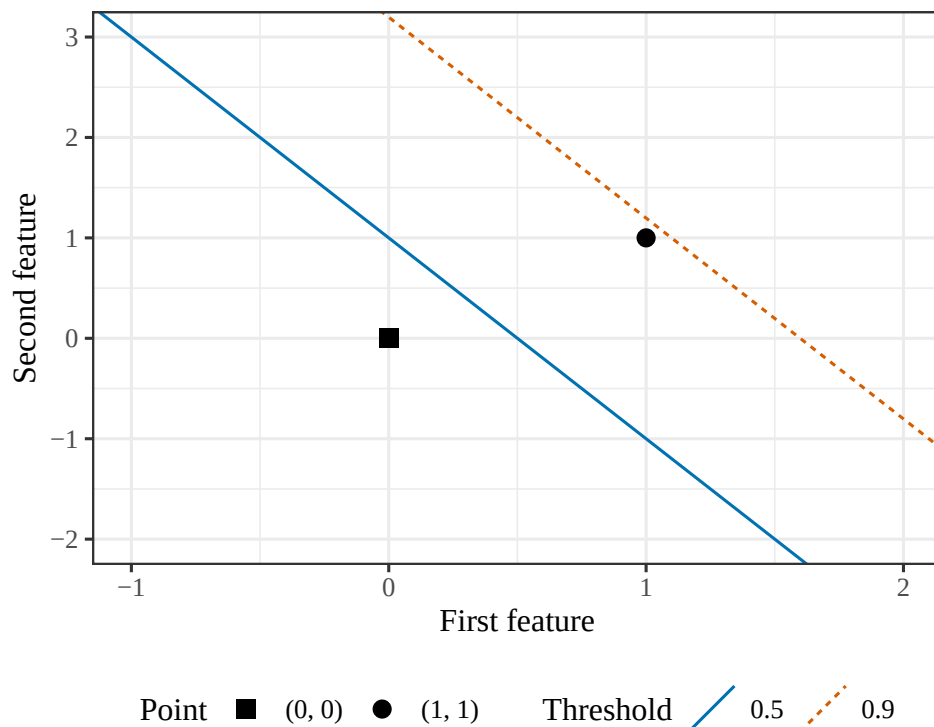
i Proof

Proof. By Solution 4.1 to Solution 4.3. □

Example 4.2 (Moving the threshold). For the model of Example 4.1, a threshold of $\theta = 0.9$ puts the boundary at $-1 + 2x_1 + x_2 = \log(0.9/0.1) = 2.197$. That is the same line, shifted by $2.197/\|\vec{\beta}\| = 0.983$ units along the normal vector $\vec{\beta} = (2, 1)$, in the direction that shrinks the region predicted 1.

4.3 R

```
library(ggplot2)
hb_coef <- c(-1, 2, 1)
hb_lines <- data.frame(
  threshold = c("0.5", "0.9"),
  intercept = (log(c(0.5, 0.9) / (1 - c(0.5, 0.9))) - hb_coef[1]) / hb_coef[3],
  slope = -hb_coef[2] / hb_coef[3]
)
hb_points <- data.frame(x1 = c(1, 0), x2 = c(1, 0), point = c("(1, 1)", "(0, 0)"))
ggplot() +
  geom_abline(
    data = hb_lines,
    aes(intercept = intercept, slope = slope, colour = threshold, linetype = threshold)
  ) +
  geom_point(data = hb_points, aes(x1, x2, shape = point), size = 3) +
  scale_colour_manual(values = c("0.5" = "#0072B2", "0.9" = "#D55E00")) +
  scale_shape_manual(values = c("(1, 1)" = 16, "(0, 0)" = 15)) +
  coord_cartesian(xlim = c(-1, 2), ylim = c(-2, 3)) +
  labs(
    x = "First feature", y = "Second feature",
    colour = "Threshold", linetype = "Threshold", shape = "Point"
  )
)
```



(a)

4.4 Python

```
import matplotlib.pyplot as plt
import numpy as np

coef_hb = np.array([-1, 2, 1])
x1_hb = np.array([-1, 2])
fig, ax = plt.subplots(figsize=(5, 4), layout="constrained")
for threshold, colour, style in [(0.5, "#0072B2", "-"), (0.9, "#D55E00", "--")]:
    x2_hb = (np.log(threshold / (1 - threshold)) - coef_hb[0] - coef_hb[1] * x1_hb) / coef_hb[2]
    ax.plot(x1_hb, x2_hb, color=colour, linestyle=style, label=f"threshold {threshold}");
ax.plot(1, 1, color="black", marker="o", markersize=8, linestyle="", label="(1, 1)");
```

5 Fitting: where the cost comes from

The form of the model is specified, and the parameter vector $\vec{\beta}$ remains to be estimated²⁶. Logistic regression is a probability model, so maximum likelihood²⁷ provides a criterion for choosing $\vec{\beta}$: the value that makes the observed labels most probable.

Maximum likelihood²⁸ determines the loss function directly. Exercise 5.1, Exercise 5.2, and Exercise 5.3 derive the resulting log loss, and Exercise 5.4 and Exercise 5.5 show why squared-error loss on the probabilities is unsuitable.

Exercise 5.1 (Probability of a binary outcome in one formula). Exercise 3.1 specified the *model*: $\Pr[Y = 1 \mid x] = \text{expit}(\vec{\beta} \cdot x)$. It did not specify how to estimate^a $\vec{\beta}$. Let $\pi_i = \text{expit}(\vec{\beta} \cdot \tilde{x}_i)$ be the probability that the model assigns to observation i having label 1. Express $\Pr[Y_i = y_i \mid \tilde{x}_i]$ as a single formula that holds for both $y_i = 0$ and $y_i = 1$, and check it at each value.

^a<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

Solution

Solution 5.1.

$$\Pr[Y_i = y_i \mid \tilde{x}_i] = \pi_i^{y_i} (1 - \pi_i)^{1-y_i}$$

At $y_i = 1$, the second factor is $(1 - \pi_i)^0 = 1$ and the expression^a reads π_i . At $y_i = 0$, the first factor is $\pi_i^0 = 1$ and the expression reads $1 - \pi_i$. Each is the model's assertion, so the single expression covers both cases.

The exponents select the factor corresponding to the observed outcome: each exponent is either 0 or 1, so one factor equals 1 and the other remains.

^a<https://morrison-lab.github.io/mds/expressions-algorithms.html#def-expression>

Lemma 5.1 (Probability mass of one Bernoulli outcome). If $Y_i \mid \tilde{x}_i \sim \text{Bernoulli}(\pi_i)$ and $y_i \in \{0, 1\}$, then

$$\Pr(Y_i = y_i \mid \tilde{x}_i) = \pi_i^{y_i} (1 - \pi_i)^{1-y_i}.$$

i Proof

Proof. By Solution 5.1. □

Example 5.1 (The probability mass of one outcome). Take $\pi_i = 0.8$. Lemma 5.1 gives the probabilities of the two outcomes, which match R's `dbinom`:

```
p_bpmf <- 0.8
c(p_bpmf^1 * (1 - p_bpmf)^0, p_bpmf^0 * (1 - p_bpmf)^1)
[1] 0.8 0.2

dbinom(c(1, 0), size = 1, prob = p_bpmf)
[1] 0.8 0.2
```

²⁶<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

²⁷<https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle>

²⁸<https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle>

Exercise 5.2 (Likelihood of the dataset). Using the single-observation probability from Exercise 5.1, express the likelihood^a $L(\vec{\beta})$ of the full dataset in terms of the $\pi_i = \text{expit}(\vec{\beta} \cdot \tilde{x}_i)$ and labels y_i , and state the assumption required.

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik>

Solution

Solution 5.2.

$$L(\vec{\beta}) = \prod_{i=1}^n \pi_i^{y_i} (1 - \pi_i)^{1-y_i}, \quad \pi_i = \text{expit}(\vec{\beta} \cdot \tilde{x}_i)$$

Multiplying requires that the observations be conditionally independent^a given their features. Conditional independence^b is weaker than marginal independence^c of the Y_i — two patients from the same household may have correlated outcomes — but it rules out datasets in which y_1 carries information about y_2 beyond what $\tilde{x}_2$ provides. Repeated measurements on the same subject, or students within a classroom, violate conditional independence and require a model with a shared random effect or cluster term.

^a<https://morrison-lab.github.io/pds/independence.html#def-cind>

^b<https://morrison-lab.github.io/pds/independence.html#def-cind>

^c<https://morrison-lab.github.io/pds/independence.html#def-indpt>

Lemma 5.2 (Likelihood of conditionally independent Bernoulli outcomes). If $Y_1, \dots, Y_n$ are conditionally independent^a given $\tilde{x}_1, \dots, \tilde{x}_n$ with $Y_i \mid \tilde{x}_i \sim \text{Bernoulli}(\pi_i)$, where π_i is determined by the coefficients $\vec{\beta}$ and the observed outcomes are $y_i \in \{0, 1\}$, the likelihood^b is

$$L(\vec{\beta}) = \prod_{i=1}^n \pi_i^{y_i} (1 - \pi_i)^{1-y_i}.$$

^a<https://morrison-lab.github.io/pds/independence.html#def-cind>

^b<https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik>

i Proof

Proof. By Solution 5.2. □

Example 5.2 (The likelihood of three outcomes). Take the outcomes $(1, 0, 1)$ with $(\pi_1, \pi_2, \pi_3) = (0.9, 0.2, 0.7)$. Lemma 5.2 gives the product of the three probability masses:

```
y_bl <- c(1, 0, 1)
p_bl <- c(0.9, 0.2, 0.7)
prod(p_bl^y_bl * (1 - p_bl)^(1 - y_bl))
[1] 0.504
```

Exercise 5.3 (Log-likelihood of the dataset). Take the logarithm of the likelihood^a $L(\vec{\beta})$ from Exercise 5.2 and simplify. Give two reasons to work with the log-likelihood^b $\log L(\vec{\beta})$ rather than $L(\vec{\beta})$.

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik>

^b<https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik>

Solution

Solution 5.3.

$$\begin{aligned}\ell(\vec{\beta}) &= \log L(\vec{\beta}) && \text{(definition of log-likelihood)} \\ &= \log \left(\prod_{i=1}^n \pi_i^{y_i} (1 - \pi_i)^{1-y_i} \right) && \text{(by @sol-bernoulli-likelihood)} \\ &= \sum_{i=1}^n \log(\pi_i^{y_i} (1 - \pi_i)^{1-y_i}) && \text{(log of a product)} \\ &= \sum_{i=1}^n [\log(\pi_i^{y_i}) + \log((1 - \pi_i)^{1-y_i})] && \text{(log of a product)} \\ &= \sum_{i=1}^n [y_i \log \pi_i + (1 - y_i) \log(1 - \pi_i)] && \text{(log of a power)}\end{aligned}$$

Two reasons.

Arithmetic. $L(\vec{\beta})$ is a product of n numbers each below 1, so for n in the thousands it underflows to 0 in floating point and every candidate $\vec{\beta}$ evaluates to 0. The logarithm turns the product into a sum of terms of moderate magnitude.

Optimization. $\log$ is strictly increasing, so it preserves the maximizer, and a sum differentiates term by term: the gradient of $\ell(\vec{\beta})$ is the sum of n per-observation gradients, which simplifies Exercise 6.1 and enables stochastic gradient descent.

Maximizing $\ell(\vec{\beta})$ is equivalent to minimizing $-\ell(\vec{\beta})$, which is called the [log loss](#) or [cross-entropy loss](#). The two names describe the same function, and no additional assumption is introduced: the log loss is the plausible loss under conditionally independent^a Bernoulli outcomes (Theorem 5.1).

^a<https://morrison-lab.github.io/pds/independence.html#def-cind>

Lemma 5.3 (Log-likelihood of conditionally independent Bernoulli outcomes). *Under the assumptions of Lemma 5.2, the log-likelihood^a is*

$$\ell(\vec{\beta}) = \sum_{i=1}^n [y_i \log \pi_i + (1 - y_i) \log(1 - \pi_i)].$$

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik>

i Proof

Proof. By Solution 5.3. □

Example 5.3 (The log-likelihood of three outcomes). With the outcomes and probabilities of Example 5.2, Lemma 5.3 gives the log-likelihood^a as a sum, which equals the log of the likelihood^b:

```
y_b11 <- c(1, 0, 1)
p_b11 <- c(0.9, 0.2, 0.7)
sum(y_b11 * log(p_b11) + (1 - y_b11) * log(1 - p_b11))

[1] -0.685179

log(prod(p_b11^y_b11 * (1 - p_b11)^(1 - y_b11)))

[1] -0.685179
```

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik>

^b<https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik>

Exercise 5.4 (Negative log-likelihood on a positive observation). Consider one observation with label $y_i = 1$ in the log-likelihood^a of Exercise 5.3. Give its contribution to $-\log L(\vec{\beta})$ at $\pi_i = 0.9, 0.5, 0.1$, and 0.01 , to three significant figures, and state what happens as $\pi_i \rightarrow 0$.

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik>

Solution

Solution 5.4. With $y_i = 1$, observation i 's contribution to $-\log L(\vec{\beta})$ is $-\log \pi_i$:

π_i	$-\log \pi_i$
0.9	0.105
0.5	0.693
0.1	2.30
0.01	4.61

As $\pi_i \rightarrow 0$, the loss grows without bound. Assigning probability 0 to an outcome that occurs gives an infinite loss; the loss is unbounded above even though the predicted probability is in $(0, 1)$.

Exercise 5.5 (Why squared error on the probabilities is unsuitable). Under squared-error loss, the positive observation of Exercise 5.4 contributes $(\pi_i - y_i)^2$, which never exceeds 1 regardless of how small π_i is. Explain what that upper bound allows a fitted model to do without incurring a large loss, and state which of the two requirements in Exercise 1.4 the [log loss](#) satisfies.

Solution

Solution 5.5. Because the squared-error loss on a single observation is at most 1, the loss on one misclassified observation is bounded above by 1 even as $\pi_i \rightarrow 0$. If allowing $\pi_i \rightarrow 0$ on one observation with $y_i = 1$ reduces the sum of squared errors across the remaining observations by more than 1, the total squared error decreases. Under the [log loss](#), $-\log \pi_i \rightarrow \infty$ as $\pi_i \rightarrow 0$ (Solution 5.4), so no finite reduction in loss on the remaining observations can offset assigning $\pi_i \rightarrow 0$ to an observation with $y_i = 1$.

This unbounded penalty satisfies the second requirement of Exercise 1.4. The loss saturates on the correct side — $-\log \pi_i$ approaches 0 with vanishing derivative as $\pi_i \rightarrow 1$, so increasing the margin on an already well-classified observation reduces the loss by very little — and grows without bound on the wrong side.

One caveat: we should not overstate the argument. Squared error on a probability is the *Brier score*, and it is a *proper* scoring rule: its expected value^a is minimized by reporting the true probability, exactly as the log loss's is. So the case against squared error here is not that it rewards dishonesty. Squared error has three faults instead: the boundedness above; the loss of the maximum-likelihood^b reading, which supplies standard errors^c and likelihood^d ratio tests for the fitted coefficients; and the vanishing gradient that Exercise 6.1 exhibits. Squared error on π_i is neither a plausible loss for Bernoulli outcomes nor a friendly loss when paired with the logistic function^e, whereas the log loss is both.

^a<https://morrison-lab.github.io/pds/expectation.html#def-expectation>

^b<https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle>

^c<https://morrison-lab.github.io/sds/estimation.html#def-SE>

^d<https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik>

^e<https://morrison-lab.github.io/mds/functions.html#def-logistic-function>

Definition 5.1 (Cross-entropy loss (also called log loss)). For a label $y_i \in \{0, 1\}$ and a predicted probability $\pi_i = \Pr(Y_i = 1 \mid \tilde{x}_i)$, the per-observation **cross-entropy loss**, also called the **log loss**, is

$$\begin{aligned} L(y_i, \pi_i) &= C_i \\ &= -y_i \log \pi_i - (1 - y_i) \log(1 - \pi_i). \end{aligned}$$

Summed over the n observations, the total log loss is the negative log-likelihood^a of conditionally independent^b Bernoulli labels (Solution 5.3),

$$\begin{aligned} C(\vec{\beta}) &= \sum_{i=1}^n C_i \\ &= \sum_{i=1}^n [-y_i \log \pi_i - (1 - y_i) \log(1 - \pi_i)] \\ &= -\ell(\vec{\beta}), \end{aligned}$$

and its average $\hat{R}(\vec{\beta}) = \frac{1}{n}C(\vec{\beta})$ is the empirical risk. Different sources apply the name *log loss* or *cross-entropy loss* to the per-observation loss function $L(y_i, \pi_i)$, to the sum $C(\vec{\beta})$, or to the average $\hat{R}(\vec{\beta})$ [15].

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik>

^b<https://morrison-lab.github.io/pds/independence.html#def-cind>

Example 5.4 (Log loss and squared error on a positive case). For an observation with $y_i = 1$, evaluate the [log loss](#) $C_i = -\log \pi_i$ and the squared error $(\pi_i - y_i)^2$ across $\pi_i = 0.8, 0.6, 0.2, 0.01$:

5.1 R

```
prob_ll <- c(0.8, 0.6, 0.2, 0.01)
log_loss_ll <- -log(prob_ll)
sq_err_ll <- (prob_ll - 1)^2
round(
  data.frame(prob = prob_ll, log_loss = log_loss_ll, squared_error = sq_err_ll),
  3
)

# A tibble: 4 x 3
  prob log_loss squared_error
  <dbl>   <dbl>         <dbl>
1  0.8     0.223           0.04
2  0.6     0.511           0.16
3  0.2     1.61            0.64
4  0.01    4.61            0.98
```

5.2 Python

```
import numpy as np
import pandas as pd

prob_ll = np.array([0.8, 0.6, 0.2, 0.01])
print(pd.DataFrame({
    "prob": prob_ll,
    "log_loss": -np.log(prob_ll),
    "squared_error": (prob_ll - 1.0) ** 2,
}).round(3))

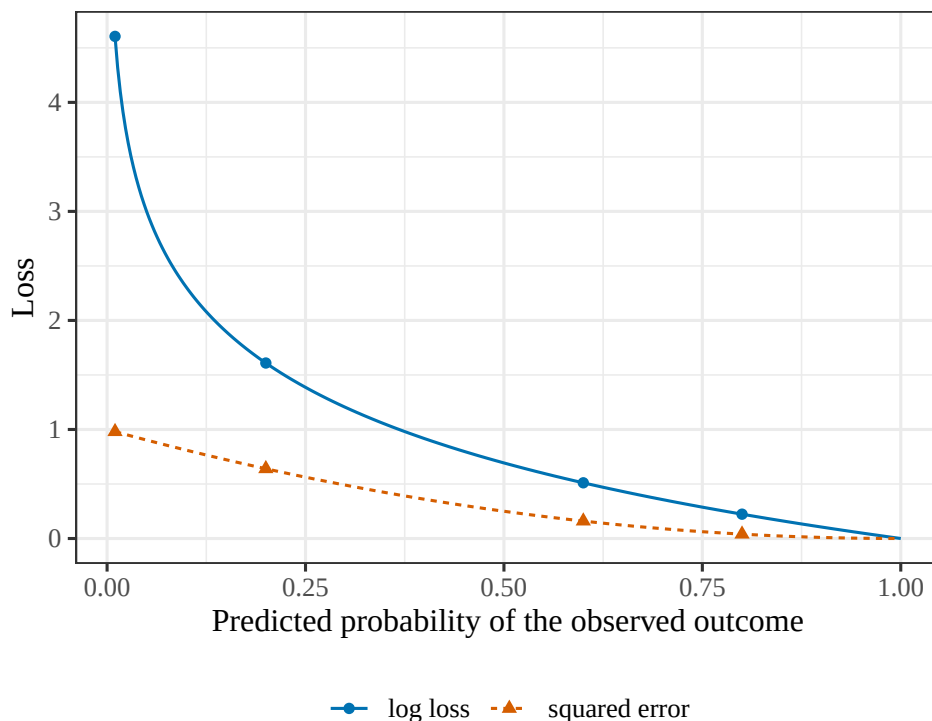
   prob  log_loss  squared_error
0  0.80    0.223           0.04
1  0.60    0.511           0.16
```

2	0.20	1.609	0.64
3	0.01	4.605	0.98

At $\pi_i = 0.8$, both losses are small (0.223 and 0.04). At $\pi_i = 0.01$, squared error reaches 0.98, just below its ceiling of 1, while the [log loss](#) reaches 4.61 and grows without bound as $\pi_i \rightarrow 0$.

5.3 R

```
library(ggplot2)
ll_curve <- data.frame(prob = seq(0.01, 1, length.out = 300))
ll_long <- data.frame(
  prob = rep(ll_curve$prob, 2),
  loss = c(-log(ll_curve$prob), (ll_curve$prob - 1)^2),
  kind = rep(c("log loss", "squared error"), each = nrow(ll_curve))
)
ll_points <- data.frame(
  prob = rep(prob_ll, 2),
  loss = c(log_loss_ll, sq_err_ll),
  kind = rep(c("log loss", "squared error"), each = length(prob_ll))
)
ggplot(ll_long, aes(prob, loss, colour = kind, linetype = kind)) +
  geom_line() +
  geom_point(aes(shape = kind), data = ll_points) +
  scale_colour_manual(values = c("log loss" = "#0072B2", "squared error" = "#D55E00")) +
  labs(x = "Predicted probability of the observed outcome", y = "Loss", colour = NULL, linetype = NULL,
```



(a)

5.4 Python

```
import matplotlib.pyplot as plt
import numpy as np

p_ll = np.linspace(0.01, 1, 300)
fig, ax = plt.subplots(figsize=(5, 4), layout="constrained")
ax.plot(p_ll, -np.log(p_ll), color="#0072B2", linestyle="-", label="log loss");
ax.plot(p_ll, (p_ll - 1) ** 2, color="#D55E00", linestyle="--", label="squared error");
ax.plot(prob_ll, -np.log(prob_ll), color="#0072B2", marker="o", linestyle="");
ax.plot(prob_ll, (prob_ll - 1) ** 2, color="#D55E00", marker="^", linestyle="");
ax.set_xlabel("Predicted probability of the observed outcome");
ax.set_ylabel("Loss");
fig.legend(loc="outside lower center", ncol=2);
plt.show()
```

Theorem 5.1 (Minimizing the log loss is maximum likelihood estimation). Suppose the outcomes $Y_1, \dots, Y_n \in \{0, 1\}$ are conditionally independent^a given $\tilde{x}_1, \dots, \tilde{x}_n$, with

$$Y_i \mid \tilde{x}_i \sim \text{Bernoulli}(\pi_i), \quad \pi_i = \text{expit}(\vec{\beta} \cdot \tilde{x}_i).$$

Then minimizing the total **log loss** $C(\vec{\beta})$ (or the empirical risk $\hat{R}(\vec{\beta}) = \frac{1}{n}C(\vec{\beta})$) with respect to $\vec{\beta}$ yields the maximum likelihood^b estimate^c $\hat{\vec{\beta}}$. The log loss is therefore the plausible loss for conditionally independent Bernoulli outcomes.

^a<https://morrison-lab.github.io/pds/independence.html#def-indpt>

^b<https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle>

^c<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

i Proof

Proof. By assumption, the outcomes are conditionally independent^a with $Y_i \mid \tilde{x}_i \sim \text{Bernoulli}(\pi_i)$, so Lemma 5.2 gives the likelihood^b and Lemma 5.3 gives the log-likelihood^c $\ell(\vec{\beta})$, which is $-C(\vec{\beta})$ by the definition of the **log loss**. Because log is strictly increasing and $n > 0$, maximizing the likelihood^d is equivalent to maximizing $\ell(\vec{\beta})$, which is equivalent to minimizing $C(\vec{\beta})$ or $\frac{1}{n}C(\vec{\beta})$. $\square$

^a<https://morrison-lab.github.io/pds/independence.html#def-cind>

^b<https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik>

^c<https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik>

^d<https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik>

Example 5.5 (Minimizing the log loss on the WCGS data recovers the maximum likelihood fit). Take the WCGS data of Exercise 1.1 (data/wcgs.csv^a), whose column `dibpat` records each man's behavior pattern, type A or type B. Subset to the men of type A (`dibpat` is "A"), so that a single curve with two coefficients describes them. Model coronary heart disease (`chd`) on age at enrollment (`age`) with $\pi_i = \text{expit}(\beta_0 + \beta_1 \text{age}_i)$. Minimize the average **log loss** (empirical risk) $\hat{R}(\vec{\beta}) = \frac{1}{n}C(\vec{\beta})$ numerically from $\vec{\beta} = \vec{0}$, and compare with the maximum likelihood^b fit:

5.5 R

```
wcgs_mle <- subset(read.csv("data/wcgs.csv"), dibpat == "A")
y_wcgs <- wcgs_mle$chd
x_wcgs <- cbind(1, wcgs_mle$age)

mean_log_loss <- function(b) {
  p <- plogis(drop(x_wcgs %*% b))
  mean(-y_wcgs * log(p) - (1 - y_wcgs) * log(1 - p))
}

loss_fit <- optim(c(0, 0), mean_log_loss, control = list(reltol = 1e-14))
glm_fit <- coef(glm(chd ~ age, family = binomial, data = wcgs_mle))
round(rbind(log_loss_minimizer = loss_fit$par, mle = unname(glm_fit)), 6)

           [,1]      [,2]
log_loss_minimizer -5.498861 0.071907
mle                -5.498862 0.071907
```

5.6 Python

```

import numpy as np
import pandas as pd
import statsmodels.api as sm
from scipy.optimize import minimize
from scipy.special import expit

wcgs_all = pd.read_csv("data/wcgs.csv")
wcgs_mle = wcgs_all[wcgs_all["dibpat"] == "A"]
y_wcgs = wcgs_mle["chd"].to_numpy(float)
x_wcgs = sm.add_constant(wcgs_mle["age"].to_numpy(float))

def mean_log_loss(b):
    p = expit(x_wcgs @ b)
    return np.mean(-y_wcgs * np.log(p) - (1.0 - y_wcgs) * np.log(1.0 - p))

loss_fit = minimize(
    mean_log_loss,
    x0=np.zeros(2),
    method="Nelder-Mead",
    options={"xatol": 1e-10, "fatol": 1e-14},
)
glm_fit = sm.GLM(y_wcgs, x_wcgs, family=sm.families.Binomial()).fit().params
print(pd.DataFrame({"intercept": [loss_fit.x[0], glm_fit[0]],
                    "age": [loss_fit.x[1], glm_fit[1]]},
                    index=["log_loss_minimizer", "mle"]).round(6))

```

	intercept	age
log_loss_minimizer	-5.498862	0.071907
mle	-5.498862	0.071907

The numerical minimizer of $\hat{R}(\vec{\beta})$ and the maximum likelihood^c estimate^d agree to within 3.4e-07, as Theorem 5.1 states.

^a<https://morrison-lab.github.io/lds/chapters/data/wcgs.csv>

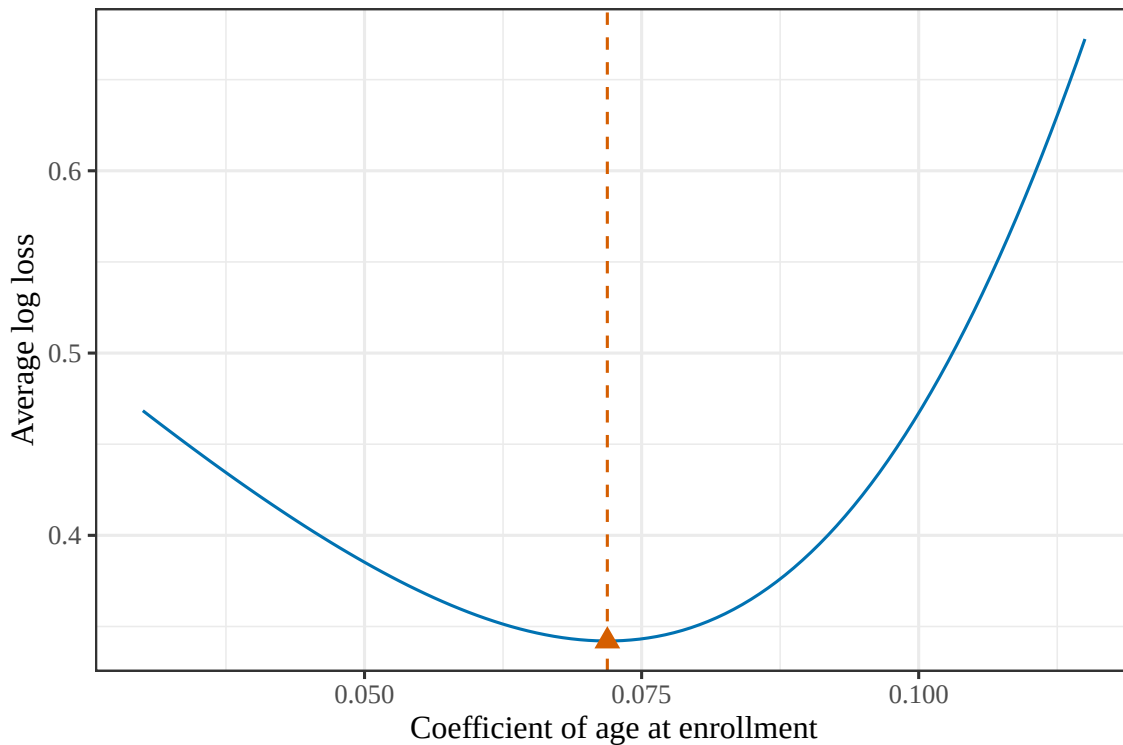
^b<https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle>

^c<https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle>

^d<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

5.7 R

```
library(ggplot2)
llm_slopes <- seq(0.03, 0.115, length.out = 111)
llm_curve <- data.frame(
  slope = llm_slopes,
  loss = vapply(llm_slopes, \(s) mean_log_loss(c(glm_fit[[1]], s)), numeric(1))
)
llm_pt <- data.frame(slope = loss_fit$par[2], loss = mean_log_loss(loss_fit$par))
ggplot(llm_curve, aes(slope, loss)) +
  geom_line(colour = "#0072B2") +
  geom_vline(xintercept = glm_fit[[2]], linetype = "dashed", colour = "#D55E00") +
  geom_point(data = llm_pt, colour = "#D55E00", shape = 17, size = 3) +
  labs(x = "Coefficient of age at enrollment", y = "Average log loss")
```

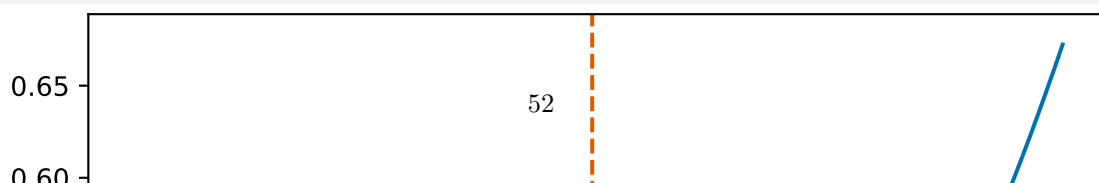


(a)

5.8 Python

```
import matplotlib.pyplot as plt
import numpy as np

slopes_llm = np.linspace(0.03, 0.115, 111)
loss_llm = [mean_log_loss(np.array([glm_fit[0], s])) for s in slopes_llm]
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.plot(slopes_llm, loss_llm, color="#0072B2");
ax.axvline(glm_fit[1], color="#D55E00", linestyle="--");
ax.plot(loss_fit.x[1], mean_log_loss(loss_fit.x), color="#D55E00", marker="^", markersize=8,
        linestyle="");
ax.set_xlabel("Coefficient of age at enrollment");
ax.set_ylabel("Average log loss");
plt.show()
```



Exercise 5.6 (The same loss, written two ways). Exercise 5.3 wrote the per-observation cost with labels $y \in \{0, 1\}$:

$$C(\eta, y) = -y \log \text{expit}(\eta) - (1 - y) \log(1 - \text{expit}(\eta)), \quad \eta = \vec{\beta} \cdot \tilde{x}$$

Much of the literature instead uses $\tilde{y} \in \{-1, +1\}$, via $\tilde{y} = 2y - 1$. Show that $C(\eta, y) = \log(1 + e^{-\tilde{y}\eta})$.

Solution

Solution 5.6. We check the two cases $y = 1$ and $y = 0$ separately. If $y = 1$, then $\tilde{y} = +1$ and the second term of C vanishes:

$$\begin{aligned} C &= -\log \text{expit}(\eta) \\ &= -\log \frac{1}{1 + e^{-\eta}} \\ &= \log(1 + e^{-\eta}) \\ &= \log(1 + e^{-\tilde{y}\eta}) \end{aligned}$$

If $y = 0$, then $\tilde{y} = -1$ and the first term vanishes. Using $1 - \text{expit}(\eta) = \text{expit}(-\eta)$ from Exercise 2.2 gives

$$\begin{aligned} C &= -\log(1 - \text{expit}(\eta)) \\ &= -\log \text{expit}(-\eta) \\ &= \log(1 + e^{\eta}) \\ &= \log(1 + e^{-\tilde{y}\eta}) \end{aligned}$$

Both cases yield the same expression^a, so the two formulas represent the same loss function under the two label conventions. Under the $\{-1, +1\}$ convention, the loss depends on η and $\tilde{y}$ only through their product $m = \tilde{y}\eta$. The margin m is positive if and only if the observation is on the correct side of the decision boundary of Exercise 4.1, and $|m|$ measures how far the score is from the boundary.

^a<https://morrison-lab.github.io/mds/expressions-algorithms.html#def-expression>

Exercise 5.7 (The same loss, written two ways (part 2)). Use the setup of Exercise 5.6.

Let $m = \tilde{y}\eta$ be the margin. Describe $\log(1 + e^{-m})$ as a function of m : its monotonicity, its value at $m = 0$, and its behaviour as $m \rightarrow +\infty$ and as $m \rightarrow -\infty$. Give the value at $m = 0$ and at $m = \pm 2$ to three significant figures.

Solution

Solution 5.7. The function $\log(1 + e^{-m})$ is strictly decreasing in m because e^{-m} is strictly decreasing. Larger margins therefore always yield smaller loss, and the loss orders observations by their signed margin.

m	$\log(1 + e^{-m})$
-2	2.13
0	0.693
+2	0.127

At $m = 0$ the loss is $\log 2 \approx 0.693$, which is the [log loss](#) of predicting probability $\frac{1}{2}$ for both classes. As $m \rightarrow +\infty$, the loss approaches 0 but remains strictly positive for every finite m . Increasing the margin of an already correctly classified observation always reduces the loss slightly, which leads to the divergence under linear separation analyzed in [Exercise 11.1](#).

As $m \rightarrow -\infty$, $\log(1 + e^{-m}) \approx \log e^{-m} = -m$, so the loss grows *linearly* with the magnitude of a negative margin, whereas squared error grows quadratically. A single severely misclassified observation therefore has bounded influence on the gradient under logistic loss, unlike under squared error, as seen in [Exercise 1.3](#).

Comparing the derivatives with respect to η makes this difference explicit. Fitting a linear predictor η to $\tilde{y} \in \{-1, +1\}$ by least squares uses the loss $(\tilde{y} - \eta)^2 = (1 - m)^2$, whose derivative with respect to η is $2(\eta - \tilde{y})$, whereas the derivative of the logistic loss is $-\tilde{y} \text{expit}(-m)$. Suppose $\tilde{y} = +1$:

- At $\eta = -4$ (a confidently wrong prediction), the logistic derivative is $-\text{expit}(4) \approx -0.98$, whereas the least-squares derivative is $2(-4 - 1) = -10$.
- At $\eta = 4$ (a confidently correct prediction), the logistic derivative is $-\text{expit}(-4) \approx -0.02$, which is near 0, whereas the least-squares derivative is $2(4 - 1) = 6$.

Squared error continues to penalize predictions with $\eta = 4 > 1$ because $\eta \neq \tilde{y}$, and places large weight on observations with large negative margins. The logistic loss avoids both behaviors. [Figure 20](#) plots the logistic, hinge, squared-error, and zero-one misclassification losses as functions of the margin m .

Exercise 5.8 (The same loss, written two ways (part 3)). Use the setup of [Exercise 5.6](#).

Compare the logistic loss with the hinge loss $\max(0, 1 - m)$, which support vector machines use. State one property that the hinge loss has and the logistic loss does not, and one property that the logistic loss has and the hinge loss does not.

Solution

Solution 5.8. The hinge loss $\max(0, 1 - m)$ is identically 0 for all $m \geq 1$. Observations with margin greater than 1 contribute zero gradient, so the fitted [decision boundary](#) of a support vector machine depends only on the observations with $m \leq 1$ (the support vectors). In contrast, the logistic loss is strictly positive for all finite m , so every observation contributes to the fit, with weight decreasing exponentially as $m \rightarrow +\infty$.

Conversely, the logistic loss is smooth everywhere, whereas the hinge loss has a corner at $m = 1$ where it is not differentiable, requiring subgradient methods or constrained quadratic programming. Moreover, because the logistic loss is the negative log-likelihood^a of a Bernoulli model, minimizing it produces calibrated probability estimates^b $\text{expit}(\eta)$, whereas a support vector machine produces an uncalibrated score whose conversion to a probability requires an additional post-processing step.

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik>

^b<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

Exercise 5.9 (The same loss, written two ways (part 4)). Use the setup of [Exercise 5.6](#).

What we actually care about is the misclassification rate, whose per-observation cost is $\mathbb{1}[m \leq 0]$. Say why it is not used for fitting. Then show that

$$\mathbb{1}[m \leq 0] \leq \frac{1}{\log 2} \log(1 + e^{-m}) \quad \text{for every } m$$

and say what reducing the right-hand side therefore gains.

Solution

Solution 5.9. The zero-one misclassification loss $\delta(m \leq 0)$ is piecewise constant^a: its derivative with respect to $\vec{\beta}$ is zero wherever it is differentiable and undefined at $m = 0$, so gradient-based optimizers receive no directional signal. It is also non-convex, and minimizing the number of misclassified training

points with a linear classifier is NP-hard in general.

To verify the upper bound $\delta(m \leq 0) \leq \log(1 + e^{-m}) / \log 2$, we check the two cases $m \leq 0$ and $m > 0$:

- If $m \leq 0$, the left-hand side is 1, and since $e^{-m} \geq 1$, we have $\log(1 + e^{-m}) \geq \log 2$, so the right-hand side is at least $\log 2 / \log 2 = 1$.
- If $m > 0$, the left-hand side is 0, and the right-hand side is strictly positive because $\log(1 + e^{-m}) > 0$ for every finite m .

Thus the inequality holds for all $m \in \mathbb{R}$.

Averaging over the training set shows that the mean^b logistic loss divided by $\log 2$ is an upper bound on the training misclassification rate. Minimizing the logistic loss therefore minimizes a smooth, convex^c surrogate upper bound on the zero-one error rate.

The bound is not tight for positive margins: at $m = 2$, the rescaled logistic loss is $0.127 / \log 2 = 0.183$ even though the misclassification loss is 0, so a model can reduce its logistic loss by increasing positive margins without changing any classification decision.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

^b<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

^c<https://morrison-lab.github.io/mds/algebra-sums.html#def-convex-function>

Theorem 5.2 (The log loss depends on the margin alone). *Let $y \in \{0, 1\}$, $\tilde{y} = 2y - 1 \in \{-1, +1\}$ and $\eta = \vec{\beta} \cdot \tilde{x}$. Then the per-observation log loss,*

$$L(y, \text{expit}(\eta)) = -y \log \text{expit}(\eta) - (1 - y) \log(1 - \text{expit}(\eta)),$$

equals $\log(1 + e^{-m})$ with the margin $m = \tilde{y}\eta$. This loss is strictly decreasing in m .

i Proof

Proof. This is the solution to Exercise 5.6 and Exercise 5.7: see Solution 5.7. □

Example 5.6 (Evaluating the log loss from the score and from the margin). Consider a linear score $\eta = 2$, so $\pi = \text{expit}(2)$ is 0.881, and compare the log loss under the two labels $y = 1$ ($\tilde{y} = +1$, margin $m = +2$) and $y = 0$ ($\tilde{y} = -1$, margin $m = -2$):

5.9 R

```
z_mf <- 2
y_mf <- c(1, 0)
y_tilde_mf <- 2 * y_mf - 1
m_mf <- y_tilde_mf * z_mf
p_mf <- plogis(z_mf)

from_prob <- -y_mf * log(p_mf) - (1 - y_mf) * log(1 - p_mf)
from_margin <- log1p(exp(-m_mf))
round(data.frame(y = y_mf, y_tilde = y_tilde_mf, margin = m_mf,
                 from_prob = from_prob, from_margin = from_margin), 3)

# A tibble: 2 x 5
  y y_tilde margin from_prob from_margin
<dbl> <dbl> <dbl> <dbl> <dbl>
1 1 1 2 0.127 0.127
2 0 -1 -2 2.13 2.13
```

5.10 Python

```
import numpy as np
import pandas as pd
from scipy.special import expit

z_mf = 2.0
y_mf = np.array([1, 0])
y_tilde_mf = 2 * y_mf - 1
m_mf = y_tilde_mf * z_mf
p_mf = expit(z_mf)

print(pd.DataFrame({
    "y": y_mf,
    "y_tilde": y_tilde_mf,
    "margin": m_mf,
    "from_prob": -y_mf * np.log(p_mf) - (1 - y_mf) * np.log(1 - p_mf),
    "from_margin": np.log1p(np.exp(-m_mf)),
}).round(3))
```

	y	y_tilde	margin	from_prob	from_margin
0	1	1	2.0	0.127	0.127
1	0	-1	-2.0	2.127	2.127

Both formulas return 0.127 on the correctly classified observation ($m = +2$) and 2.13 on the misclassified one ($m = -2$).

```

library(ggplot2)

m_grid <- seq(-2.5, 2.5, length.out = 501)
margin_df <- rbind(
  data.frame(
    m = m_grid,
    loss = log1p(exp(-m_grid)),
    kind = "Logistic: log(1 + exp(-m))"
  ),
  data.frame(
    m = m_grid,
    loss = pmax(0, 1 - m_grid),
    kind = "Hinge: max(0, 1 - m)"
  ),
  data.frame(
    m = m_grid,
    loss = (1 - m_grid)^2,
    kind = "Least squares: (1 - m)^2"
  ),
  data.frame(
    m = m_grid,
    loss = as.numeric(m_grid <= 0),
    kind = "Misclassification: 1[m <= 0]"
  )
)
margin_df$kind <- factor(
  margin_df$kind,
  levels = c(
    "Logistic: log(1 + exp(-m))",
    "Hinge: max(0, 1 - m)",
    "Least squares: (1 - m)^2",
    "Misclassification: 1[m <= 0]"
  )
)

logistic_pts <- data.frame(
  m = c(-2, 0, 2),
  loss = log1p(exp(-c(-2, 0, 2)))
)

ggplot(margin_df) +
  aes(m, loss, colour = kind, linetype = kind) +
  geom_vline(xintercept = 0, colour = "grey70", linetype = "dotted") +
  geom_line(linewidth = 0.8) +
  geom_point(
    data = logistic_pts,
    aes(m, loss),
    inherit.aes = FALSE,
    colour = "#7570b3",
    size = 2.4
  ) +
  scale_colour_manual(values = c(
    "Logistic: log(1 + exp(-m))" = "#7570b3",
    "Hinge: max(0, 1 - m)" = "#d95f02",
    "Least squares: (1 - m)^2" = "#1f78b4",
    "Misclassification: 1[m <= 0]" = "#1b9e77"
  )) +
  scale_linetype_manual(values = c(
    "Logistic: log(1 + exp(-m))" = "solid", 57
    "Hinge: max(0, 1 - m)" = "dashed",
    "Least squares: (1 - m)^2" = "dotdash",
    "Misclassification: 1[m <= 0]" = "longdash"
  ))

```

Exercise 5.10 (Three names for the same optimization). The fit of Exercise 5.3 can be described three ways, and different fields use different ones.

Starting from $\arg \max_{\vec{\beta}} \ell(\vec{\beta})$, derive the other two formulations: minimizing the negative log-likelihood^a, and empirical risk minimization $\arg \min_{\vec{\beta}} \hat{R}(\vec{\beta})$ with $\hat{R} = \frac{1}{n} \sum_i L(y_i, \pi_i)$.

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik>

Solution

Solution 5.10. Maximizing ℓ is minimizing $-\ell$, and dividing by the positive constant n changes neither:

$$\begin{aligned} \arg \max_{\vec{\beta}} \sum_i [y_i \log \pi_i + (1 - y_i) \log(1 - \pi_i)] &= \arg \min_{\vec{\beta}} - \sum_i [\dots] \\ &= \arg \min_{\vec{\beta}} \frac{1}{n} \sum_i L(y_i, \pi_i) \end{aligned}$$

Exercise 5.11 (Three names for the same optimization (part 2)). Use the setup of Exercise 5.10. State the per-observation loss $L(y_i, \pi_i)$.

Solution

Solution 5.11. The per-observation loss is

$$L(y, \pi) = -y \log \pi - (1 - y) \log(1 - \pi),$$

the **log loss**, also called the **cross-entropy loss** — Solution 5.14 says why.

Exercise 5.12 (Three names for the same optimization (part 3)). Use the setup of Exercise 5.10. Identify which step introduces the factor $\frac{1}{n}$.

Solution

Solution 5.12. The $\frac{1}{n}$ enters at the last step and nowhere else.

Exercise 5.13 (Three names for the same optimization (part 4)). Use the setup of Exercise 5.10. State what the factor $\frac{1}{n}$ does and does not change, referring to Exercise 12.7.

Solution

Solution 5.13. It cannot move the minimizer, because scaling an objective by a positive constant^a leaves its argmin^b alone, and it cannot change the score equations, because those set a gradient to zero and zero scales to zero.

It does change every quantity that is not an argmin: the gradient's size, and so any step size tuned against it; and the second derivative, and so any standard error^c computed from it. Exercise 12.7 is the same distinction seen from the opposite side, where a $\frac{1}{n}$ had been attached to a log-likelihood^d that did not have one.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

^b<https://morrison-lab.github.io/mds/algebra-basics.html#def-argmin>

^c<https://morrison-lab.github.io/sds/estimation.html#def-SE>

^d<https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik>

Exercise 5.14 (Three names for the same optimization (part 5)). Use the setup of Exercise 5.10. For distributions^a $\tilde{p}$ and $\tilde{q}$ over the same outcomes, the **cross-entropy**^b is $H(\tilde{p}, \tilde{q}) = -\sum_j p_j \log q_j$. Let $p = (1 - y, y)$ be the point-mass distribution^c at the observed label $y \in \{0, 1\}$, and let $\tilde{q} = (1 - \pi, \pi)$. Show that $H(\tilde{p}, \tilde{q})$ equals the per-observation loss of Exercise 5.10.

^a<https://morrison-lab.github.io/pds/random-variables.html#def-cdf>

^b<https://morrison-lab.github.io/pds/expectation.html#def-cross-entropy>

^c<https://morrison-lab.github.io/pds/random-variables.html#def-cdf>

Solution

Solution 5.14. The two distributions^a are over the two outcomes $\{0, 1\}$, so the sum has two terms:

$$\begin{aligned} H(\tilde{p}, \tilde{q}) &= -p_0 \log q_0 - p_1 \log q_1 \\ &= -(1 - y) \log(1 - \pi) - y \log \pi, \end{aligned}$$

which is $L(y, \pi)$ exactly.

So the **log loss** is the cross-entropy^b between the label, read as a degenerate distribution^c putting all its mass on the observed outcome, and the distribution the model predicts. The three names in this exercise's title are one calculation.

^a<https://morrison-lab.github.io/pds/random-variables.html#def-cdf>

^b<https://morrison-lab.github.io/pds/expectation.html#def-cross-entropy>

^c<https://morrison-lab.github.io/pds/random-variables.html#def-cdf>

Exercise 5.15 (Three names for the same optimization (part 6)). Use the setup of Exercise 5.10. Show $H(\tilde{p}, \tilde{q}) \geq H(\tilde{p}, \tilde{p})$, with equality only when $\tilde{q} = \tilde{p}$.

Solution

Solution 5.15. Subtract and collect:

$$\begin{aligned} H(\tilde{p}, \tilde{q}) - H(\tilde{p}, \tilde{p}) &= \sum_j p_j \log \frac{q_j}{p_j} \\ &=: D_{\text{KL}}(\tilde{p} \parallel \tilde{q}). \end{aligned}$$

This difference is the Kullback–Leibler divergence^a of $\tilde{q}$ from $\tilde{p}$. Use $\log x \leq x - 1$, with equality only at $x = 1$:

$$\begin{aligned} -D_{\text{KL}}(\tilde{p} \parallel \tilde{q}) &= \sum_j p_j \log \frac{q_j}{p_j} \\ &\leq \sum_j p_j \left(\frac{q_j}{p_j} - 1 \right) \\ &= \sum_j q_j - \sum_j p_j \\ &= 1 - 1 \\ &= 0, \end{aligned}$$

so $D_{\text{KL}} \geq 0$ and $H(\tilde{p}, \tilde{q}) \geq H(\tilde{p}, \tilde{p})$. Equality needs $q_j/p_j = 1$ wherever $p_j > 0$, which for two distributions^b that both sum to one implies $\tilde{q} = \tilde{p}$.

^a<https://morrison-lab.github.io/pds/expectation.html#def-kl-divergence>

^b<https://morrison-lab.github.io/pds/random-variables.html#def-cdf>

Exercise 5.16 (Three names for the same optimization (part 7)). Use the setup of Exercise 5.10. Evaluate $H(\tilde{p}, \tilde{p})$ for the $\tilde{p}$ of Exercise 5.14.

Solution

Solution 5.16. Here $\tilde{p} = (1 - y, y)$ with $y \in \{0, 1\}$, so all its mass is on one outcome and

$$\begin{aligned} H(\tilde{p}, \tilde{p}) &= -1 \log 1 - 0 \log 0 \\ &= 0, \end{aligned}$$

taking $0 \log 0 = 0$, its limit.

Exercise 5.17 (Three names for the same optimization (part 8)). Use the setup of Exercise 5.10. State the smallest achievable per-observation loss and when it is attained.

Solution

Solution 5.17. The smallest per-observation loss is therefore 0, achieved only when $\pi = y$ exactly. That is unreachable. $\pi = \text{expit}(\eta) \in (0, 1)$ strictly for every finite η , by Exercise 2.1, so the loss is strictly positive on every observation no matter how well the model fits. It is the same open interval that made Exercise 11.1's likelihood^a keep increasing toward its supremum without reaching it: the best available fit is always an approach to a lower bound that is never reached.

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik>

Exercise 5.18 (Three names for the same optimization (part 9)). Use the setup of Exercise 5.10. Show that a distribution^a over M outcomes has $H(\tilde{p}, \tilde{p}) \in [0, \log M]$, and state which distributions^b attain the minimum and the maximum.

^a<https://morrison-lab.github.io/pds/random-variables.html#def-cdf>

^b<https://morrison-lab.github.io/pds/random-variables.html#def-cdf>

Solution

Solution 5.18. Each term $-p_j \log p_j$ is non-negative for $p_j \in [0, 1]$, so $H(\tilde{p}, \tilde{p}) \geq 0$, with equality only if every term vanishes — that is, every p_j is 0 or 1. So the minimum 0 is attained exactly at the degenerate distributions^a, one outcome certain.

For the upper end, compare against the uniform $\tilde{u}$ with $u_j = 1/M$ and use Solution 5.15's inequality:

$$\begin{aligned} 0 &\leq D_{\text{KL}}(\tilde{p} \parallel \tilde{u}) \\ &= \sum_j p_j \log(p_j M) \\ &= -H(\tilde{p}, \tilde{p}) + \log M, \end{aligned}$$

so $H(\tilde{p}, \tilde{p}) \leq \log M$, with equality exactly when $\tilde{p} = \tilde{u}$. A certain outcome has zero entropy^b. The uniform $\tilde{u}$ distribution^c has the maximum entropy.

^a<https://morrison-lab.github.io/pds/random-variables.html#def-cdf>

^b<https://morrison-lab.github.io/pds/expectation.html#def-entropy>

^c<https://morrison-lab.github.io/pds/random-variables.html#def-cdf>

Exercise 5.19 (Three names for the same optimization (part 10)). Use the setup of Exercise 5.10. ℓ_2 regularization replaces $\hat{R}$ by $\hat{R} + \lambda \sum_{k=1}^p \beta_k^2$, summing from $k = 1$ so that the intercept^a β_0 is unpenalized. Determine how penalizing β_0 would alter the intercept score equation of Exercise 6.5, and

use that result to explain why β_0 is left unpenalized.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

Solution

Solution 5.19. Differentiate the objective $\hat{R}(\vec{\beta}) + \lambda \sum_{k \geq 1} \beta_k^2$ with respect to β_0 . Unpenalized, the intercept^a column of $D(\mathbf{x})$ is ones, so

$$\frac{\partial \hat{R}}{\partial \beta_0} = -\frac{1}{n} \sum_i (y_i - \pi_i) \quad \Rightarrow \quad \sum_i \pi_i = \sum_i y_i, \\ = 0$$

which is Exercise 6.5: the fitted probabilities sum to the observed number of positive outcomes. Now add $\lambda \beta_0^2$ to the objective. The stationarity condition becomes $-\frac{1}{n} \sum_i (y_i - \pi_i) + 2\lambda \beta_0 = 0$, that is

$$\sum_i \pi_i = \sum_i y_i - 2n\lambda \hat{\beta}_0.$$

Aggregate calibration no longer holds, and the difference between $\sum_i \pi_i$ and $\sum_i y_i$ does not depend on the predictors. Because the penalty shrinks $\hat{\beta}_0$ toward 0, and $\beta_0 = 0$ corresponds to a baseline probability of $\frac{1}{2}$, shrinking β_0 biases the average predicted probability toward $\frac{1}{2}$: rare outcomes are systematically over-predicted and common outcomes are systematically under-predicted.

A second reason is invariance to predictor location. A penalty on a slope^b β_k penalizes large rates of change with respect to x_k . The intercept is not a rate of change. Centering a predictor, $x_k \mapsto x_k - \bar{x}_k$, changes $\hat{\beta}_0$ and leaves every slope unchanged, so penalizing β_0 makes the fitted probabilities depend on the choice of origin for the predictors. Standardizing the predictors puts the slopes on a common scale, while the intercept still depends on the overall outcome prevalence.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

^b<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope>

Theorem 5.3 (Cross-entropy is smallest at the true distribution). *For distributions p and q over the same outcomes with $q_j > 0$ wherever $p_j > 0$,*

$$H(p, q) \geq H(p, p),$$

with equality only when $q = p$. A term with $p_j = 0$ contributes 0, by the convention $0 \log 0 = 0$. If $q_j = 0$ for some j with $p_j > 0$, then $H(p, q) = \infty$ and the inequality holds trivially. The difference is the Kullback–Leibler divergence $D_{\text{KL}}(p \parallel q) \geq 0$.

i Proof

Proof. This is the solution to Exercise 5.15: see Solution 5.15. □

Example 5.7 (Cross-entropy against a wrong guess). Let $p = (0.9, 0.1)$ be the true distribution and $q = (0.5, 0.5)$ a guess.

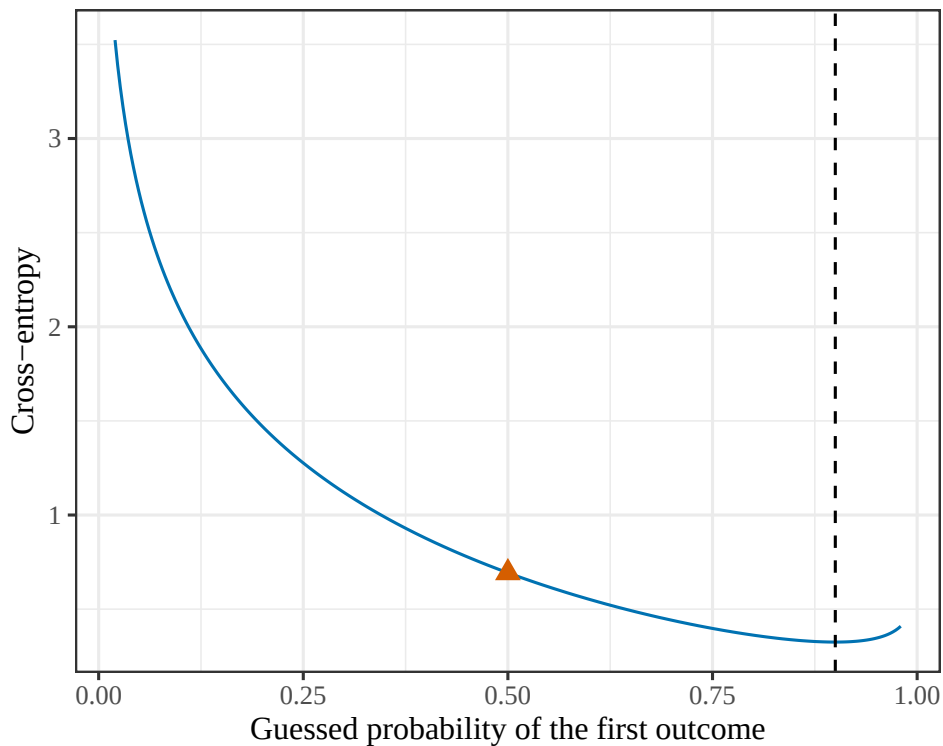
```
ce_p <- c(0.9, 0.1)
ce_q <- c(0.5, 0.5)
ce_pq <- -sum(ce_p * log(ce_q))
ce_pp <- -sum(ce_p * log(ce_p))
c(H_pq = ce_pq, H_pp = ce_pp, KL = ce_pq - ce_pp)
```

H_pq	H_pp	KL
0.6931472	0.3250830	0.3680642

Here $H(p, q) = 0.693 \geq H(\tilde{p}, \tilde{p}) = 0.325$, as Theorem 5.3 says. The gap $D_{\text{KL}}(\tilde{p} \parallel q) = 0.368$ is positive because $\tilde{q} \neq \tilde{p}$.

5.11 R

```
library(ggplot2)
ce_curve <- data.frame(q1 = seq(0.02, 0.98, length.out = 300))
ce_curve$ce <- -(ce_p[1] * log(ce_curve$q1) + ce_p[2] * log(1 - ce_curve$q1))
ggplot(ce_curve, aes(q1, ce)) +
  geom_line(colour = "#0072B2") +
  geom_vline(xintercept = ce_p[1], linetype = "dashed") +
  geom_point(
    data = data.frame(q1 = ce_q[1], ce = ce_pq),
    colour = "#D55E00", shape = 17, size = 3
  ) +
  labs(x = "Guessed probability of the first outcome", y = "Cross-entropy")
```

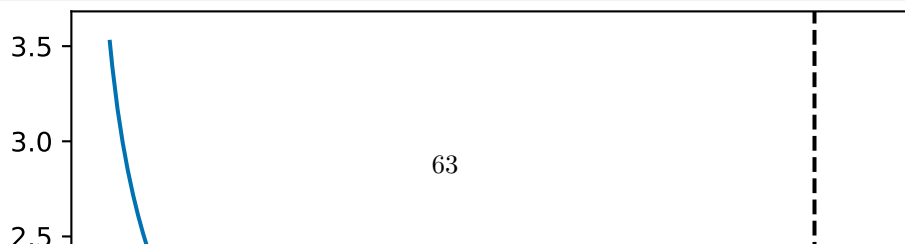


(a)

5.12 Python

```
import matplotlib.pyplot as plt
import numpy as np

p_ce = np.array([0.9, 0.1])
q1_ce = np.linspace(0.02, 0.98, 300)
fig, ax = plt.subplots(figsize=(5, 4), layout="constrained")
ax.plot(q1_ce, -(p_ce[0] * np.log(q1_ce) + p_ce[1] * np.log(1 - q1_ce)), color="#0072B2");
ax.axvline(p_ce[0], color="black", linestyle="--");
ax.plot(0.5, -(p_ce[0] * np.log(0.5) + p_ce[1] * np.log(0.5)), color="#D55E00", marker="^", markersize=30);
ax.set_xlabel("Guessed probability of the first outcome");
ax.set_ylabel("Cross-entropy");
plt.show()
```



Theorem 5.4 (The range of the entropy). *A distribution $p_{\sim}$ over M outcomes has*

$$0 \leq H(p_{\sim}) \leq \log M.$$

The minimum 0 holds exactly when one outcome is certain, and the maximum $\log M$ exactly when $p_{\sim}$ is uniform.

i Proof

Proof. This is the solution to Exercise 5.18: see Solution 5.18. □

Example 5.8 (Entropy of three distributions over four outcomes).

```
en_entropy <- function(p) -sum(ifelse(p > 0, p * log(p), 0))
en_dists <- list(
  certain = c(1, 0, 0, 0),
  skewed = c(0.7, 0.1, 0.1, 0.1),
  uniform = rep(0.25, 4)
)
vapply(en_dists, en_entropy, numeric(1))
```

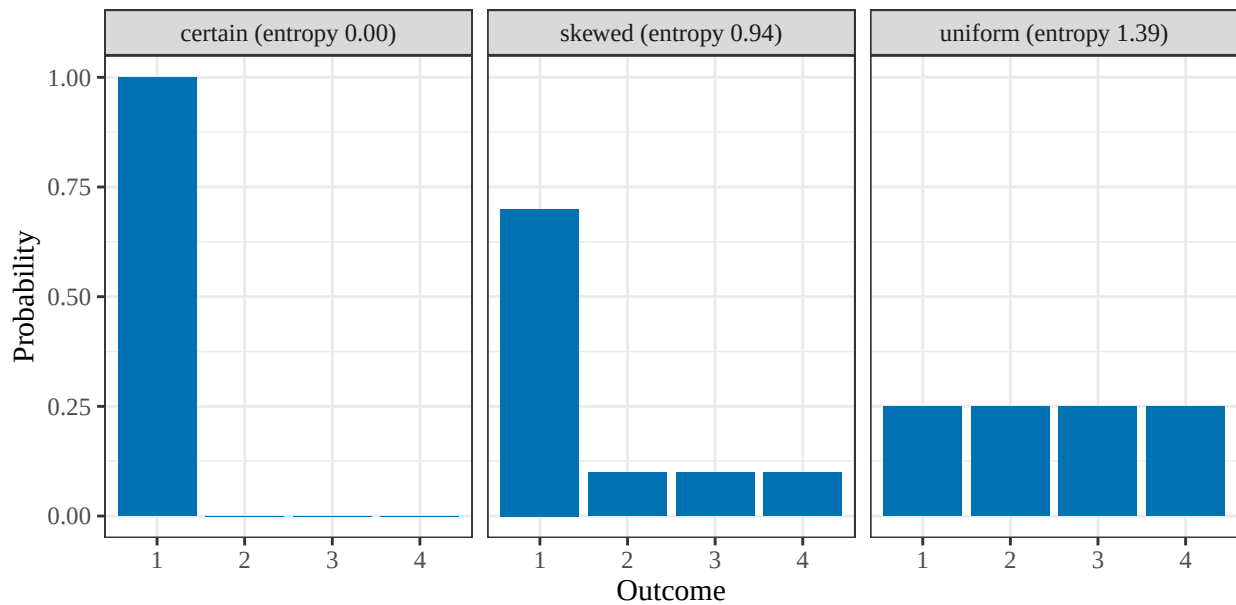
```
   certain   skewed   uniform
0.000000 0.940448 1.386294
```

With $M = 4$ outcomes, Theorem 5.4 puts the entropy between 0 and $\log 4 = 1.386$. The certain distribution^a has entropy 0, the uniform distribution has entropy $\log 4$, and the skewed distribution has entropy at 0.94, in-between the other two.

^a<https://morrison-lab.github.io/pds/random-variables.html#def-cdf>

5.13 R

```
library(ggplot2)
enf <- do.call(rbind, lapply(names(en_dists), function(nm) {
  data.frame(
    distribution = sprintf("%s (entropy %.2f)", nm, en_entropy(en_dists[[nm]]) + 0),
    outcome = factor(1:4),
    prob = en_dists[[nm]]
  )
}))
enf$distribution <- factor(enf$distribution, levels = unique(enf$distribution))
ggplot(enf, aes(outcome, prob)) +
  geom_col(fill = "#0072B2") +
  facet_wrap(vars(distribution)) +
  labs(x = "Outcome", y = "Probability")
```



(a)

5.14 Python

```
import matplotlib.pyplot as plt
import numpy as np

def entropy_enf(p):
    p = np.asarray(p, float)
    return -np.sum(np.where(p > 0, p * np.log(np.where(p > 0, p, 1)), 0))

dists_enf = {
    "certain": [1, 0, 0, 0],
    "skewed": [0.7, 0.1, 0.1, 0.1],
    "uniform": [0.25] * 4,
}

fig, axes = plt.subplots(1, 3, figsize=(7, 3.5), sharey=True, layout="constrained")
for ax, (name, p) in zip(axes, dists_enf.items()):
    ax.bar(range(1, 5), p, color="#0072B2");
    ax.set_title(f"{name} (entropy {entropy_enf(p) + 0.0:.2f})", fontsize=9);
    ax.set_xticks(range(1, 5));
    ax.set_xlabel("Outcome");
axes[0].set_ylabel("Probability");
plt.show()
```

65

certain (entropy 0.00) skewed (entropy 0.94) uniform (entropy 1.39)

5.15 Where cross-entropy comes from

Cross-entropy is not a machine learning invention. It comes from Claude Shannon’s 1948 theory of communication [16], where it measures the average number of bits per symbol required when encoding a source with a code designed for another distribution³⁵.

For a source that emits symbol j with probability p_j , no uniquely decodable code averages fewer than $H(p, p)$ bits per symbol, the base-2 entropy³⁶ of Theorem 5.4. A code built for a distribution³⁷ q uses on average about the base-2 cross-entropy³⁸ $H(p, q)$ bits per symbol³⁹. The difference $D_{\text{KL}}(p \parallel q)$ of Theorem 5.3 is the average number of extra bits per symbol required by using q instead of p , and it is zero if and only if $q = p$. For the source-coding theorems behind these bounds, see Source Coding⁴⁰ in *Information Theory for Data Science*.

Minimizing cross-entropy⁴¹ over q brings q closer to p , and fitting a [logistic regression](#) does this at each observation. The reference distribution⁴² $\tilde{p}$ places probability 1 on the observed label and 0 on the other outcome, while the model’s distribution q is $(1 - \pi, \pi)$. A distribution that places probability 1 on a single outcome has entropy⁴³ 0, so for a labeled training point the cross-entropy and the Kullback–Leibler divergence⁴⁴ are equal.

The base of the logarithm chooses the unit of information. Base 2 measures information in bits, and the natural logarithm measures it in *nats*. Since $\log_2 z = \ln z / \ln 2$, switching between them multiplies every term by a positive constant⁴⁵, so it does not change the minimizer. Throughout this chapter $\log$ denotes the natural logarithm, so the losses are measured in nats.

i Watch

- Weinberger fits [logistic regression](#) by maximum likelihood^a, from 30:50^b [10], Lecture 11.
- Starmer also fits logistic regression by maximum likelihood^c [17].
- He explains cross-entropy^d [18] as a loss for a neural network’s classifier.

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle>

^b<https://www.youtube.com/watch?v=GnKDzIOxfzI&t=1850s>

^c<https://www.youtube.com/watch?v=BfKanl1aSG0>

^d<https://www.youtube.com/watch?v=6ArSys5qHAU>

i Read

GeeksforGeeks explains binary cross-entropy^a [19].

^a<https://www.geeksforgeeks.org/deep-learning/binary-cross-entropy-log-loss-for-binary-classification/>

6 The gradient

There is no closed-form expression⁴⁶ for $\hat{\beta}$, so we compute it iteratively by the methods of the gradient descent chapter. Those methods require the gradient of the [log loss](#). Exercise 6.1 shows that the derivative of the logistic function⁴⁷ cancels in this gradient, leaving an expression of the same form as in linear regression.

³⁵<https://morrison-lab.github.io/pds/random-variables.html#def-cdf>

³⁶<https://morrison-lab.github.io/pds/expectation.html#def-entropy>

³⁷<https://morrison-lab.github.io/pds/random-variables.html#def-cdf>

³⁸<https://morrison-lab.github.io/pds/expectation.html#def-cross-entropy>

³⁹<https://morrison-lab.github.io/ids/source-coding.html#thm-wrong-code>

⁴⁰<https://morrison-lab.github.io/ids/source-coding.html>

⁴¹<https://morrison-lab.github.io/pds/expectation.html#def-cross-entropy>

⁴²<https://morrison-lab.github.io/pds/random-variables.html#def-cdf>

⁴³<https://morrison-lab.github.io/pds/expectation.html#def-entropy>

⁴⁴<https://morrison-lab.github.io/pds/expectation.html#def-kl-divergence>

⁴⁵<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

⁴⁶<https://morrison-lab.github.io/mds/expressions-algorithms.html#def-closed-form-expression>

⁴⁷<https://morrison-lab.github.io/mds/functions.html#def-logistic-function>

Exercise 6.1 (The gradient, and the factor that cancels). Let $z_i = \vec{\beta} \cdot \tilde{x}_i$, $\pi_i = \text{expit}(z_i)$, and $C_i = -y_i \log \pi_i - (1 - y_i) \log(1 - \pi_i)$ be observation i 's contribution to the [log loss](#) of Exercise 5.3. Show that $\frac{\partial C_i}{\partial z_i} = \pi_i - y_i$. Use $\text{expit}' = \text{expit}(1 - \text{expit})$ from Exercise 2.3, and do not split into cases.

Solution

Solution 6.1. Differentiate the two terms and substitute $\text{expit}'(z_i) = \pi_i(1 - \pi_i)$:

$$\begin{aligned} \frac{\partial C_i}{\partial z_i} &= -y_i \frac{\text{expit}'(z_i)}{\pi_i} + (1 - y_i) \frac{\text{expit}'(z_i)}{1 - \pi_i} \\ &= -y_i \frac{\pi_i(1 - \pi_i)}{\pi_i} + (1 - y_i) \frac{\pi_i(1 - \pi_i)}{1 - \pi_i} \end{aligned}$$

The π_i cancels in the first term and the $1 - \pi_i$ in the second:

$$\begin{aligned} &= -y_i(1 - \pi_i) + (1 - y_i)\pi_i \\ &= -y_i + y_i\pi_i + \pi_i - y_i\pi_i \\ &= \pi_i - y_i \end{aligned}$$

The derivative of the cost with respect to the score is the **residual**: predicted probability minus observed label. Nothing is left of the exponentials, and nothing is left of expit' .

Exercise 6.2 (The gradient, and the factor that cancels (part 2)). Use the setup of Exercise 6.1. Hence give $\nabla_{\vec{\beta}} C$ for the whole dataset, first as a sum over observations and then in matrix form, where $D(\mathbf{x})$ is the $n \times (p + 1)$ design matrix and $\tilde{\pi}$ is the vector of fitted probabilities.

Solution

Solution 6.2. $z_i = \vec{\beta} \cdot \tilde{x}_i$, so $\nabla_{\vec{\beta}} z_i = \tilde{x}_i$, and the chain rule gives

$$\begin{aligned} \nabla_{\vec{\beta}} C &= \sum_{i=1}^n (\pi_i - y_i) \tilde{x}_i \\ &= X^\top (\tilde{\pi} - y) \end{aligned}$$

Least squares has gradient $D(\mathbf{x})^\top (D(\mathbf{x})\vec{\beta} - y)$, which is the same expression^a with $\tilde{\pi}$ in place of the linear fit $D(\mathbf{x})\vec{\beta}$. The two models have *identical* gradient structure: design matrix transposed, times [residual](#). Only the link from $\vec{\beta}$ to the fitted value differs, and that difference is exactly what stops the equation having a closed-form solution^b — see why we cannot just solve for it.

One observation at a time, the residual shows what each point does to a descent step $\vec{\beta} \leftarrow \vec{\beta} - \alpha \nabla_{\vec{\beta}} C$:

- If $y_i = 0$ and $\pi_i \approx 0$, or $y_i = 1$ and $\pi_i \approx 1$, the residual is near zero, and the point barely changes the gradient.
- If $y_i = 1$ and $\pi_i \approx 0$, the step adds in about $\alpha \tilde{x}_i$ — a larger inner product gives a larger prediction.
- If $y_i = 0$ and $\pi_i \approx 1$, the step subtracts about $\alpha \tilde{x}_i$ — a smaller inner product gives a smaller prediction.

The magnitude of the residual, $|\pi_i - y_i|$, is the probability that the model assigns to the opposite label $1 - y_i$. Each observation therefore contributes to the gradient in the direction of its feature vector $\tilde{x}_i$, weighted by the model's predicted probability of misclassifying that observation.

^a<https://morrison-lab.github.io/mds/expressions-algorithms.html#def-expression>

^b<https://morrison-lab.github.io/mds/expressions-algorithms.html#def-closed-form-solution>

Exercise 6.3 (The gradient, and the factor that cancels (part 3)). Use the setup of Exercise 6.1. Exercise 6.1 has a factor that cancels. Differentiate the squared-error cost $S_i = \frac{1}{2}(\pi_i - y_i)^2$ with respect to z_i instead, and compare the two derivatives at $z_i = 8$ with $y_i = 0$, an observation for which the predicted probability is close to 1 while the observed label is 0. Report the ratio.

Solution

Solution 6.3. By the chain rule,

$$\begin{aligned}\frac{\partial S_i}{\partial z_i} &= (\pi_i - y_i) \text{expit}'(z_i) \\ &= (\pi_i - y_i) \pi_i (1 - \pi_i)\end{aligned}$$

The factor $\text{expit}'(z_i)$ remains in the derivative because squared error has no logarithm to cancel it. At $z_i = 8$ with $y_i = 0$:

$$\begin{aligned}\pi_i &= \text{expit}(8) \\ &= 0.99966,\end{aligned}$$

so

- **log loss:** $\pi_i - y_i = 0.99966$
- **squared error:** $0.99966 \times 0.99966 \times 0.00033535 = 3.351 \times 10^{-4}$

The log-loss derivative is about 2983 times larger than the squared-error derivative.

Although the model assigns probability 0.9997 to the wrong class, squared error produces a gradient of only 3.35×10^{-4} , so gradient descent makes negligible progress. This vanishing-gradient behavior makes squared error unsuitable for saturating outputs, whereas in Solution 6.1 the factor $1/\pi_i$ from differentiating the logarithm cancels $\text{expit}'(z_i)$.

Exercise 2.4 showed that expit is nearly flat far from the **decision boundary**. Log loss is constructed so that this flatness does not diminish the gradient on misclassified observations.

Exercise 6.4 (The gradient, and the factor that cancels (part 4)). Use the setup of Exercise 6.1. Show that $\|\nabla_{\vec{\beta}} C_i\| < \|\tilde{x}_i\|$ for every $\vec{\beta}$, and say what that guarantees.

Solution

Solution 6.4. Because $\nabla_{\vec{\beta}} C_i = (\pi_i - y_i) \tilde{x}_i$, we have $\|\nabla_{\vec{\beta}} C_i\| = |\pi_i - y_i| \cdot \|\tilde{x}_i\|$. Since $\pi_i \in (0, 1)$ by Exercise 2.1 and $y_i \in \{0, 1\}$, $|\pi_i - y_i| < 1$ and the bound follows.

This bound guarantees that no single observation can produce a gradient larger in norm than $\|\tilde{x}_i\|$, regardless of how large the classification error is. The contribution of any single observation to a gradient step is bounded by the norm of its feature vector, which is one reason to standardize features on different scales before fitting.

Applying the same bound to the second derivative yields an explicit step-size condition. The Hessian^a is $D(\mathbf{x})^\top W D(\mathbf{x})$ with $W = \text{diag}(\pi_i(1 - \pi_i))$, and Exercise 2.4 showed that every $\pi_i(1 - \pi_i) \leq 1/4$. Every eigenvalue of the Hessian is therefore at most $\frac{1}{4} \lambda_{\max}(X^\top X)$ for all $\vec{\beta}$.

This uniform curvature bound determines a safe step size. The step-size bound shows the stability condition $\alpha < 2/\lambda_{\max}(\nabla^2 C)$ for a quadratic objective. Using the uniform upper bound $\frac{1}{4} \lambda_{\max}(X^\top X)$ on the eigenvalues of $\nabla^2 C$ gives

$$\alpha < \frac{8}{\lambda_{\max}(X^\top X)}$$

Exercise 7.7 evaluates this bound on a small dataset where it gives $\alpha < 4/3$, compared with the exact local stability threshold of $3/2$. In practice, a line search chooses step sizes adaptively without computing $\lambda_{\max}(X^\top X)$.

^a<https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian>

Exercise 6.5 (The gradient, and the factor that cancels (part 5)). Use the setup of Exercise 6.1. At the maximum the gradient vanishes, giving the score equations $D(\mathbf{x})^\top (\pi - y) = \vec{0}$. Suppose the first column of $D(\mathbf{x})$ is a column of ones, and the second is a binary indicator $x_i \in \{0, 1\}$ for membership of some group. State in words what each of those two equations says about the fitted model.

Solution

Solution 6.5. The column of ones gives $\sum_i \pi_i = \sum_i y_i$: the sum of the fitted probabilities equals the observed number of positive outcomes. A logistic regression with an intercept^a is therefore calibrated in aggregate on the training set: if 30 of 100 training observations have $y_i = 1$, the fitted probabilities sum to 30.

The binary indicator gives $\sum_{i: x_i=1} \pi_i = \sum_{i: x_i=1} y_i$, so the same equality holds *within the subgroup* where $x_i = 1$. Subtracting this subgroup equation from the intercept equation shows that it also holds in the complementary subgroup where $x_i = 0$.

Each indicator feature therefore enforces aggregate calibration within the subgroup it defines, and a categorical^b predictor encoded with indicator variables forces the fitted probabilities to match the observed positive counts in every level of the predictor. The model cannot be miscalibrated in aggregate on any subgroup represented in the design matrix, though it can be miscalibrated on subgroups defined by variables omitted from the model, which motivates checking calibration across held-out subgroup definitions.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

^b<https://morrison-lab.github.io/sds/data.html#def-categorical-var>

Definition 6.1 (Residual of a logistic regression). The **residual** of observation i in logistic regression is its predicted probability minus its label,

$$\pi_i - y_i, \quad \pi_i = \text{expit}(\vec{\beta} \cdot \tilde{x}_i).$$

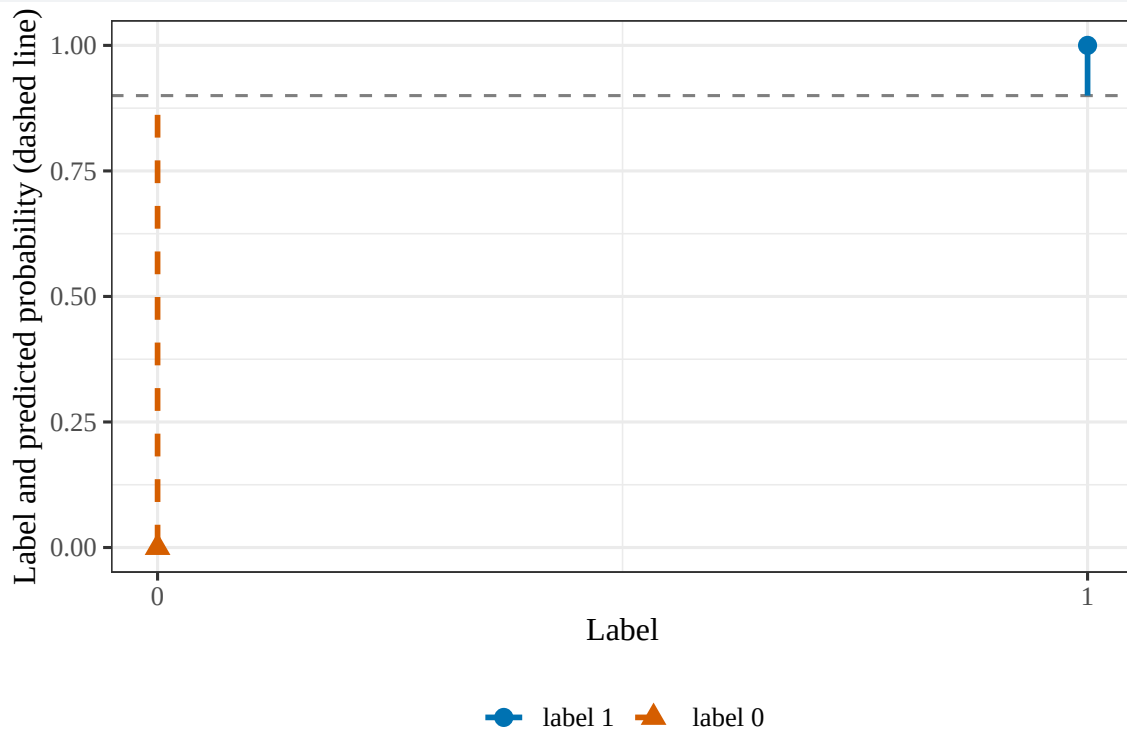
It is the negative of the response residual $y_i - \pi_i$, and it is in $(-1, 1)$.

Example 6.1 (Residuals of two men in the WCGS data). The first man in `data/wcgs.csv`^a with coronary heart disease has $y_i = 1$ and fitted probability $\pi_i = 0.069$, so his residual is $\pi_i - y_i = -0.931$. The first man without it has $y_i = 0$ and $\pi_i = 0.134$, so his residual is 0.134.

^a<https://morrison-lab.github.io/lds/chapters/data/wcgs.csv>

6.1 R

```
library(ggplot2)
lrp_pts <- data.frame(
  label = c(1, 0),
  prob = 0.9,
  kind = factor(c("label 1", "label 0"), levels = c("label 1", "label 0"))
)
ggplot(lrp_pts, aes(x = label, y = label, colour = kind)) +
  geom_hline(yintercept = 0.9, linetype = "dashed", colour = "grey50") +
  geom_segment(aes(xend = label, y = label, yend = prob, linetype = kind), linewidth = 1) +
  geom_point(aes(shape = kind), size = 3) +
  scale_x_continuous(breaks = c(0, 1)) +
  scale_colour_manual(values = c("label 1" = "#0072B2", "label 0" = "#D55E00")) +
  scale_shape_manual(values = c("label 1" = 16, "label 0" = 17)) +
  scale_linetype_manual(values = c("label 1" = "solid", "label 0" = "dashed")) +
  labs(
    x = "Label", y = "Label and predicted probability (dashed line)",
    colour = NULL, shape = NULL, linetype = NULL
  )
)
```



(a)

6.2 Python

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.axhline(0.9, color="grey", linestyle="--");
ax.vlines(1, 0.9, 1, color="#0072B2", linestyle="-", linewidth=2);
ax.plot(1, 1, color="#0072B2", marker="o", markersize=8, linestyle="", label="label 1");
ax.vlines(0, 0, 0.9, color="#D55E00", linestyle="--", linewidth=2);
ax.plot(0, 0, color="#D55E00", marker="^", markersize=8, linestyle="", label="label 0");
ax.set_xticks([0, 1]);
ax.set_xlabel("Label");
ax.set_ylabel("Label and predicted probability (dashed line)");
fig.legend(loc="outside lower center", ncol=2);
plt.show()
```

Theorem 6.1 (Derivative of the log loss with respect to the score). *Let $z_i = \vec{\beta} \cdot \tilde{x}_i$ and $\pi_i = \text{expit}(z_i)$. Observation i 's contribution to the log loss of Definition 5.1 has derivative*

$$\frac{\partial C_i}{\partial z_i} = \pi_i - y_i,$$

the residual of Definition 6.1.

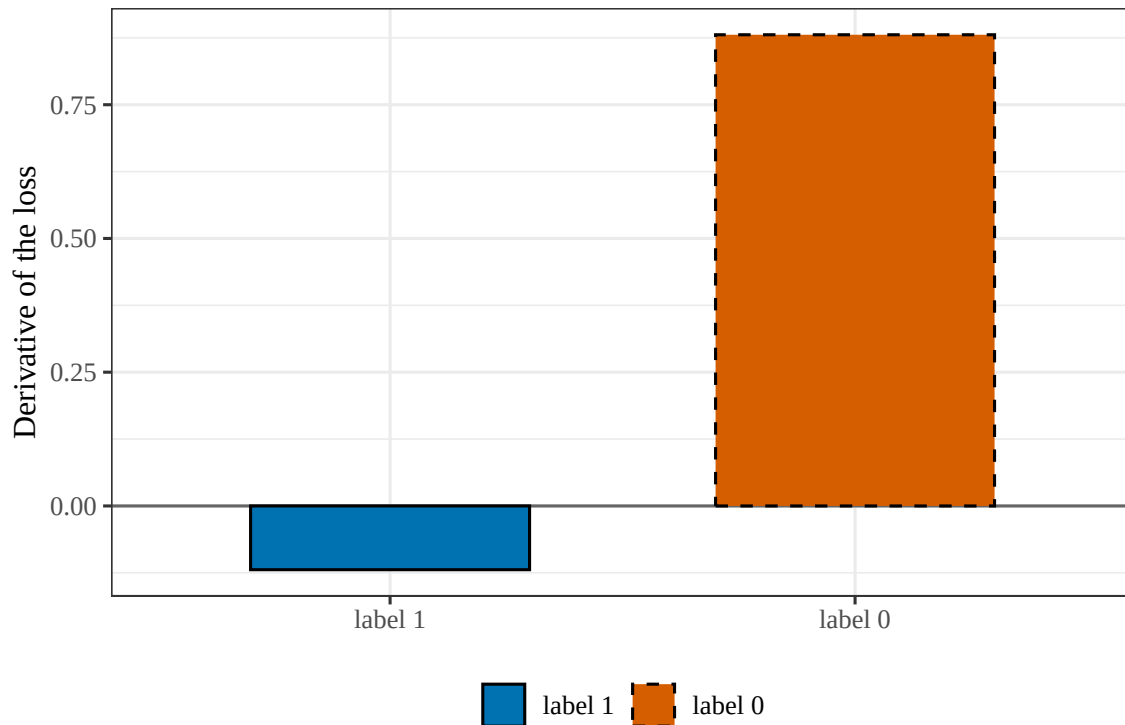
i Proof

Proof. By Solution 6.1. □

Example 6.2 (The residual of a man with a body mass index of 25). A man with $\text{bmi} = 25$ has $\pi_i = 0.083$ in the model of Example 2.2 (Example 2.3). If he has coronary heart disease, $y_i = 1$ and $\frac{\partial C_i}{\partial z_i} = \pi_i - 1 = -0.917$. If he does not, $y_i = 0$ and the derivative is $\pi_i = 0.083$: the derivative is about 11 times as large in magnitude when the label is 1, because the model had judged his disease unlikely.

6.3 R

```
library(ggplot2)
lrf_bars <- data.frame(
  label = factor(c("label 1", "label 0"), levels = c("label 1", "label 0")),
  derivative = c(plogis(2) - 1, plogis(2) - 0)
)
ggplot(lrf_bars, aes(label, derivative, fill = label, linetype = label)) +
  geom_hline(yintercept = 0, colour = "grey40") +
  geom_col(colour = "black", width = 0.6) +
  scale_fill_manual(values = c("label 1" = "#0072B2", "label 0" = "#D55E00")) +
  scale_linetype_manual(values = c("label 1" = "solid", "label 0" = "dashed")) +
  labs(x = NULL, y = "Derivative of the loss", fill = NULL, linetype = NULL)
```

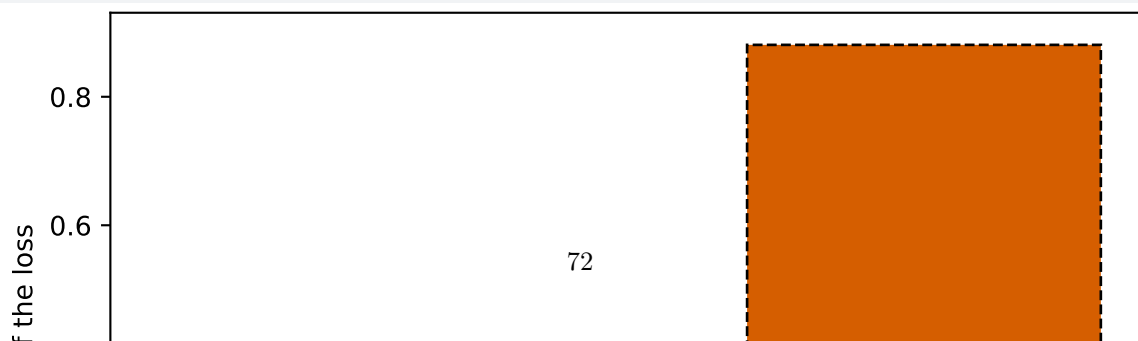


(a)

6.4 Python

```
import matplotlib.pyplot as plt
from scipy.special import expit

fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.axhline(0, color="grey");
ax.bar(["label 1"], [expit(2) - 1], color="#0072B2", edgecolor="black", width=0.6);
ax.bar(["label 0"], [expit(2) - 0], color="#D55E00", edgecolor="black", linestyle="--", width=0.6);
ax.set_ylabel("Derivative of the loss");
plt.show()
```



Theorem 6.2 (Gradient of the log loss). *The gradient of the log loss of Definition 5.1 with respect to $\vec{\beta}$ is*

$$\nabla C(\vec{\beta}) = \sum_{i=1}^n (\pi_i - y_i) \tilde{x}_i = D^\top (\tilde{\pi} - y),$$

where D is the design matrix, with the constant column of ones that absorbs the intercept, and $\tilde{\pi}$ is the vector of fitted probabilities.

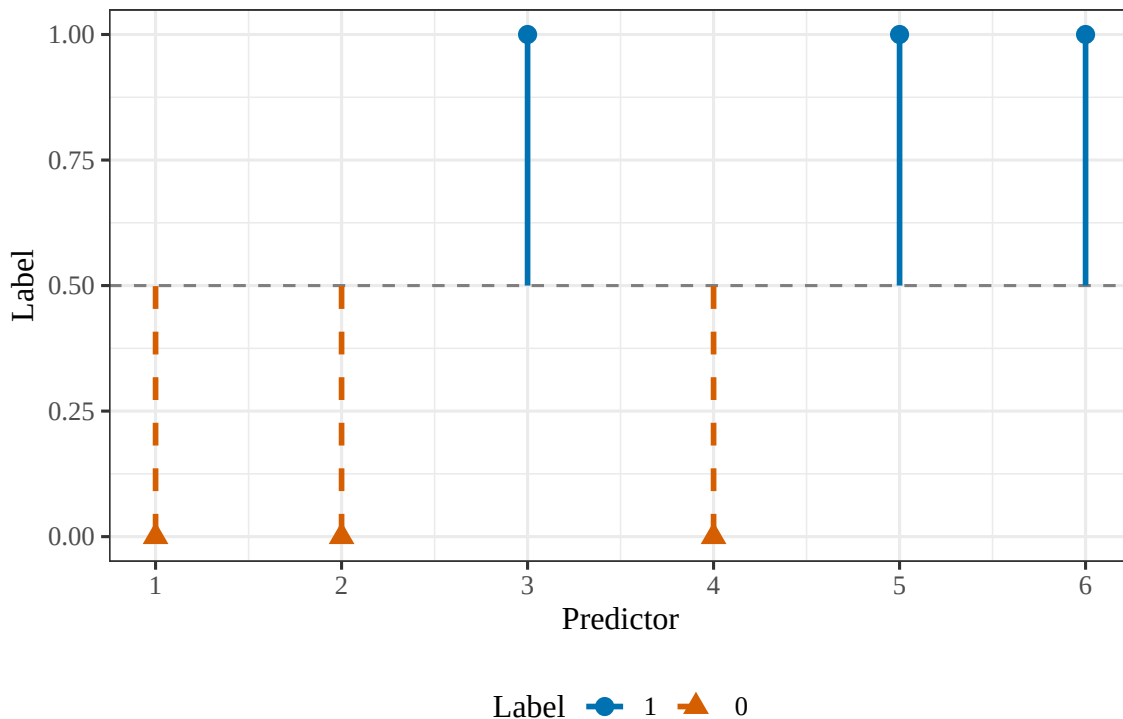
i Proof

Proof. By Theorem 6.1 and the chain rule, as in Solution 6.2. □

Example 6.3 (The gradient at zero coefficients). Consider six points with predictor values $1, \dots, 6$ and labels $0, 0, 1, 0, 1, 1$, evaluated at $\vec{\beta} = \vec{0}$. Every fitted probability is then $\pi_i = \text{expit}(0) = 1/2$. The intercept entry of $D^\top (\tilde{\pi} - y)$ is $\sum_{i=1}^n (1/2 - y_i) = 0$. The slope entry is $\sum_{i=1}^n (1/2 - y_i)x_i = -3.5$. The negative slope entry says that the loss falls as the slope increases, so a first step of gradient descent raises the slope.

6.5 R

```
library(ggplot2)
lgg_pts <- data.frame(
  x = 1:6,
  label = factor(c(0, 0, 1, 0, 1, 1), levels = c(1, 0))
)
ggplot(lgg_pts, aes(x, as.numeric(as.character(label)), colour = label)) +
  geom_hline(yintercept = 0.5, linetype = "dashed", colour = "grey50") +
  geom_segment(
    aes(xend = x, yend = 0.5, linetype = label), linewidth = 1
  ) +
  geom_point(aes(shape = label), size = 3) +
  scale_colour_manual(values = c("1" = "#0072B2", "0" = "#D55E00")) +
  scale_shape_manual(values = c("1" = 16, "0" = 17)) +
  scale_linetype_manual(values = c("1" = "solid", "0" = "dashed")) +
  scale_x_continuous(breaks = 1:6) +
  labs(x = "Predictor", y = "Label", colour = "Label", shape = "Label", linetype = "Label")
```



(a)

6.6 Python

```
import matplotlib.pyplot as plt
import numpy as np

x_lgg = np.arange(1, 7)
y_lgg = np.array([0, 0, 1, 0, 1, 1])
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.axhline(0.5, color="grey", linestyle="--");
for label_lgg, colour, marker, style in [(1, "#0072B2", "o", "-"), (0, "#D55E00", "^", "--)]:
    keep_lgg = y_lgg == label_lgg
    ax.vlines(x_lgg[keep_lgg], 0.5, y_lgg[keep_lgg], color=colour, linestyle=style, linewidth=2);
    ax.plot(x_lgg[keep_lgg], y_lgg[keep_lgg], color=colour, marker=marker, markersize=8,
            linestyle="", label=f"label {label_lgg}");
ax.set_xticks(x_lgg);
ax.set_xlabel("Predictor");
ax.set_ylabel("Label");
```

Theorem 6.3 (Hessian of the log loss). *The Hessian of the log loss of Definition 5.1 with respect to $\vec{\beta}$ is*

$$\begin{aligned}\nabla^2 C(\vec{\beta}) &= \sum_{i=1}^n \pi_i(1 - \pi_i) x_i x_i^\top \\ &= D^\top W D,\end{aligned}$$

where W is the diagonal matrix with entries $\pi_i(1 - \pi_i)$. Because each $\pi_i(1 - \pi_i) > 0$, $D^\top W D$ is positive semidefinite for every $\vec{\beta}$, so $C(\vec{\beta})$ is convex^a and therefore a friendly loss.

^a<https://morrison-lab.github.io/mds/algebra-sums.html#def-convex-function>

i Proof

Proof. By Theorem 6.2, $\nabla C(\vec{\beta}) = \sum_{i=1}^n (\pi_i - y_i) x_i$. Only π_i depends on $\vec{\beta}$, and by Theorem 2.2 and the chain rule its gradient is $\pi_i(1 - \pi_i) x_i$. Differentiating each term of the sum gives $\pi_i(1 - \pi_i) x_i x_i^\top$, and the sum of these is $D^\top W D$. $\square$

Example 6.4 (The Hessian at zero coefficients). For the six points of Example 6.3 at $\vec{\beta} = \vec{0}$, every $\pi_i = 1/2$, so $W = \frac{1}{4}I$.

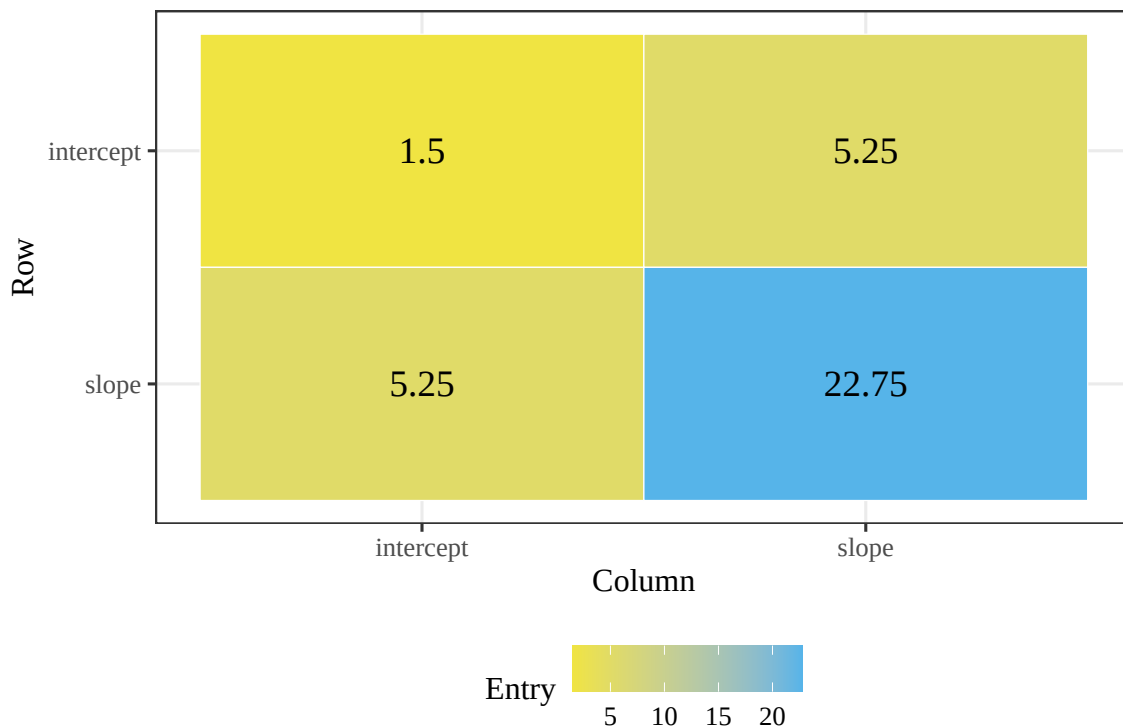
```
# the six points, with the column of ones for the intercept
x <- 1:6
design <- cbind(1, x)
# fitted probabilities at zero coefficients, and their weights p (1 - p)
p <- rep(1 / 2, 6)
w <- p * (1 - p)
# the Hessian: design' W design
crossprod(design, w * design)
```

```
      x
      1.50  5.25
x 5.25 22.75
```

The intercept entry is $\frac{1}{4} \cdot 6 = 1.5$, the off-diagonal entry is $\frac{1}{4} \sum_i x_i = 5.25$, and the slope entry is $\frac{1}{4} \sum_i x_i^2 = 22.75$.

6.7 R

```
library(ggplot2)
hsf_x <- 1:6
hsf_design <- cbind(1, hsf_x)
hsf_h <- crossprod(hsf_design, (1 / 4) * hsf_design)
hsf_cells <- expand.grid(row = c("intercept", "slope"), col = c("intercept", "slope"))
hsf_cells$value <- as.vector(hsf_h)
ggplot(hsf_cells, aes(col, row, fill = value)) +
  geom_tile(colour = "white") +
  geom_text(aes(label = value), colour = "black", size = 5) +
  scale_fill_gradient(low = "#F0E442", high = "#56B4E9") +
  scale_y_discrete(limits = c("slope", "intercept")) +
  labs(x = "Column", y = "Row", fill = "Entry")
```



(a)

6.8 Python

```
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap
import numpy as np

x_hsf = np.arange(1, 7)
design_hsf = np.column_stack([np.ones(6), x_hsf])
h_hsf = design_hsf.T @ (0.25 * design_hsf)
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
im = ax.imshow(h_hsf, cmap=LinearSegmentedColormap.from_list("hsf", ["#F0E442", "#56B4E9"]))
for i in range(2):
    for j in range(2):
        ax.text(j, i, f"{h_hsf[i, j]:g}", ha="center", va="center", fontsize=14);
ax.set_xticks([0, 1], ["intercept", "slope"]);
ax.set_yticks([0, 1], ["intercept", "slope"]);
ax.set_xlabel("Column");
ax.set_ylabel("Row");
fig.colorbar(im, ax=ax, label="Entry");
plt.show()
```

Theorem 6.4 (The residuals are orthogonal to every predictor at a minimizer). *At any finite minimizer $\hat{\beta}$ of the log loss, the gradient vanishes:*

$$D^{\top}(\tilde{\pi} - y) = \vec{0}.$$

Each column of D is orthogonal to the residuals, and the column of ones makes the residuals sum to zero.

i Proof

Proof. The log loss is differentiable and $\hat{\beta}$ minimizes it over all coefficient vectors, so its gradient is zero there. By Theorem 6.2, that gradient is $D^{\top}(\tilde{\pi} - y)$. The j th entry of this vector is the dot product of the j th column of D with the residuals, and the entry for the column of ones is $\sum_{i=1}^n (\pi_i - y_i)$. $\square$

Example 6.5 (Residuals at the fitted coefficients). Fit a logistic regression with an intercept to the six points of Example 6.3, and compute $D^{\top}(\tilde{\pi} - y)$.

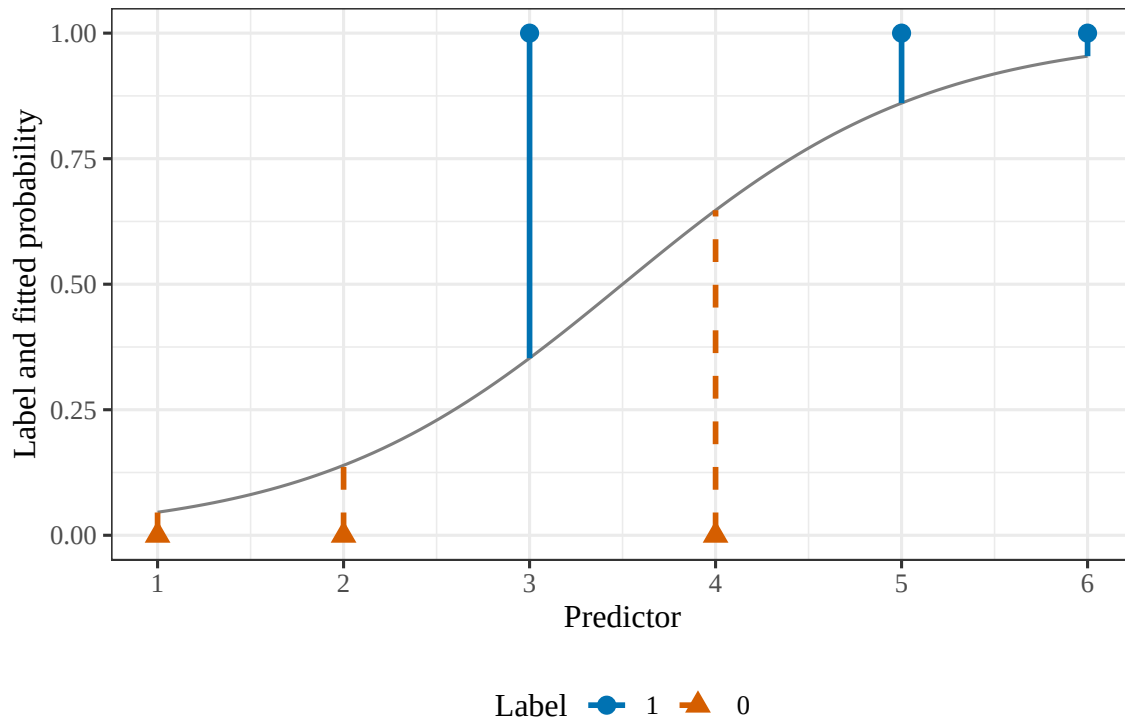
```
# the six points
x <- 1:6
y <- c(0, 0, 1, 0, 1, 1)
# the fit, and its design matrix with the column of ones
fit <- glm(y ~ x, family = binomial)
design <- model.matrix(fit)
# the residuals: fitted probabilities minus labels
residual <- fitted(fit) - y
# the gradient, rounded to six decimals
round(drop(crossprod(design, residual)), 6)
```

```
(Intercept)      x
          0          0
```

Both entries are zero to six decimals, as Theorem 6.4 requires.

6.9 R

```
library(ggplot2)
lst_pts <- data.frame(
  x = x,
  label = factor(y, levels = c(1, 0)),
  y = y,
  fitted = fitted(fit)
)
lst_curve <- data.frame(x = seq(1, 6, length.out = 100))
lst_curve$prob <- predict(fit, newdata = lst_curve, type = "response")
ggplot(lst_pts, aes(x, y, colour = label)) +
  geom_line(data = lst_curve, aes(y = prob), colour = "grey50") +
  geom_segment(aes(xend = x, yend = fitted, linetype = label), linewidth = 1) +
  geom_point(aes(shape = label), size = 3) +
  scale_colour_manual(values = c("1" = "#0072B2", "0" = "#D55E00")) +
  scale_shape_manual(values = c("1" = 16, "0" = 17)) +
  scale_linetype_manual(values = c("1" = "solid", "0" = "dashed")) +
  scale_x_continuous(breaks = 1:6) +
  labs(x = "Predictor", y = "Label and fitted probability", colour = "Label", shape = "Label",
       linetype = "Label")
```



(a)

6.10 Python

```
import matplotlib.pyplot as plt

points_lst = r.lst_pts
curve_lst = r.lst_curve
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.plot(curve_lst["x"], curve_lst["prob"], color="grey");
for label_lst, colour, marker, style in [(1, "#0072B2", "o", "-"), (0, "#D55E00", "^", "--)]:
    keep_lst = points_lst["y"] == label_lst
    sub_lst = points_lst[keep_lst]
    ax.vlines(sub_lst["x"], sub_lst["fitted"], sub_lst["y"], color=colour, linestyle=style,
              linewidth=2);
    ax.plot(sub_lst["x"], sub_lst["y"], color=colour, marker=marker, markersize=8,
```

Exercise 6.6 (Fitting by gradient descent, in code). Simulate $n = 400$ observations with two standard normal features x_1, x_2 and labels drawn from $\Pr[Y = 1 \mid x] = \text{expit}(-0.5 + 2x_1 - 1.5x_2)$, so the coefficients that produced the data are known to be $\vec{\beta} = (-0.5, 2, -1.5)$.

In Python, R or Julia, minimize the *mean^a log loss* $J(\vec{\beta}) = \frac{1}{n} \sum_i C_i$ by gradient descent^b from $\vec{\beta} = \vec{0}$, with step size $\alpha = 0.5$ for 10,000 steps, using the gradient of Exercise 6.2. Say how that gradient's sign and scale relate to the $D(\mathbf{x})^\top(y - \pi)$ of Exercise 5.3's log-likelihood^c.

^a<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

^b<https://morrison-lab.github.io/mds/optimization.html#def-gradient-descent>

^c<https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik>

Solution

Solution 6.6. The gradient of J is $X^\top(\pi - y)/n$, which is the log-likelihood's $X^\top(y - \pi)$ with its sign flipped, because we are descending the negative log-likelihood rather than ascending the log-likelihood, and divided by n . Dividing by n changes no minimizer, as Exercise 5.10 showed, and makes a sensible step size independent of how much data there is.

6.11 Python

```
import numpy as np

sigmoid = lambda z: 0.5 * (1 + np.tanh(0.5 * z))

rng = np.random.default_rng(0)
n = 400
x1, x2 = rng.normal(size=n), rng.normal(size=n)
X = np.column_stack([np.ones(n), x1, x2])
y = (rng.random(n) < sigmoid(-0.5 + 2 * x1 - 1.5 * x2)).astype(float)

def J(b):
    p = sigmoid(X @ b)
    return -np.mean(y * np.log(p) + (1 - y) * np.log(1 - p))

def grad_J(b):
    return X.T @ (sigmoid(X @ b) - y) / n

beta = np.zeros(3)
for _ in range(10_000):
    beta = beta - 0.5 * grad_J(beta)

print(beta.round(2))          # true values: [-0.5, 2, -1.5]
[-0.54  1.95 -1.18]

print(J(beta).round(4))
0.4294
```

6.12 R

```

sigmoid <- function(z) 0.5 * (1 + tanh(0.5 * z))

set.seed(0)
n <- 400
x1 <- rnorm(n)
x2 <- rnorm(n)
X <- cbind(1, x1, x2) # nolint: object_name_linter. X matches the prose.
y <- as.numeric(runif(n) < sigmoid(-0.5 + 2 * x1 - 1.5 * x2))

J <- function(b) { # nolint: object_name_linter.
  p <- sigmoid(drop(X %*% b))
  -mean(y * log(p) + (1 - y) * log(1 - p))
}

grad_J <- function(b) drop(t(X) %*% (sigmoid(drop(X %*% b)) - y) / n) # nolint

beta <- c(0, 0, 0)
for (step in seq_len(10000)) beta <- beta - 0.5 * grad_J(beta)

round(beta, 2) # true values: c(-0.5, 2, -1.5)

      x1      x2
-0.53  1.98 -1.46
round(J(beta), 4)
[1] 0.4201

```

6.13 Julia

```

using Random, LinearAlgebra, Statistics

sigmoid(z) = 0.5 .* (1 .+ tanh.(0.5 .* z));

Random.seed!(0);
n = 400;
x1, x2 = randn(n), randn(n);
X = [ones(n) x1 x2];
y = Float64.(rand(n) .< sigmoid(-0.5 .+ 2 .* x1 .- 1.5 .* x2));

function J(b)
  p = sigmoid(X * b)
  -mean(y .* log.(p) .+ (1 .- y) .* log.(1 .- p))
end;

grad_J(b) = X' * (sigmoid(X * b) .- y) ./ n;

beta = zeros(3);
for _ in 1:10_000
  global beta = beta - 0.5 .* grad_J(beta)
end

println(round.(beta, digits = 2)) # true values: [-0.5, 2, -1.5]
[-0.48, 1.97, -1.62]
println(round(J(beta), digits = 4))
0.4069

```

```
# @sol-fit-by-gradient-descent-part2: the Python estimate, its standard error, and the R estimate
py_beta <- py$beta
py_grad <- max(abs(py$grad_J(py_beta)))
py_fit <- glm(py$y ~ py$X[, 2] + py$X[, 3], family = binomial)
py_se <- summary(py_fit)$coefficients[3, "Std. Error"]
py_z <- abs(py_beta[3] - (-1.5)) / py_se
r_beta <- beta

# @sol-fit-by-gradient-descent-part3: the two sigmoid forms in double precision
sig_tanh <- function(z) 0.5 * (1 + tanh(0.5 * z))
sig_exp <- function(z) 1 / (1 + exp(-z))
far <- seq(-60, 0, by = 0.01)
tanh_far <- sig_tanh(far)
tanh_floor <- min(tanh_far[tanh_far > 0])
tanh_zero_from <- max(far[tanh_far == 0])
exp_at_700 <- sig_exp(-700)
near <- seq(-10, 10, by = 0.001)
abs_gap <- max(abs(sig_tanh(near) - sig_exp(near)))
rel_gap <- abs(sig_tanh(near) - sig_exp(near)) / sig_exp(near)
worst_z <- near[which.max(rel_gap)]
worst_p <- sig_exp(worst_z)
worst_ulps <- abs(sig_tanh(worst_z) - worst_p) / 2^(floor(log2(worst_p)) - 52)
fmt <- function(v, d = 2) formatC(v, format = "e", digits = d)
```

Exercise 6.7 (Fitting by gradient descent, in code (part 2)). Use the setup of Exercise 6.6. Compare the fitted coefficients with $(-0.5, 2, -1.5)$. If one of them differs from the true value by more than you expected^a, determine whether the optimizer terminated before converging or whether the difference comes from another source, and explain how to check.

^a<https://morrison-lab.github.io/pds/expectation.html#def-expectation>

Solution

Solution 6.7. The Python version recovers $(-0.54, 1.95, -1.18)$ against the $(-0.5, 2, -1.5)$ from which the data were generated. The iteration has converged: the largest entry of the gradient at the final iterate is $2.22\text{e-}16$. That value -1.18 is the maximum likelihood^a estimate^b for this sample of 400 points. The discrepancy from -1.5 is sampling error: the standard error^c on that coefficient is 0.164, so -1.18 is 1.96 standard errors^d from -1.5 . The R tab illustrates this with an independent^e sample: it draws 400 observations from R's random number generator and estimates^f -1.46 for the same coefficient. The difference in the Python estimate comes from the random sample rather than the optimizer.

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle>

^b<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

^c<https://morrison-lab.github.io/sds/estimation.html#def-SE>

^d<https://morrison-lab.github.io/sds/estimation.html#def-SE>

^e<https://morrison-lab.github.io/pds/independence.html#def-indpt>

^f<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

Exercise 6.8 (Fitting by gradient descent, in code (part 3)). Use the setup of Exercise 6.6. Express the sigmoid^a as $\frac{1}{2}(1 + \tanh(\eta/2))$ rather than $1/(1 + e^{-\eta})$. Exercise 2.5 shows that these are the same mathematical function. State whether they are the same floating-point computation.

^a<https://morrison-lab.github.io/mds/functions.html#def-logistic-function>

Solution

Solution 6.8. They are not the same floating-point computation.

Exercise 6.9 (Fitting by gradient descent, in code (part 4)). Use the setup of Exercise 6.6. Compare their numerical behavior for large negative η .

Solution

Solution 6.9. The `tanh` form loses relative accuracy for negative η much earlier than the exponential form.

The exponential form evaluates $e^{-\eta}$, which overflows to $+\infty$ below about $\eta = -709$; dividing 1 by $+\infty$ still returns 0. The `tanh` form never overflows, but loses relative accuracy for negative η . Adding 1 to $\tanh(\eta/2)$ rounds onto the floating-point grid near 1, so the smallest positive value it can return is $5.55\text{e-}17$, and for $\eta \leq -38.13$ it returns 0. By contrast, the exponential form evaluates to $9.9\text{e-}305$ at $\eta = -700$.

In a loss function that evaluates $\log(p)$, rounding p to 0 turns -700 into $-\infty$. That underflow does not occur in this simulation, where $|\eta|$ remains below 10, though the two forms still differ in relative error. In *absolute* terms the two forms differ by at most $2.22\text{e-}16$ over $|\eta| \leq 10$. In *relative* terms, which govern the accuracy of $\log(p)$, the difference grows as p approaches 0: the maximum relative difference over $|\eta| \leq 10$ is $5.81\text{e-}13$, at $\eta = -9.987$, where p is $4.60\text{e-}05$ and the two floating-point values differ by 3941 units in the last place (ulps). This loss of significant digits comes from catastrophic cancellation when adding 1 to $\tanh(\eta/2) \approx -1$, though it leaves more than enough digits over $|\eta| \leq 10$. For extreme negative scores, neither intermediate probability should be formed: evaluating the log-likelihood^a directly from η (for example, via $\eta - \log(1 + e^\eta)$ for $\eta < 0$) avoids forming p altogether.

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik>

Exercise 6.10 (Fitting by gradient descent, in code (part 5)). Use the setup of Exercise 6.6. Name the library call you would use instead in each language, and check whether it fits the same objective by default.

Solution

Solution 6.10. In practice you would call a library: `glm(y ~ x1 + x2, family = binomial)` in R, or `LogisticRegression` from scikit-learn in Python. Beware the Python one: scikit-learn applies an ℓ_2 penalty by default, so it fits the regularized problem of Exercise 5.19 and will not reproduce these numbers until you turn the penalty off.

7 A fit small enough to do by hand

For a balanced dataset of six patients in two groups, the maximum likelihood⁵² estimate⁵³ has a closed-form solution. Solving this six-patient problem in full provides an exact benchmark for testing step sizes, Newton's method, and standard errors⁵⁴ in the exercises that follow.

Exercise 7.1 (A logistic regression you can fit by hand). We randomize six patients to control or treatment, and record whether each recovers. Code the group as $x_i = -1$ for control and $x_i = +1$ for treatment, and let $y_i = 1$ mean recovery.

⁵²<https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle>

⁵³<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

⁵⁴<https://morrison-lab.github.io/sds/estimation.html#def-SE>

group	x	n	recovered
control	-1	3	1
treatment	+1	3	2

Fit $\Pr[Y = 1 \mid x] = \text{expit}(\beta_0 + \beta_1 x)$.

Give the design matrix $D(\mathbf{x})$, and evaluate $\nabla_{\vec{\beta}} \ell = D(\mathbf{x})^\top (\mathbf{y} - \boldsymbol{\pi})$ at $\vec{\beta} = \vec{0}$. Interpret each of its two components.

Solution

Solution 7.1. Order the patients control-first, and take the three control outcomes to be $y = 0, 0, 1$ and the three treatment outcomes $y = 0, 1, 1$:

$$X = \begin{bmatrix} 1 & -1 \\ 1 & -1 \\ 1 & -1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

At $\vec{\beta} = \vec{0}$ every $\pi_i = \text{expit}(0) = 1/2$, so $\mathbf{y} - \boldsymbol{\pi} = (-\frac{1}{2}, -\frac{1}{2}, \frac{1}{2}, -\frac{1}{2}, \frac{1}{2}, \frac{1}{2})^\top$ and

$$X^\top (\mathbf{y} - \boldsymbol{\pi}) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

The **intercept** component is $\sum_i (y_i - \pi_i) = 3 - 6 \times \frac{1}{2} = 0$. Three of six patients recovered, and the model at $\vec{\beta} = \vec{0}$ already predicts three, so there is nothing to correct: the starting point happens to be calibrated in aggregate, in the sense of Exercise 6.5.

The **slope** component is $\sum_i x_i (y_i - \pi_i) = 1$, and it is positive. That says the residual is correlated with the feature: the group coded +1 has more recoveries than the current model predicts and the group coded -1 has fewer, so increasing β_1 will improve the fit. The gradient already shows the treatment effect before anything has been fitted.

Exercise 7.2 (A logistic regression you can fit by hand (part 2)). Use the setup of Exercise 7.1.

Give the **log loss** at $\vec{\beta} = \vec{0}$, exactly and to four significant figures. Why is this the value for *any* dataset of six observations at $\vec{\beta} = \vec{0}$?

Solution

Solution 7.2. Every $\pi_i = 1/2$, so every observation contributes $-\log \frac{1}{2} = \log 2$ whatever its label, and

$$\begin{aligned} -\ell(\vec{0}) &= 6 \log 2 \\ &= 4.159 \end{aligned}$$

So the value does not depend on the data. At $\vec{\beta} = \vec{0}$ the model predicts 1/2 for everyone, so it scores $\log 2$ per observation regardless of the outcomes, and the total is $n \log 2$. That makes $n \log 2$ the natural reference point: a model that knows nothing achieves it, and we measure the fitted model's improvement in Solution 7.6 against it.

Exercise 7.3 (A logistic regression you can fit by hand (part 3)). Use the setup of Exercise 7.1.

Argue from the shape of the data that $\hat{\beta}_0 = 0$, then use the score equation for β_1 to find $\hat{\beta}_1$ exactly.

Solution

Solution 7.3. The two groups have equal size and symmetric^a coding $x = \pm 1$. Let $\pi_+ = \text{expit}(\beta_0 + \beta_1)$ and $\pi_- = \text{expit}(\beta_0 - \beta_1)$. The intercept^b score equation is $3\pi_- + 3\pi_+ = 1 + 2 = 3$, which simplifies to $\pi_- + \pi_+ = 1$. By Exercise 2.2, $\pi_- = 1 - \pi_+$ holds if and only if $\beta_0 - \beta_1 = -(\beta_0 + \beta_1)$, which implies $\hat{\beta}_0 = 0$.

With $\hat{\beta}_0 = 0$ we have $\pi_- = 1 - \pi_+$. The slope^c score equation is

$$\begin{aligned}\sum_i x_i(y_i - \pi_i) &= -(1 - 3\pi_-) + (2 - 3\pi_+) \\ &= 0 \\ &\implies 1 + 3\pi_- - 3\pi_+ = 0\end{aligned}$$

Substituting $\pi_- = 1 - \pi_+$ gives $1 + 3 - 6\pi_+ = 0$, so $\pi_+ = \frac{2}{3}$ and $\pi_- = \frac{1}{3}$. Then

$$\begin{aligned}\hat{\beta}_1 &= \text{logit}\left(\frac{2}{3}\right) \\ &= \log \frac{2/3}{1/3} \\ &= \log 2 \\ &= 0.6931\end{aligned}$$

^a<https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-symmetric-matrix>

^b<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

^c<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope>

Exercise 7.4 (A logistic regression you can fit by hand (part 4)). Use the setup of Exercise 7.1. Give the fitted probabilities, and verify both score equations.

Solution

Solution 7.4. The fitted probability vector is $\hat{\pi} = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{2}{3}, \frac{2}{3}, \frac{2}{3})$.

Intercept^a: $\sum_i \pi_i = 3 \times \frac{1}{3} + 3 \times \frac{2}{3} = 1 + 2 = 3 = \sum_i y_i$.

Slope^b: $\sum_i x_i \pi_i = -1 + 2 = 1$ and $\sum_i x_i y_i = -1 + 2 = 1$, so their difference is zero.

Here $\pi_- = 1/3$ equals the proportion of recoveries in the control group (1 of 3), and $\pi_+ = 2/3$ equals the proportion of recoveries in the treatment group (2 of 3): **the fitted probabilities equal the observed group proportions**. With one binary predictor and an intercept, the two score equations require the sum of fitted probabilities to match the observed count in each group, and because there are two parameters for two groups, the model has as many free parameters as group proportions and reproduces them exactly.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

^b<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope>

Exercise 7.5 (A logistic regression you can fit by hand (part 5)). Use the setup of Exercise 7.1. The treatment group's odds^a of recovery are 2 : 1 and the control group's are 1 : 2, so the odds ratio^b between the groups is 4. But $e^{\hat{\beta}_1} = 2$. Explain the difference between $e^{\hat{\beta}_1}$ and the odds ratio, and say what $\hat{\beta}_1$ would have been under the coding $x \in \{0, 1\}$.

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-odds>

^b<https://morrison-lab.github.io/sds/categorical-tests.html#def-odds-ratio>

Solution

Solution 7.5. The slope^a β_1 is the change in log-odds^b per *one-unit* increase in x , and the two groups are separated by two units under the $x \in \{-1, +1\}$ coding. The treatment-versus-control odds ratio^c is therefore $e^{2\hat{\beta}_1} = e^{2\log 2} = 4$, whereas $e^{\hat{\beta}_1} = 2$ is the odds ratio for a one-unit increase from $x = -1$ to $x = 0$ (or from $x = 0$ to $x = +1$).

Under the indicator coding $x \in \{0, 1\}$, the two groups are separated by one unit, so $\hat{\beta}_1 = \log 4 = 1.386$ and $e^{\hat{\beta}_1} = 4$ is directly the between-group odds ratio. The intercept^d also changes to $\text{logit}(1/3) = -\log 2$, so that the fitted probability for the control group ($x = 0$) remains $1/3$.

The fitted probabilities are identical under both codings; only the parameterization differs. Encoding a two-level predictor as a $\{0, 1\}$ indicator makes $e^{\hat{\beta}_1}$ equal the between-group odds ratio directly.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope>

^b<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

^c<https://morrison-lab.github.io/sds/categorical-tests.html#def-odds-ratio>

^d<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

Exercise 7.6 (A logistic regression you can fit by hand (part 6)). Use the setup of Exercise 7.1.

Give the log loss at $\hat{\beta}$, and use the drop from Exercise 7.2 to test $H_0 : \beta_1 = 0$ with a likelihood^a ratio test. Say what the result implies for the trial.

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik>

Solution

Solution 7.6. Four observations have outcomes matching the majority in their group (two treated patients who recovered and two control patients who did not) and contribute $-\log \frac{2}{3}$ each, while the remaining two observations contribute $-\log \frac{1}{3}$ each:

$$\begin{aligned} -\ell(\hat{\beta}) &= 4 \log \frac{2}{3} + 2 \log 3 \\ &= 6 \log 3 - 4 \log 2 \\ &= 3.819 \end{aligned}$$

The reduction in negative log-likelihood^a from Solution 7.2 is $6 \log 2 - (6 \log 3 - 4 \log 2) = 10 \log 2 - 6 \log 3 = 0.3398$, so the likelihood^b ratio between the fitted model and the null model is $e^{0.3398} = 1.405$. The likelihood ratio test statistic is twice the change in log-likelihood, $G^2 = 2 \times 0.3398 = 0.6796$, which against the χ^2_1 distribution^c gives $p = 0.41$.

The six-patient trial therefore provides very little statistical evidence against the null hypothesis^d. Although a fourfold odds ratio^e ($e^{2\hat{\beta}_1} = 4$) is a large point estimate^f, six observations are far too few to distinguish it from random variation: the estimate $\hat{\beta}_1 = 0.693$ has a standard error^g of $\sqrt{3}/2 = 0.866$ — larger than the estimate itself — because the observed information matrix is $D(\mathbf{x})^\top W D(\mathbf{x}) = \frac{2}{9} D(\mathbf{x})^\top D(\mathbf{x}) = \frac{4}{3} I$, with $W_{ii} = \frac{1}{3} \times \frac{2}{3} = \frac{2}{9}$ for every observation.

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik>

^b<https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik>

^c<https://morrison-lab.github.io/pds/random-variables.html#def-cdf>

^d<https://morrison-lab.github.io/sds/inference.html#def-null-hypothesis>

^e<https://morrison-lab.github.io/sds/categorical-tests.html#def-odds-ratio>

^f<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

^g<https://morrison-lab.github.io/sds/estimation.html#def-SE>

Exercise 7.7 (Choosing the step size for the fit you just did by hand). Exercise 7.1 has an exact solution, $\hat{\beta} = (0, \log 2)$, so we can compute the exact error of gradient ascent^a at every step. Run gradient ascent on the log-likelihood,

$$\vec{\beta}_{t+1} = \vec{\beta}_t + \alpha X^\top (y - \tilde{\pi}_t),$$

starting from $\vec{\beta}_0 = \vec{0}$.

Show that β_0 stays at 0 for every step, whatever α is, so the iteration is one-dimensional.

^a<https://morrison-lab.github.io/mds/optimization.html#def-gradient-ascent>

Solution

Solution 7.7. The intercept column of X is all ones, so the first component of the gradient is $\sum_i (y_i - \pi_i)$. Suppose $\beta_0 = 0$ at some step. Then $z_i = \beta_1 x_i$, so the three controls all have $\pi = \text{expit}(-\beta_1)$ and the three treated all have $\pi = \text{expit}(\beta_1)$, and Exercise 2.2 gives $\text{expit}(-\beta_1) + \text{expit}(\beta_1) = 1$. So $\sum_i \pi_i = 3$, while

$$\begin{aligned} \sum_i y_i &= 1 + 2 \\ &= 3, \end{aligned}$$

and the component is exactly zero. The intercept therefore does not move; since it starts at 0, it stays there for every step and every α .

This is the same symmetry Exercise 7.3 used to argue $\hat{\beta}_0 = 0$, now used for the whole trajectory rather than just the endpoint.

Exercise 7.8 (Choosing the step size for the fit you just did by hand (part 2)). Use the setup of Exercise 7.7.

Show that the remaining component of the gradient is

$$g(\beta_1) = 4 - 6 \text{expit}(\beta_1),$$

Solution

Solution 7.8. The second component is $\sum_i x_i (y_i - \pi_i)$, which with the same substitution is

$$\begin{aligned} -[1 - 3 \text{expit}(-\beta_1)] + [2 - 3 \text{expit}(\beta_1)] &= 1 + 3[\text{expit}(-\beta_1) - \text{expit}(\beta_1)] \\ &= 1 + 3[1 - 2 \text{expit}(\beta_1)] \\ &= 4 - 6 \text{expit}(\beta_1). \end{aligned}$$

Exercise 7.9 (Choosing the step size for the fit you just did by hand (part 3)). Use the setup of Exercise 7.7.

Check it against the value Exercise 7.1 found at $\vec{\beta} = \vec{0}$.

Solution

Solution 7.9. At $\beta_1 = 0$ this is $4 - 3 = 1$, which is what Exercise 7.1 found, and at $\beta_1 = \log 2$ it is $4 - 4 = 0$, which is the score equation. The whole run is the scalar iteration $\beta_1 \leftarrow \beta_1 + \alpha[4 - 6 \text{expit}(\beta_1)]$.

Exercise 7.10 (Choosing the step size for the fit you just did by hand (part 4)). Use the setup of Exercise 7.7.

Set $\alpha = \frac{1}{2}$ and give β_1 after each of five steps, to six decimal places. Tabulate the error $\beta_1 - \log 2$ alongside, and the ratio of successive errors. What value does that ratio approach?

Solution

Solution 7.10. With $\alpha = \frac{1}{2}$:

step	β_1	$\beta_1 - \log 2$	ratio to previous
1	0.500000	-0.193147	—
2	0.632622	-0.060525	0.3134
3	0.673371	-0.019776	0.3267
4	0.686598	-0.006549	0.3312
5	0.690969	-0.002178	0.3326

The error keeps its sign — every iterate approaches $\log 2 = 0.693147$ from below — and shrinks by a factor approaching $1/3$. [Solution 7.13](#) derives that factor of $1/3$.

Exercise 7.11 (Choosing the step size for the fit you just did by hand (part 5)). Use the setup of [Exercise 7.7](#).

Now set $\alpha = 2$ and give five steps.

Solution

Solution 7.11. With $\alpha = 2$, five steps give

$$2.000000, \quad -0.569565, \quad 3.094388, \quad -0.385540, \quad 2.756962.$$

Exercise 7.12 (Choosing the step size for the fit you just did by hand (part 6)). Use the setup of [Exercise 7.7](#).

Does the sequence of iterates diverge? Prove your answer, using the bound on $\|\nabla C_i\|$ from [Exercise 6.4](#).

Solution

Solution 7.12. It overshoots the maximizer at each step, but it does **not** diverge.

[Exercise 6.4](#) bounds each observation's gradient by $\|\tilde{x}_i\|$, and here that makes the scalar gradient bounded: $\text{expit} \in (0, 1)$ strictly, so

$$g(\beta_1) = 4 - 6 \text{expit}(\beta_1) \in (-2, 4)$$

for every $\beta_1 \in \mathbb{R}$. A single step therefore changes β_1 by more than -2α and less than 4α , whatever the current value.

That bound confines the iterates to a compact interval. Let $m = \log 2$ be the fixed point, let $J = [m - 2\alpha, m + 4\alpha]$, and consider any $\beta_1 \in J$. If $\beta_1 \geq m$ then $\text{expit}(\beta_1) \geq \frac{2}{3}$, so $g(\beta_1) \leq 0$ and the step is downward: the new value is at most $\beta_1 \leq m + 4\alpha$ and at least $\beta_1 - 2\alpha \geq m - 2\alpha$. If instead $\beta_1 \leq m$ then $g(\beta_1) \geq 0$, and the two inequalities hold in the opposite direction: the new value is at least $\beta_1 \geq m - 2\alpha$ and at most $\beta_1 + 4\alpha \leq m + 4\alpha$. In both cases the next iterate is in J .

We start at $\beta_1 = 0$, which is in J whenever $2\alpha \geq \log 2$, so for $\alpha = 2$ the sequence remains in $[-3.307, 8.693]$ for all t . All five iterates for $\alpha = 2$ are inside this interval.

Exercise 7.13 (Choosing the step size for the fit you just did by hand (part 7)). Use the setup of [Exercise 7.7](#).

Linearize the update map of [Exercise 7.7](#) about $\hat{\beta}_1 = \log 2$ and find its derivative there.

Solution

Solution 7.13. Let the update map be $T(\beta_1) = \beta_1 + \alpha[4 - 6 \text{expit}(\beta_1)]$. Then

$$\begin{aligned} T'(\beta_1) &= 1 - 6\alpha \text{expit}'(\beta_1) \\ &= 1 - 6\alpha \text{expit}(\beta_1)[1 - \text{expit}(\beta_1)], \end{aligned}$$

using $\text{expit}' = \text{expit}(1 - \text{expit})$ from Exercise 2.3. At the fixed point $\text{expit} = \frac{2}{3}$, so

$$\begin{aligned} \text{expit}' &= \frac{2}{3} \cdot \frac{1}{3} \\ &= \frac{2}{9} \end{aligned}$$

and

$$\begin{aligned} T'(\log 2) &= 1 - 6\alpha \cdot \frac{2}{9} \\ &= 1 - \frac{4}{3}\alpha. \end{aligned}$$

Exercise 7.14 (Choosing the step size for the fit you just did by hand (part 8)). Use the setup of Exercise 7.7.

For which α does the iteration converge?

Solution

Solution 7.14. Near $\log 2$ the error is multiplied by $1 - \frac{4}{3}\alpha$ at each step, so the iteration converges locally when $|1 - \frac{4}{3}\alpha| < 1$, which holds when

$$0 < \alpha < \frac{3}{2}.$$

Exercise 7.15 (Choosing the step size for the fit you just did by hand (part 9)). Use the setup of Exercise 7.7.

Check that your answer accounts for both Exercise 7.10 and Exercise 7.11.

Solution

Solution 7.15. Both earlier parts follow from this derivative. At $\alpha = \frac{1}{2}$ the multiplier is $1 - \frac{2}{3} = \frac{1}{3}$: positive, so the error keeps its sign, and equal to $1/3$, matching the ratio observed in Solution 7.10. At $\alpha = 2$ it is $1 - \frac{8}{3} = -\frac{5}{3}$: negative, so the error alternates in sign at every step, and larger than 1 in magnitude, so the fixed point is unstable. Together with Solution 7.12, this determines the behaviour at $\alpha = 2$: the sequence can neither converge to $\log 2$ nor leave J .

Exercise 7.16 (Choosing the step size for the fit you just did by hand (part 10)). Use the setup of Exercise 7.7.

The step-size bound in Exercise 6.4 gives $\alpha < 8/\lambda_{\max}(D(\mathbf{x})^\top D(\mathbf{x}))$. Evaluate it here, compare it with the exact threshold from Exercise 7.14, and account for the difference between the two numbers.

Solution

Solution 7.16. $D(\mathbf{x})$ has an intercept^a column of ones and an x column of $(-1, -1, -1, +1, +1, +1)$, which are orthogonal^b, so

$$X^\top X = \begin{pmatrix} 6 & 0 \\ 0 & 6 \end{pmatrix}, \quad \lambda_{\max}(X^\top X) = 6,$$

and the bound gives $\alpha < 8/6 = 4/3$. The exact threshold is $3/2$, so the bound is conservative by a factor of $\frac{3}{2} \div \frac{4}{3} = \frac{9}{8}$.

The difference comes from the inequality $W_{ii} \leq 1/4$. Equality holds only where $\pi_i = 1/2$, whereas at the maximum likelihood^c estimate^d the fitted probabilities are $1/3$ and $2/3$, so every $W_{ii} = \frac{1}{3} \cdot \frac{2}{3} = \frac{2}{9}$. The Hessian^e at the maximum is therefore $\frac{2}{9}X^\top X = \frac{4}{3}I$ rather than $\frac{1}{4}X^\top X = \frac{3}{2}I$, and $2/\lambda_{\max}$ of the Hessian at the optimum is $3/2$. The ratio $\frac{9}{8}$ equals $\frac{1}{4} \div \frac{2}{9}$.

The bound is a worst case over *all* $\vec{\beta}$, and it is loose by the factor by which $\pi_i(1 - \pi_i)$ at the optimum falls below $1/4$. A fit that separates the classes well has fitted probabilities near 0 and 1, small curvature at the optimum, and therefore a larger stable step size — which is the saturation of Exercise 2.4 viewed in terms of step size. The same small derivative that slows gradient ascent^f in the tails also prevents overshooting from growing unboundedly.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

^b<https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-orthogonal-vectors>

^c<https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle>

^d<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

^e<https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian>

^f<https://morrison-lab.github.io/mds/optimization.html#def-gradient-ascent>

Exercise 7.17 (Choosing the step size for the fit you just did by hand (part 11)). Use the setup of Exercise 7.7.

Run $\alpha = 2$ for a few thousand steps. Describe its long-run behaviour.

Solution

Solution 7.17. It approaches a **period-2 cycle**: after a few hundred steps β_1 alternates between -0.468955 and 2.912682 . The sequence neither converges to a single point nor grows without bound; it approaches a pair of points that map to each other under T .

The bifurcation occurs at $\alpha = 3/2$, where the multiplier of Solution 7.13 equals -1 . Just above $3/2$ the cycle is narrow — at $\alpha = 1.51$ the two points are 0.458381 and 0.947953 , on either side of $\log 2$ — and it widens as α increases.

Exercise 7.18 (Choosing the step size for the fit you just did by hand (part 12)). Use the setup of Exercise 7.7.

State what the corresponding behaviour would be for the least-squares cost of the gradient descent chapter.

Solution

Solution 7.18. Least squares behaves differently. There the objective is quadratic, so the update is linear, with the error multiplied by $1 - 2\alpha\lambda$ at every step and no higher-order terms. When $|1 - 2\alpha\lambda| > 1$, the error is multiplied by that factor at every step: the sequence diverges geometrically, and the iterates grow without bound.

The logistic loss does not diverge in this way, for the reason shown in Solution 7.12. Because the fitted probabilities are in $(0, 1)$, the gradient is bounded on $\mathbb{R}$, so an oversized step produces a bounded oscillation rather than divergence to infinity. An excessive step size therefore manifests differently in the two problems: least squares overflows, whereas **logistic regression** oscillates between finite values without converging.

8 Newton's method

Gradient ascent⁵⁵ uses the first derivative and requires a user-chosen step size. Newton's method⁵⁶ also uses the second derivative, which determines both the direction and the length of the step. For a model with a modest number of coefficients it converges in a few iterations, and standard statistical software uses it by default.

Exercise 8.1 (Newton's method, and why it is a weighted least squares fit). Exercise 7.7 needed five steps and a chosen step size to reach three decimal places. Newton's method uses the curvature instead, and needs neither.

Expand ℓ to second order about $\vec{\beta}_t$ and maximize the quadratic exactly.

Solution

Solution 8.1. Exercise 6.1 gave $\nabla \ell = X^\top(y - \pi)$ and $\nabla^2 \ell = -X^\top W X$, so the second-order expansion about $\vec{\beta}_t$ is

$$\ell(\vec{\beta}_t + \tilde{d}) \approx \ell(\vec{\beta}_t) + \tilde{d}^\top X^\top(y - \pi_t) - \frac{1}{2} \tilde{d}^\top X^\top W_t X \tilde{d}.$$

This expansion is a concave quadratic in $\tilde{d}$. Differentiating and setting to zero gives $X^\top(y - \pi_t) = X^\top W_t X \tilde{d}$.

Exercise 8.2 (Newton's method, and why it is a weighted least squares fit (part 2)). Use the setup of Exercise 8.1.

Show that the resulting update is

$$\vec{\beta}_{t+1} = \vec{\beta}_t + (X^\top W_t X)^{-1} X^\top(y - \pi_t),$$

with $W_t = \text{diag}(\pi_i(1 - \pi_i))$ as in Exercise 6.4.

Solution

Solution 8.2. Solving for $\tilde{d}$ gives $\tilde{d} = (X^\top W_t X)^{-1} X^\top(y - \pi_t)$, which is the stated update.

Exercise 8.3 (Newton's method, and why it is a weighted least squares fit (part 3)). Use the setup of Exercise 8.1.

Say what is missing compared with gradient ascent^a.

^a<https://morrison-lab.github.io/mds/optimization.html#def-gradient-ascent>

Solution

Solution 8.3. What is missing is α . Gradient ascent^a takes the gradient as a direction and then needs a separate decision about how far to step, which is the subject of Exercise 7.7. Newton's method determines the step length from the curvature: it takes the exact maximizer of the local quadratic, so no tuning parameter remains.

^a<https://morrison-lab.github.io/mds/optimization.html#def-gradient-ascent>

Exercise 8.4 (Newton's method, and why it is a weighted least squares fit (part 4)). Use the setup of Exercise 8.1.

⁵⁵<https://morrison-lab.github.io/mds/optimization.html#def-gradient-ascent>

⁵⁶<https://morrison-lab.github.io/mds/optimization.html#def-newton-method>

Define the **working response** $\tilde{z}_t = D(\mathbf{x})\tilde{\beta}_t + W_t^{-1}(y - \pi_t)$ and show that the update of Exercise 8.1 is exactly

$$\tilde{\beta}_{t+1} = (D(\mathbf{x})^\top W_t D(\mathbf{x}))^{-1} D(\mathbf{x})^\top W_t \tilde{z}_t,$$

the weighted least squares fit of $\tilde{z}_t$ on $D(\mathbf{x})$ with weights W_t . So the algorithm^a is called **iteratively reweighted least squares**. Evaluate $\tilde{z}_0$ for Exercise 7.1's six patients at $\tilde{\beta}_0 = \vec{0}$, and say how to interpret the six numbers.

^a<https://morrison-lab.github.io/mds/expressions-algorithms.html#def-algorithm>

Solution

Solution 8.4. Substituting the definition of $\tilde{z}_t$ and splitting the product gives

$$\begin{aligned} X^\top W_t \tilde{z}_t &= X^\top W_t X \tilde{\beta}_t + X^\top W_t W_t^{-1} (y - \pi_t) \\ &= X^\top W_t X \tilde{\beta}_t + X^\top (y - \pi_t). \end{aligned}$$

Multiplying through by $(D(\mathbf{x})^\top W_t D(\mathbf{x}))^{-1}$ gives $\tilde{\beta}_t + (D(\mathbf{x})^\top W_t D(\mathbf{x}))^{-1} D(\mathbf{x})^\top (y - \pi_t)$, which is the update from Solution 8.2. Every Newton step is therefore a weighted least squares fit of a response that changes at each iteration, with weights that also change at each iteration, as the name iteratively reweighted least squares states.

For Exercise 7.1 at $\tilde{\beta}_0 = \vec{0}$: $X\tilde{\beta}_0 = \vec{0}$, every $\pi_i = \frac{1}{2}$, so $W_0 = \frac{1}{4}I$ and

$$\begin{aligned} \tilde{z}_0 &= \vec{0} + 4(y - \tfrac{1}{2}) \\ &= (2, -2, -2, 2, 2, -2)^\top, \end{aligned}$$

taking the patients in the order (control: recovered, failed, failed; treatment: recovered, recovered, failed).

Those six numbers are on the log-odds^a scale rather than the probability scale. Each value is the current linear prediction plus the **residual** converted to log-odds by dividing by $d\pi/dz = \pi(1 - \pi) = \frac{1}{4}$. It is a linear approximation of the log-odds that would have produced this observation, and fitting a line to those approximations produces the step. Regressing $\tilde{z}_0$ on $D(\mathbf{x})$ with weights $\frac{1}{4}$ returns $(0, \frac{2}{3})$, which is the Newton step that Exercise 7.1 computed.

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

Exercise 8.5 (Newton's method, and why it is a weighted least squares fit (part 5)). Use the setup of Exercise 8.1.

Run four Newton steps on Exercise 7.1 from $\tilde{\beta}_0 = \vec{0}$. Tabulate β_1 and the error $|\beta_1 - \log 2|$.

Solution

Solution 8.5.

step	β_1	error $ \beta_1 - \log 2 $	previous error squared
1	0.666666666667	2.648×10^{-2}	—
2	0.693033399004	1.138×10^{-4}	7.012×10^{-4}
3	0.693147178402	2.157×10^{-9}	1.295×10^{-8}
4	0.693147180560	2.2×10^{-16}	4.655×10^{-18}

The intercept^a stays at zero throughout, by the symmetry argument from Exercise 7.7.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

Exercise 8.6 (Newton’s method, and why it is a weighted least squares fit (part 6)). Use the setup of Exercise 8.1. Compare each error with the *square* of the one before it.

Solution

Solution 8.6. Each error is close to a fixed multiple of the square of the preceding error: $1.138 \times 10^{-4} \div 7.012 \times 10^{-4} = 0.162$, and $2.157 \times 10^{-9} \div 1.295 \times 10^{-8} = 0.167$.

Exercise 8.7 (Newton’s method, and why it is a weighted least squares fit (part 7)). Use the setup of Exercise 8.1. Estimate^a the constant^b relating them.

^a<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

^b<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

Solution

Solution 8.7. The constant^a is approximately 1/6.

The row for step 4 departs from the pattern only because 2.2×10^{-16} is the smallest floating-point spacing near 0.693. The method reached floating-point roundoff at step 3 and could not improve further.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

Exercise 8.8 (Newton’s method, and why it is a weighted least squares fit (part 8)). Use the setup of Exercise 8.1.

Derive that constant^a as a closed-form expression^b. For Newton’s method applied to $g(\beta_1) = 0$, the error satisfies $e_{t+1} \approx |g''/2g'|e_t^2$ near the root. Evaluate it at $\hat{\beta}_1 = \log 2$ using $\text{expit}'' = \text{expit}(1 - \text{expit})(1 - 2 \text{expit})$, and check it against Exercise 8.7.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

^b<https://morrison-lab.github.io/mds/expressions-algorithms.html#def-closed-form-expression>

Solution

Solution 8.8. The score is $g(\beta_1) = 4 - 6 \text{expit}(\beta_1)$, so

$$g'(\beta_1) = -6 \text{expit}(1 - \text{expit}), \quad g''(\beta_1) = -6 \text{expit}(1 - \text{expit})(1 - 2 \text{expit}).$$

At $\hat{\beta}_1 = \log 2$ we have $\text{expit} = \frac{2}{3}$, hence $\text{expit}(1 - \text{expit}) = \frac{2}{9}$ and $1 - 2 \text{expit} = -\frac{1}{3}$, giving $g' = -\frac{4}{3}$ and $g'' = \frac{4}{9}$. So

$$\begin{aligned} \left| \frac{g''}{2g'} \right| &= \frac{4/9}{8/3} \\ &= \frac{1}{6}, \end{aligned}$$

which confirms the value observed in Solution 8.7. As a check, $\frac{1}{6}(1.138 \times 10^{-4})^2 = 2.158 \times 10^{-9}$, compared with the 2.157×10^{-9} observed.

The factor $1 - 2 \text{expit}$ is the only factor in g'' that can vanish, and it vanishes at $\text{expit} = \frac{1}{2}$. A fit whose probabilities are near $\frac{1}{2}$ has a nearly quadratic log-likelihood^a, and Newton’s method solves a nearly quadratic problem in a single step.

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik>

Exercise 8.9 (Newton’s method, and why it is a weighted least squares fit (part 9)). Use the setup of Exercise 8.1.

Compare the two algorithms^a on this problem: give the error after three steps, the error after five steps, and the number of arithmetic operations per step as a function of n and p .

^a<https://morrison-lab.github.io/mds/expressions-algorithms.html#def-algorithm>

Solution

Solution 8.9.

	after 3 steps	after 5 steps	arithmetic per step
gradient ascent ^a , $\alpha = \frac{1}{2}$	2×10^{-2}	2×10^{-3}	$O(np)$
Newton’s method ^b	2×10^{-9}	the floating-point limit	$O(np^2 + p^3)$

Gradient ascent multiplies the error by $1/3$ at each step, so it gains about half a decimal digit per step. Newton’s method squares the error at each step, so the number of correct digits doubles at each step: one, then four, then nine, at which point it reaches the floating-point limit. It also requires no step-size parameter and cannot oscillate from an improperly chosen step size.

Exercise 8.10 (Newton’s method, and why it is a weighted least squares fit (part 10)). Use the setup of Exercise 8.1.

State when Newton’s method^a requires less total computation than gradient ascent^b.

^a<https://morrison-lab.github.io/mds/optimization.html#def-newton-method>

^b<https://morrison-lab.github.io/mds/optimization.html#def-gradient-ascent>

Solution

Solution 8.10. The trade-off is the computation of the Hessian^a. Forming $X^T W X$ takes $O(np^2)$ operations and solving the linear system takes $O(p^3)$ operations, compared with $O(np)$ operations for a gradient step. One Newton step requires roughly as much arithmetic as p gradient steps, so Newton’s method requires less total work whenever it saves more than p steps. For p in the tens or hundreds — as in most regressions in this course — Newton’s method requires far less total work, which is why `glm` uses it.

^a<https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian>

Exercise 8.11 (Newton’s method, and why it is a weighted least squares fit (part 11)). Use the setup of Exercise 8.1.

State why neural networks use first-order methods instead.

Solution

Solution 8.11. Newton’s method becomes infeasible when p is very large. A neural network can have p in the millions, so $X^T W X$ has 10^{12} or more entries and cannot be held in RAM or factorized. The chapter on neural networks therefore returns to first-order methods and to the step-size selection problem that Newton’s method avoids.

Exercise 8.12 (Newton’s method, and why it is a weighted least squares fit (part 12)). Use the setup of Exercise 8.1.

W_t depends on $\vec{\beta}_t$ but not on y . Say what follows for the relationship between the observed information and the Fisher information, and hence for the relationship between Newton’s method and Fisher scoring.

Solution

Solution 8.12. The weight matrix W_t is computed from the fitted probabilities, which depend on $\vec{\beta}_t$ and $D(\mathbf{x})$ alone. No response y_i appears in the Hessian^a $-D(\mathbf{x})^\top W D(\mathbf{x})$.

The Hessian is therefore constant^b with respect to the randomness in Y , so taking its expectation^c over Y leaves it unchanged. The observed information^d and the Fisher information are the *same matrix* at every $\vec{\beta}$ and for every dataset.

Newton's method^e uses the observed information and Fisher scoring uses the expected information^f, so for [logistic regression](#) the two algorithms^g are identical. This equality is a property of the logit^h link — the canonical link for the Bernoulli family — rather than of binary regression in general. If the same data are fitted with a probit or complementary log-log link, the two information matrices differ and the two algorithms take different steps.

There is a second consequence. Because W has non-negative diagonalⁱ entries, $X^\top W X$ is positive semi-definite and the Hessian $-X^\top W X$ is negative semi-definite *everywhere*. The log-likelihood^j is therefore concave, as Exercise 11.5 used, which guarantees that any stationary point^k found by Newton's method is a global maximizer.

^a<https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian>

^b<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

^c<https://morrison-lab.github.io/pds/expectation.html#def-expectation>

^d<https://morrison-lab.github.io/sds/intro-MLEs.html#def-oinf>

^e<https://morrison-lab.github.io/mds/optimization.html#def-newton-method>

^f<https://morrison-lab.github.io/sds/intro-MLEs.html#def-einf>

^g<https://morrison-lab.github.io/mds/expressions-algorithms.html#def-algorithm>

^h<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

ⁱ<https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-diagonal-matrix>

^j<https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik>

^k<https://morrison-lab.github.io/mds/optimization.html#def-stationary-point>

i Remark (Fisher information is the curvature of the Kullback–Leibler divergence)

Remark 8.1 (Fisher information is the curvature of the Kullback–Leibler divergence). The expected (Fisher) information^a is also the second derivative of the Kullback–Leibler divergence^b of the model with parameter $\vec{\beta}$ from the model with the true parameter, taken at the true parameter (proved in the sds notes^c). The expected [cross-entropy loss](#) of this chapter equals that divergence plus the entropy^d of the true distribution^e, which does not depend on $\vec{\beta}$. So the two have the same curvature, and Fisher scoring steps using the curvature of the quantity that [logistic regression](#) minimizes on average.

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-einf>

^b<https://morrison-lab.github.io/pds/expectation.html#def-kl-divergence>

^c<https://morrison-lab.github.io/sds/intro-MLEs.html#thm-fisher-kl-curvature>

^d<https://morrison-lab.github.io/pds/expectation.html#def-entropy>

^e<https://morrison-lab.github.io/pds/random-variables.html#def-cdf>

i Watch

Andrew Ng fits [logistic regression](#) by Newton's method^a, from 1:05:57^b [9], Lecture 3.

^a<https://morrison-lab.github.io/mds/optimization.html#def-newton-method>

^b<https://www.youtube.com/watch?v=het9HFqo1TQ&t=3957s>

i Watch

Hastie and Tibshirani fit [logistic regression](#) in code in their Python lab^a [4].

^a<https://www.youtube.com/watch?v=wOGBILLuc4I>

i Read

GeeksforGeeks explains Newton's method^a [20].

^a<https://www.geeksforgeeks.org/machine-learning/newtons-method-in-machine-learning/>

9 How sure are the coefficients?

A fitted coefficient is one draw from a sampling distribution⁵⁷, like any estimate⁵⁸. The curvature of the log-likelihood⁵⁹ that Newton's method uses to compute its step also determines how tightly $\hat{\beta}$ is estimated⁶⁰: a log-likelihood sharply peaked at $\hat{\beta}$ yields small standard errors⁶¹, and a flat one leaves a wide range of plausible values.

i Remark (Inference for logistic regression)

Remark 9.1 (Inference for logistic regression). The statistics notes treat the deviance, the null deviance, the likelihood ratio test of the slopes, and the Wald interval for a fitted logistic coefficient in Inference for Logistic Regression^a. That page shows that the drop from the null deviance to the deviance is the likelihood ratio statistic. It also applies the central limit theorem for maximum likelihood estimators to show that, for a large sample, the fitted coefficients $\hat{\beta}$ are approximately Gaussian with mean $\vec{\beta}$ and covariance $(D^\top W D)^{-1}$, where W is the weight matrix of Theorem 6.3. In that result the weights $\pi_i(1 - \pi_i)$ are computed from the true probabilities; the standard errors, like the Newton step, replace them with the weights computed from the fitted probabilities.

^a<https://morrison-lab.github.io/sds/logistic-inference.html>

10 Fitting on real data

Software fits the model without hand calculation. The next exercise applies it to a dataset where the outcome is difficult to predict.

Exercise 10.1 (Predicting the direction of the stock market). The `Smarket` data hold the daily returns of the S&P 500 index on 1250 trading days from 2001 to 2005 [21, Sec. 4.7]. For each day, the data give the returns on each of the five previous trading days (`Lag1` to `Lag5`), the number of shares traded that day (`Volume`), that day's own return (`Today`), and whether the index went up or down that day (`Direction`). `Direction` is `Down` exactly when `Today` is negative, so `Today` is not a predictor.

Fit a logistic regression for the probability that the market goes up, using the five lagged returns and the volume as predictors. Fit it on the days before 2005 only, then predict the direction for each day in 2005. Threshold the predicted probability at 0.5.

How often is the prediction correct on the 2005 days?

Solution

Solution 10.1.

⁵⁷<https://morrison-lab.github.io/pds/random-variables.html#def-cdf>

⁵⁸<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

⁵⁹<https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik>

⁶⁰<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

⁶¹<https://morrison-lab.github.io/sds/estimation.html#def-SE>

10.1 Python

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import statsmodels.formula.api as smf

smarket_py = pd.read_csv("data/smarket.csv")
print(smarket_py.head(3))
```

	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today	Direction
0	2001	0.381	-0.192	-2.624	-1.055	5.010	1.1913	0.959	Up
1	2001	0.959	0.381	-0.192	-2.624	-1.055	1.2965	1.032	Up
2	2001	1.032	0.959	0.381	-0.192	-2.624	1.4112	-0.623	Down

```
smarket_py["up"] = (smarket_py["Direction"] == "Up").astype(int)
train_py = smarket_py["Year"] < 2005

fit_py = smf.glm(
    "up ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume",
    data=smarket_py[train_py],
    family=sm.families.Binomial(),
).fit()
p_py = fit_py.predict(smarket_py[~train_py])
pred_py = np.where(p_py > 0.5, "Up", "Down")
actual_py = smarket_py.loc[~train_py, "Direction"]

print(pd.crosstab(pred_py, actual_py, rownames=["predicted"], colnames=["actual"]))
```

	Down	Up
predicted		
Down	77	97
Up	34	44

```
print("accuracy:", round(float(np.mean(pred_py == actual_py)), 3))

accuracy: 0.48

print("always Up:", round(float(np.mean(actual_py == "Up")), 3))

always Up: 0.56

def wald_ci_py(p, n):
    # A 95% interval for a proportion: p plus or minus 1.96 standard errors.
    half = 1.96 * np.sqrt(p * (1 - p) / n)
    return round(float(p - half), 3), round(float(p + half), 3)

print(wald_ci_py(float(np.mean(pred_py == actual_py)), len(actual_py)))

(0.418, 0.542)
```

10.2 R

```
smarket <- read.csv("data/smarket.csv")
str(smarket)
```

```
'data.frame': 1250 obs. of 9 variables:
 $ Year      : int  2001 2001 2001 2001 2001 2001 2001 2001 2001 2001 2001 ...
 $ Lag1      : num  0.381 0.959 1.032 -0.623 0.614 ...
 $ Lag2      : num  -0.192 0.381 0.959 1.032 -0.623 ...
 $ Lag3      : num  -2.624 -0.192 0.381 0.959 1.032 ...
```

```

$ Lag4      : num  -1.055 -2.624 -0.192 0.381 0.959 ...
$ Lag5      : num   5.01 -1.055 -2.624 -0.192 0.381 ...
$ Volume    : num   1.19 1.3 1.41 1.28 1.21 ...
$ Today     : num   0.959 1.032 -0.623 0.614 0.213 ...
$ Direction: chr   "Up" "Up" "Down" "Up" ...

smarket <- dplyr::mutate(smarket, up = as.integer(Direction == "Up"))
train <- smarket$Year < 2005

fit <- glm(
  up ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
  data = dplyr::filter(smarket, train),
  family = binomial
)
p <- predict(fit, dplyr::filter(smarket, !train), type = "response")
pred <- ifelse(p > 0.5, "Up", "Down")
actual <- smarket$Direction[!train]

table(predicted = pred, actual = actual)

      actual
predicted Down Up
      Down   77 97
      Up    34 44

accuracy <- mean(pred == actual)
always_up <- mean(actual == "Up")
c(accuracy = accuracy, always_up = always_up)

accuracy always_up
0.4801587 0.5595238

# A 95% interval for an accuracy: the proportion plus or minus 1.96 standard errors,
# which is accurate when the number of test days is in the hundreds.
wald_ci <- function(p, n) {
  p + c(lower = -1, upper = 1) * qnorm(0.975) * sqrt(p * (1 - p) / n)
}
ci_accuracy <- wald_ci(accuracy, length(actual))
ci_always_up <- wald_ci(always_up, length(actual))
# Both rules are scored on the same days, so the paired difference in
# correctness, day by day, has a t interval for its mean.
paired_ci <- function(a, b) {
  d <- as.integer(a) - as.integer(b)
  mean(d) + c(lower = -1, upper = 1) * qt(0.975, length(d) - 1) * sd(d) / sqrt(length(d))
}
ci_vs_up <- paired_ci(pred == actual, actual == "Up")

```

The model predicts the right direction on 48% of the 252 days in 2005 (95% interval [41.8, 54.2]%).

Exercise 10.2 (Predicting the direction of the stock market (part 2)). Use the setup of Exercise 10.1. How does that compare with always predicting “Up”?

Solution

Solution 10.2. Always predicting “Up” is right on 56% of them ([49.8, 62.1]%). Each interval is the proportion plus or minus 1.96 standard errors^a, $\sqrt{\hat{p}(1-\hat{p})/n}$ each. The two rules are scored on the same days, so their difference is taken day by day: the model’s accuracy minus the constant^b rule’s is -7.9 percentage points (95% interval [-18.2, 2.3]), an interval that includes 0. The intervals reflect

sampling variation across the test days in 2005 rather than across the training set. The fitted model does worse than a rule that ignores every predictor on these days, but the interval shows that these days alone do not establish that it is worse. Its predicted probabilities are all between 0.44 and 0.53, so the model never separates one day from another with any confidence. Figure 28 shows that the days that went up and the days that went down receive almost the same predictions.

^a<https://morrison-lab.github.io/sds/estimation.html#def-SE>

^b<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

```
library(ggplot2)
ggplot(tibble::tibble(p = p, actual = actual), aes(p, actual)) +
  geom_jitter(height = 0.15, width = 0, colour = "grey40") +
  geom_vline(xintercept = 0.5, linetype = "dashed") +
  labs(x = "predicted probability of Up", y = "actual direction")
```

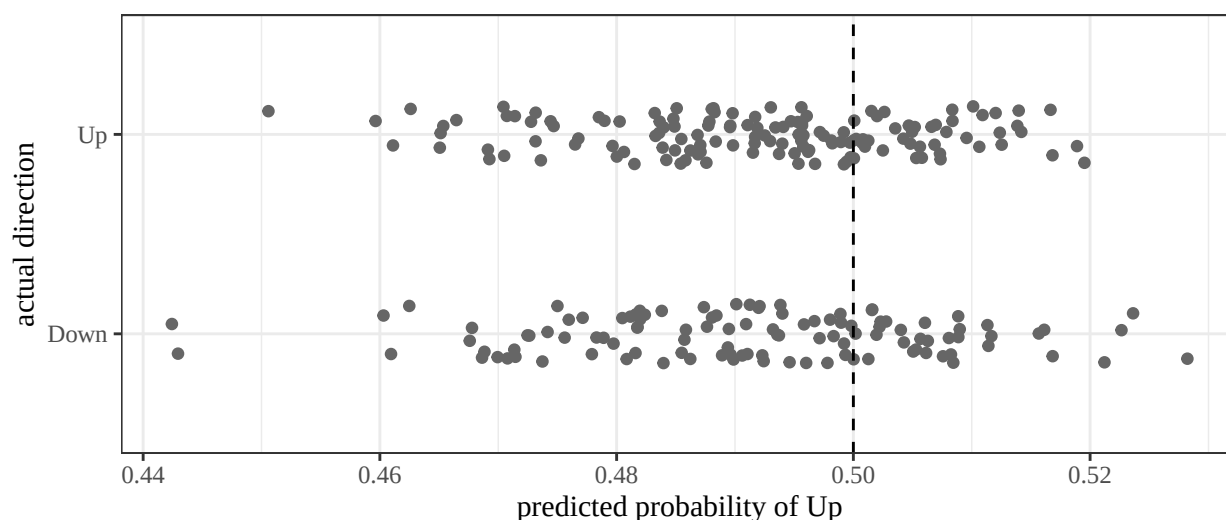


Figure 28: Predicted probability that the market goes up, for each trading day in 2005, by what the market actually did. The dashed line is the threshold of 0.5.

Exercise 10.3 (Fitting the stock market model by gradient descent). Exercise 10.1 fitted the 2001 to 2004 `Smarket` days with a library routine. Fit the same model with the gradient descent of Exercise 6.6. Standardize the six predictors with the training days' means and standard deviations, start from $\vec{\beta} = \vec{0}$, and use step size $\alpha = 0.5$ for 2,000 steps.

How close are the coefficients to the library fit, after you convert them back to the original units?

Solution

Solution 10.3.

10.3 Python

```

cols = ["Lag1", "Lag2", "Lag3", "Lag4", "Lag5", "Volume"]
tr = smarket_py[train_py]
mu, sd = tr[cols].mean(), tr[cols].std()
Xs_py = np.column_stack([np.ones(len(tr)), ((tr[cols] - mu) / sd).to_numpy()])
Xte_py = np.column_stack(
    [np.ones((~train_py).sum()), ((smarket_py.loc[~train_py, cols] - mu) / sd).to_numpy()]
)
y_tr_py = tr["up"].to_numpy()
sig = lambda z: 0.5 * (1 + np.tanh(0.5 * z))

b_py = np.zeros(Xs_py.shape[1])
for _ in range(2000):
    b_py = b_py - 0.5 * Xs_py.T @ (sig(Xs_py @ b_py) - y_tr_py) / len(y_tr_py)

slopes_py = b_py[1:] / sd.to_numpy()
intercept_py = b_py[0] - np.sum(slopes_py * mu.to_numpy())
print("gradient descent:", np.round(np.r_[intercept_py, slopes_py], 3))

gradient descent: [ 0.191 -0.054 -0.046  0.007  0.006 -0.004 -0.116]
print("library fit:      ", fit_py.params.round(3).to_numpy())

library fit:      [ 0.191 -0.054 -0.046  0.007  0.006 -0.004 -0.116]

same_py = (sig(Xte_py @ b_py) > 0.5) == (p_py.to_numpy() > 0.5)
print("same predictions:", same_py.mean())

same predictions: 1.0

```

10.4 R

```

cols <- c("Lag1", "Lag2", "Lag3", "Lag4", "Lag5", "Volume")
smarket_train <- smarket |>
  dplyr::filter(train) |>
  dplyr::select(dplyr::all_of(cols))
smarket_test <- smarket |>
  dplyr::filter(!train) |>
  dplyr::select(dplyr::all_of(cols))
mu <- colMeans(smarket_train)
sdev <- apply(smarket_train, 2, sd)
# nolint start: object_name_linter.
Xs <- cbind(1, scale(smarket_train, mu, sdev))
Xte <- cbind(1, scale(smarket_test, mu, sdev))
# nolint end
y_tr <- smarket$up[train]
sig <- function(z) 0.5 * (1 + tanh(0.5 * z))

b <- rep(0, ncol(Xs))
for (i in 1:2000) {
  b <- b - 0.5 * drop(t(Xs) %*% (sig(drop(Xs %*% b)) - y_tr)) / length(y_tr)
}

slopes <- b[-1] / sdev
coef_gd <- c(b[1] - sum(slopes * mu), slopes)
rbind(gradient_descent = coef_gd, library_fit = coef(fit))

               Lag1      Lag2      Lag3      Lag4
gradient_descent 0.1912126 -0.05417829 -0.04580533 0.007200118 0.006440875

```

```
library_fit      0.1912126 -0.05417829 -0.04580533 0.007200118 0.006440875
               Lag5      Volume
gradient_descent -0.004222672 -0.116257
library_fit      -0.004222672 -0.116257
```

```
same <- mean((sig(drop(Xte %*% b)) > 0.5) == (p > 0.5))
same
```

```
[1] 1
```

The two sets of coefficients agree to 4.3e-10 or better.

Exercise 10.4 (Fitting the stock market model by gradient descent (part 2)). Use the setup of Exercise 10.3.

Do the two fits make the same predictions on the 2005 days?

Solution

Solution 10.4. The two fits make the same prediction on 100% of the 2005 days. Gradient descent^a reaches the library's answer because the [log loss](#) is convex^b, and standardizing the predictors keeps every coefficient on a similar scale, so one step size works for all of them.

^a<https://morrison-lab.github.io/mds/optimization.html#def-gradient-descent>

^b<https://morrison-lab.github.io/mds/algebra-sums.html#def-convex-function>

i Read

The lab for this chapter in ISLP_labs^a fits the same model, and then fits [linear discriminant analysis](#), [quadratic discriminant analysis](#) (Section 13), and nearest neighbors [21].

^ahttps://github.com/intro-stat-learning/ISLP_labs

11 When the coefficients diverge

Everything so far assumed that a finite maximizer of the likelihood⁶² exists. When a hyperplane separates the classes without error, no finite maximizer exists. This case occurs frequently in small samples⁶³: fitting software still returns finite numbers after stopping at its iteration limit, and the Wald test gives the opposite of the conclusion from the likelihood ratio test.

Exercise 11.1 (When the data separate perfectly). A second trial, six patients, with a dose x and recovery y :

x	-3	-2	-1	+1	+2	+3
y	0	0	0	1	1	1

⁶²<https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik>

⁶³<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample>

Every patient below dose 0 failed and every patient above it recovered. The data are **separated**: a single threshold classifies all six correctly.

Argue as in Exercise 7.3 that $\hat{\beta}_0 = 0$, and show that the log-likelihood^a then reduces to

$$\ell(\beta_1) = 2 \log \text{expit}(\beta_1) + 2 \log \text{expit}(2\beta_1) + 2 \log \text{expit}(3\beta_1).$$

Show that ℓ is strictly increasing, and find $\sup_{\beta_1} \ell(\beta_1)$. Determine whether the supremum is attained.

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik>

Solution

Solution 11.1. The design is symmetric^a — the three $y = 0$ doses are the negatives of the three $y = 1$ doses — so the intercept^b score $\sum_i (y_i - \pi_i)$ vanishes at $\beta_0 = 0$ for the same reason as in Exercise 7.3. With $\beta_0 = 0$, the six contributions are $\log \text{expit}(3\beta_1)$, $\log \text{expit}(2\beta_1)$, $\log \text{expit}(\beta_1)$, $\log \text{expit}(\beta_1)$, $\log \text{expit}(2\beta_1)$, and $\log \text{expit}(3\beta_1)$, using $1 - \text{expit}(-\eta) = \text{expit}(\eta)$ for the three failures, which gives the stated ℓ . Differentiating gives

$$\begin{aligned} \ell'(\beta_1) &= 2[1 - \text{expit}(\beta_1)] + 4[1 - \text{expit}(2\beta_1)] + 6[1 - \text{expit}(3\beta_1)] \\ &= 2 \text{expit}(-\beta_1) + 4 \text{expit}(-2\beta_1) + 6 \text{expit}(-3\beta_1) \\ &> 0 \end{aligned}$$

for every β_1 , since expit is strictly positive. Thus ℓ is strictly increasing, and as $\beta_1 \rightarrow \infty$ each $\text{expit} \rightarrow 1$ and each $\log \text{expit} \rightarrow 0$, so $\sup \ell = 0$ and the likelihood has supremum 1.

That supremum is not attained. Because $\text{expit}(\eta) < 1$ strictly for every finite η , $\ell(\beta_1) < 0$ at every finite β_1 . **There is no maximum likelihood estimate.** The likelihood has no finite maximizer: it increases strictly toward its supremum without reaching it, as $\beta_1 \rightarrow \infty$.

^a<https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-symmetric-matrix>

^b<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

Exercise 11.2 (When the data separate perfectly (part 2)). Use the setup of Exercise 11.1.

Now prove this result in general, without direct arithmetic. Suppose some $\vec{\beta}$ satisfies $\vec{\beta} \cdot \tilde{x}_i > 0$ for every observation with $y_i = 1$ and $\vec{\beta} \cdot \tilde{x}_i < 0$ for every observation with $y_i = 0$. Show that replacing $\vec{\beta}$ by $c\vec{\beta}$ for any $c > 1$ strictly increases the likelihood^a, and state what that implies about maximizing the likelihood.

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik>

Solution

Solution 11.2. By the margin form of Theorem 5.2, the log-likelihood^a is $\ell(\vec{\beta}) = \sum_i \log \text{expit}(\tilde{y}_i \vec{\beta} \cdot \tilde{x}_i)$ with $\tilde{y}_i = 2y_i - 1 \in \{-1, +1\}$.

The separation hypothesis states that every margin is strictly positive: $\tilde{y}_i \vec{\beta} \cdot \tilde{x}_i > 0$ for all i . Scaling to $c\vec{\beta}$ multiplies every margin by c , so for $c > 1$ each margin strictly increases, and because expit is strictly increasing, every term of the sum strictly increases. Hence $\ell(c\vec{\beta}) > \ell(\vec{\beta})$.

No $\vec{\beta}$ can be a maximizer, because $2\vec{\beta}$ always achieves a strictly larger log-likelihood. The supremum of the log-likelihood^b is 0, so the supremum of the likelihood is 1. Both are approached along the ray $c \rightarrow \infty$, and neither is attained at any finite $\vec{\beta}$. Solution 11.1 was this same argument for a specific six-point dataset.

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik>

^b<https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik>

Exercise 11.3 (When the data separate perfectly (part 3)). Use the setup of Exercise 11.1. The observed information for β_1 is $I(\beta_1) = \sum_i x_i^2 \pi_i(1 - \pi_i)$. Evaluate it at $\beta_1 = 1, 2, 5, 10, 20$, and give the standard error^a $1/\sqrt{I}$ at each. Describe what a fitting routine reports if it terminates when the gradient norm falls below a tolerance.

^a<https://morrison-lab.github.io/sds/estimation.html#def-SE>

Solution

Solution 11.3. With $\beta_0 = 0$, $I(\beta_1)$ depends on β_1 alone, so it can be evaluated directly at the five values:

```
sep <- tibble::tibble(x = c(-3, -2, -1, 1, 2, 3), y = c(0, 0, 0, 1, 1, 1))
sep_info <- function(b1) {
  p <- plogis(b1 * sep$x)
  sum(sep$x^2 * p * (1 - p))
}
sep_b1 <- c(1, 2, 5, 10, 20)
sep_i <- vapply(sep_b1, sep_info, numeric(1))
sep_se <- 1 / sqrt(sep_i)

# Typeset a number for the page: plain if it is not tiny,
# and as  $m \times 10^{\{e\}}$  when its exponent is -4 or lower.
sep_num <- function(v, digits = 3) {
  v <- signif(v, digits)
  e <- floor(log10(abs(v)))
  if (e > -4) {
    return(format(v))
  }
  sprintf("%s \\times 10^{\\%d}", format(signif(v / 10^e, digits)), e)
}
sep_nums <- function(v, digits = 3) {
  vapply(v, sep_num, "", digits = digits)
}

knitr::kable(
  tibble::tibble(
    b1 = sep_b1,
    info = sep_nums(sep_i),
    se = sep_nums(sep_se)
  ),
  col.names = c("$\\coef{1}$", "$I(\\coef{1})$", "$\\mathrm{SE} = 1/\\sqrt{I}$"),
  align = "r"
)
```

β_1	$I(\beta_1)$	$SE = 1/\sqrt{I}$
1	2.05	0.699
2	0.396	1.59
5	0.0137	8.55
10	9.08×10^{-5}	105
20	4.12×10^{-9}	15600

The information approaches zero because I is a weighted sum of $\pi_i(1 - \pi_i)$ and every π_i approaches 0 or 1, where that product vanishes.

A fitting routine that stops when the gradient is small will therefore terminate, even though it has not found a maximum. The gradient is small because the *curvature* has vanished, and the routine halts wherever its stopping tolerance and iteration limit stop it. Fitting these six points with R's `glm`:

```
sep_fit <- glm(y ~ x, family = binomial, data = sep)
```

```
parameters::parameters(sep_fit) |> parameters::print_md(digits = 3)
```

Parameter	Log-Odds	SE	95% CI	z	p
(Intercept)	-9.065e-08	48419.140	(-94899.770, 94899.770)	-1.872e-12	> .999
x	23.268	48418.863	(-94875.958, 94922.495)	4.806e-04	> .999

```
sep_b1_hat <- coef(sep_fit)[["x"]]
sep_p_hat <- summary(sep_fit)$coefficients["x", "Pr(>|z|)"]
# The slope's p-value, formatted by insight::format_p()
sep_p_printed <- insight::format_p(sep_p_hat)
```

R warns, “fitted probabilities numerically 0 or 1 occurred”, meaning some fitted probabilities are indistinguishable from 0 or 1, and stops after 24 iterations. The reported value 23.27 is an artifact of the stopping tolerance rather than an estimate^a of a finite parameter. Running the same fit with a tighter tolerance:

```
sep_tight <- suppressWarnings(
  glm(y ~ x, family = binomial, data = sep,
      control = glm.control(epsilon = 1e-14, maxit = 100))
)
c(slope = coef(sep_tight)[["x"]], iterations = sep_tight$iter)
```

```
      slope iterations
      30.68281    32.00000
returns 30.68 after 32 iterations.
```

^a<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

Exercise 11.4 (When the data separate perfectly (part 4)). Use the setup of Exercise 11.1.

The Wald statistic for $H_0 : \beta_1 = 0$ is $z = \hat{\beta}_1 \sqrt{I(\hat{\beta}_1)}$. Evaluate it at the same five values. Separately, compute the likelihood^a ratio statistic $G^2 = 2[\sup \ell - \ell(0)]$ and its p -value. The two tests yield opposite conclusions. State which test is valid, and explain why the other test fails.

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik>

Solution

Solution 11.4. The Wald statistic $z = \hat{\beta}_1 \sqrt{I(\hat{\beta}_1)}$ behaves non-monotonically:

```
sep_z <- sep_b1 * sqrt(sep_i)
sep_p <- 2 * pnorm(-sep_z)
sep_wald <- rbind(
  "$z$" = sep_nums(sep_z),
  "$p$" = sep_nums(sep_p)
)
colnames(sep_wald) <- paste0("$\\coef{1} = ", sep_b1, "$")
knitr::kable(sep_wald, align = "r")
```

	$\beta_1 = 1$	$\beta_1 = 2$	$\beta_1 = 5$	$\beta_1 = 10$	$\beta_1 = 20$
z	1.43	1.26	0.584	0.0953	0.00128
p	0.153	0.208	0.559	0.924	0.999

```
sep_peak <- optimize(
  \ (b1) b1 * sqrt(sep_info(b1)),
  interval = c(0, 10),
  maximum = TRUE
)
sep_g2 <- 12 * log(2)
sep_g2_p <- pchisq(sep_g2, df = 1, lower.tail = FALSE)
```

It rises to about 1.44 near $\beta_1 = 1.12$ and then decreases toward zero. It never reaches conventional significance thresholds along the path, and as the fit improves, the Wald test reports *weaker* significance. R's fit in Solution 11.3 assigns the slope^a a p -value of 0.9996, which formats as $p > .999$, on data where the dose predicts the outcome without error.

The likelihood^b ratio test gives the opposite conclusion. Because

$$\begin{aligned}\ell(0) &= 6 \log \frac{1}{2} \\ &= -6 \log 2,\end{aligned}$$

and $\sup \ell = 0$, we have

$$\begin{aligned}G^2 &= 2[0 - (-6 \log 2)] \\ &= 12 \log 2 \\ &= 8.3178,\end{aligned}$$

which against χ_1^2 gives $p = 0.0039$. This value equals R's null deviance in the same fit, 8.3178, with [residual](#) deviance 3.1×10^{-10} .

Use the likelihood ratio test. The Wald statistic measures how many standard errors^c $\hat{\beta}_1$ is from zero, and both the numerator and denominator diverge: the coefficient grows without bound, but the standard error^d grows faster, so their ratio approaches zero. The Wald statistic uses the curvature *at* $\hat{\beta}_1$, out in the flat tail where the curvature vanishes, rather than the change in log-likelihood^e between 0 and $\hat{\beta}_1$, which measures how much better the separated fit explains the data. This phenomenon is the Hauck–Donner effect, described by Hauck and Donner in 1977.

Consequently, on a separated or nearly separated dataset, a large reported standard error does not indicate weak evidence; it is a symptom of separation.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope>

^b<https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik>

^c<https://morrison-lab.github.io/sds/estimation.html#def-SE>

^d<https://morrison-lab.github.io/sds/estimation.html#def-SE>

^e<https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik>

Exercise 11.5 (When the data separate perfectly (part 5)). Use the setup of Exercise 11.1.

Add a ridge penalty and maximize $\ell(\beta_1) - \frac{\lambda}{2}\beta_1^2$ instead. Show that a maximizer exists for every $\lambda > 0$. Find $\hat{\beta}_1$ for $\lambda = 1$, $\lambda = 0.1$ and $\lambda = 0.01$, and explain what the penalty achieves and what trade-off it introduces.

Solution

Solution 11.5. We maximize $f(\beta_1) = \ell(\beta_1) - \frac{\lambda}{2}\beta_1^2$.

Every log expit is negative, so $\ell \leq 0$ everywhere, and therefore $f(\beta_1) \leq -\frac{\lambda}{2}\beta_1^2 \rightarrow -\infty$ as $|\beta_1| \rightarrow \infty$. A continuous function that tends to $-\infty$ in both directions attains its maximum, so a maximizer exists. It

is also unique: ℓ is concave because its Hessian^a is $-X^T W X$ with W positive definite^b, and $-\frac{\lambda}{2}\beta_1^2$ is strictly concave, so f is strictly concave.

Setting $f' = 0$ gives $2 \expit(-\beta_1) + 4 \expit(-2\beta_1) + 6 \expit(-3\beta_1) = \lambda\beta_1$. For $\beta_1 < 0$ the left-hand side is positive and $\lambda\beta_1$ is negative, so any root must be positive. For $\beta_1 \geq 0$ the left-hand side decreases from 6 at $\beta_1 = 0$ toward 0, while the right-hand side increases from 0, so the two curves intersect at a unique point. Solving numerically:

```
sep_lambda <- c(1, 0.1, 0.01)
sep_ridge_fit <- function(lambda) {
  uniroot(
    \(b1) 2 * plogis(-b1) + 4 * plogis(-2 * b1) + 6 * plogis(-3 * b1) - lambda * b1,
    interval = c(0, 100),
    tol = 1e-10
  )$root
}
sep_ridge <- vapply(sep_lambda, sep_ridge_fit, numeric(1))
sep_ridge_tbl <- rbind("$\\hcoef{1}$" = round(sep_ridge, 3))
colnames(sep_ridge_tbl) <- paste0("$\\lambda = ", sep_lambda, "$")
knitr::kable(sep_ridge_tbl, align = "r")
```

	$\lambda = 1$	$\lambda = 0.1$	$\lambda = 0.01$
$\hat{\beta}_1$	1.104	2.295	3.946

The penalty yields a finite, unique maximizer with a finite standard error^c and a well-defined Wald statistic.

However, the resulting value depends strongly on the choice of λ . The maximizer increases by a factor of 3.6 when λ decreases by a factor of 100, and it diverges as $\lambda \rightarrow 0$ because the data alone do not bound the slope^d. The penalty does not supply information missing from the likelihood^e; it imposes a prior preference for smaller coefficients and reports the resulting penalized maximizer.

That regularization is appropriate when prior knowledge rules out very large coefficients, but the dependence on λ should be stated explicitly. The conclusion from the data is not that $\hat{\beta}_1 = 2.295$, but that the six patients are separated by dose, the association is statistically significant at $p = 0.0039$, and the magnitude of the slope is not identified by this dataset alone.

^a<https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian>

^b<https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-positive-definite>

^c<https://morrison-lab.github.io/sds/estimation.html#def-SE>

^d<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope>

^e<https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik>

Theorem 11.1 (Separable data have no maximum likelihood estimate). *Suppose some $\vec{\beta}$ has $\vec{\beta} \cdot \tilde{x}_i > 0$ for every observation with $y_i = 1$ and $\vec{\beta} \cdot \tilde{x}_i < 0$ for every observation with $y_i = 0$. Then $\ell(c\vec{\beta}) > \ell(\vec{\beta})$ for every $c > 1$, so the likelihood of the logistic regression has no maximizer. The supremum of the log-likelihood ℓ is 0, so the supremum of the likelihood is 1. Both are approached along the ray $c \rightarrow \infty$ and attained nowhere.*

i Proof

Proof. This is the solution to Exercise 11.2: see Solution 11.2. □

Example 11.1 (The likelihood keeps rising along a ray). Consider the six patients of Exercise 11.1, with doses $-3, -2, -1, 1, 2, 3$ and recoveries $0, 0, 0, 1, 1, 1$. The slope $\beta_1 = 1$ separates them. Scale it by $c = 1, 2, 5, 10$, with $\beta_0 = 0$.

```

sp_x <- c(-3, -2, -1, 1, 2, 3)
sp_y <- c(0, 0, 0, 1, 1, 1)
sp_llik <- function(b1) sum(dbinom(sp_y, 1, plogis(b1 * sp_x), log = TRUE))
sp_c <- c(1, 2, 5, 10)
data.frame(c = sp_c, loglik = vapply(sp_c, sp_llik, numeric(1)))

```

```

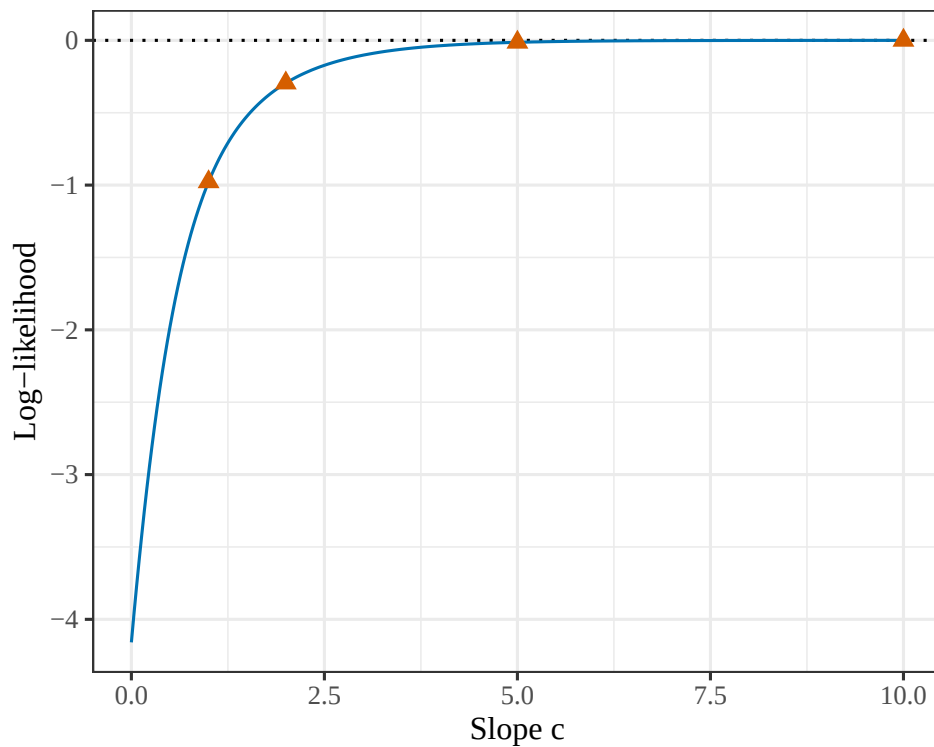
# A tibble: 4 x 2
      c    loglik
  <dbl>   <dbl>
1     1 -0.978
2     2 -0.295
3     5 -0.0135
4    10 -0.0000908

```

As Theorem 11.1 says, the log-likelihood rises at every step, from -0.978 at $c = 1$ to $-9.08\text{e-}05$ at $c = 10$. It approaches its supremum 0 and never reaches it, so the likelihood approaches its supremum 1 and never reaches that either.

11.1 R

```
library(ggplot2)
sep_grid <- data.frame(c = seq(0, 10, length.out = 300))
sep_grid$loglik <- vapply(sep_grid$c, sp_llik, numeric(1))
sep_marks <- data.frame(c = sp_c, loglik = vapply(sp_c, sp_llik, numeric(1)))
ggplot(sep_grid, aes(c, loglik)) +
  geom_hline(yintercept = 0, linetype = "dotted") +
  geom_line(colour = "#0072B2") +
  geom_point(data = sep_marks, colour = "#D55E00", shape = 17, size = 2.5) +
  labs(x = "Slope c", y = "Log-likelihood")
```



(a)

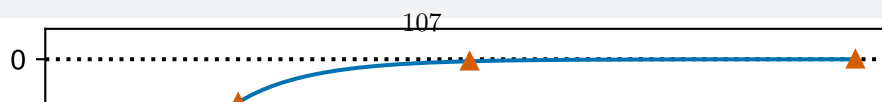
11.2 Python

```
import matplotlib.pyplot as plt
import numpy as np

x_sep = np.array([-3, -2, -1, 1, 2, 3])
y_sep = np.array([0, 0, 0, 1, 1, 1])

def llik_sep(b1):
    p = 1 / (1 + np.exp(-b1 * x_sep))
    return np.sum(y_sep * np.log(p) + (1 - y_sep) * np.log(1 - p))

c_sep = np.linspace(0, 10, 300)
marks_sep = np.array([1, 2, 5, 10])
fig, ax = plt.subplots(figsize=(5, 4), layout="constrained")
ax.axhline(0, color="black", linestyle="-");
ax.plot(c_sep, [llik_sep(c) for c in c_sep], color="#0072B2");
ax.plot(marks_sep, [llik_sep(c) for c in marks_sep], color="#D55E00", marker="^", linestyle="");
ax.set_xlabel("Slope c");
ax.set_ylabel("Log-likelihood");
plt.show()
```



11.3 R

```
library(ggplot2)

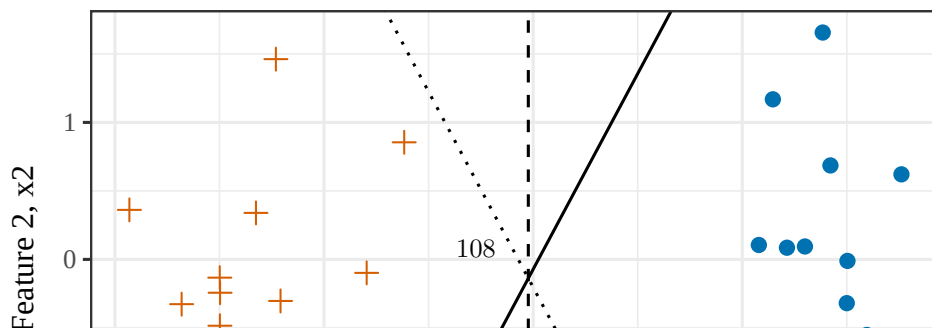
set.seed(571)
n_per_class <- 15
sim_df <- data.frame(
  x1 = c(rnorm(n_per_class, -2.5, 0.6), rnorm(n_per_class, 2.5, 0.6)),
  x2 = rnorm(2 * n_per_class, 0, 0.9),
  class = rep(c("Left", "Right"), each = n_per_class)
)

# Each boundary passes through the midpoint of the class means,
# at a different angle (in degrees) from the horizontal axis.
pivot <- c(mean(sim_df$x1), mean(sim_df$x2))
angles <- c(f1 = 55, f2 = 90, f3 = 125)
boundary_df <- data.frame(
  boundary = paste(names(angles), "= 0"),
  angle = angles * pi / 180
)

# Check that every boundary separates the two classes.
side <- function(angle) {
  (sim_df$x1 - pivot[1]) * sin(angle) - (sim_df$x2 - pivot[2]) * cos(angle) > 0
}
stopifnot(all(vapply(
  boundary_df$angle,
  function(a) nrow(unique(data.frame(side(a), sim_df$class))) == 2,
  logical(1)
)))

# geom_abline cannot draw a vertical line, so draw each boundary as a segment.
boundary_df$x_start <- pivot[1] - 10 * cos(boundary_df$angle)
boundary_df$x_end <- pivot[1] + 10 * cos(boundary_df$angle)
boundary_df$y_start <- pivot[2] - 10 * sin(boundary_df$angle)
boundary_df$y_end <- pivot[2] + 10 * sin(boundary_df$angle)

ggplot(sim_df) +
  aes(x1, x2) +
  geom_point(aes(colour = class, shape = class), size = 2.5) +
  geom_segment(
    data = boundary_df,
    aes(x = x_start, y = y_start, xend = x_end, yend = y_end, linetype = boundary)
  ) +
  scale_colour_manual(values = c(Left = "#D55E00", Right = "#0072B2")) +
  scale_shape_manual(values = c(Left = 3, Right = 16)) +
  scale_linetype_manual(values = c(
    "f1 = 0" = "solid", "f2 = 0" = "dashed", "f3 = 0" = "dotted"
  )) +
  coord_cartesian(xlim = range(sim_df$x1), ylim = range(sim_df$x2)) +
  labs(x = "Feature 1, x1", y = "Feature 2, x2", colour = NULL, shape = NULL, linetype = NULL)
```



Every boundary in Figure 30 classifies the training data perfectly, and the boundary that is farthest from the nearest points of either class has the largest margin. Logistic regression does not find it. Its loss decays exponentially for points already on the right side and never reaches zero, so on separable data the likelihood keeps improving as $\|\tilde{\beta}\|$ grows and there is no finite optimum at all (Theorem 11.1). A penalty makes the problem well-posed (Solution 11.5). Only in the limit as the penalty strength goes to zero does the direction of the fitted boundary approach the maximum-margin one [22]. Support vector machines maximize the margin directly, at a finite solution.

12 More than two classes

Assigning each class its own coefficient vector and normalizing the exponentiated scores yields the **softmax** model. It reduces to binary **logistic regression** when there are two classes, and serves as the output layer of multi-class neural network classifiers in the neural networks chapter.

Exercise 12.1 (Comparing softmax probability vectors). Sketch the probability mass function that the softmax produces for each of these score vectors. Show the relative heights of the three probabilities in each vector; Exercise 12.2 computes their numerical values.

$$\tilde{v}_1 = \text{softmax} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \quad \tilde{v}_2 = \text{softmax} \begin{pmatrix} 3 \\ 3 \\ 3 \end{pmatrix}, \quad \tilde{v}_3 = \text{softmax} \begin{pmatrix} 0 \\ -1 \\ 2 \end{pmatrix}, \quad \tilde{v}_4 = \text{softmax} \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}.$$

Sketch all four, then state which two are identical and why this follows before computing any exponentials.

Solution

Solution 12.1. The probability vectors $\tilde{v}_1$ and $\tilde{v}_2$ are equal.

Exercise 12.6 showed that the softmax function is invariant to adding a constant^a to every score, and $(3, 3, 3)^\top = (0, 0, 0)^\top + 3$. Because the two score vectors differ by a constant shift, they produce the same probability vector without further calculation.

Both $\tilde{v}_1$ and $\tilde{v}_2$ are uniform, with $1/3$ in each coordinate (Figure 31). In $\tilde{v}_3$, the third coordinate is largest and the second coordinate is near zero; in $\tilde{v}_4$, the first coordinate is the largest, while the second and third coordinates are equal and smaller.

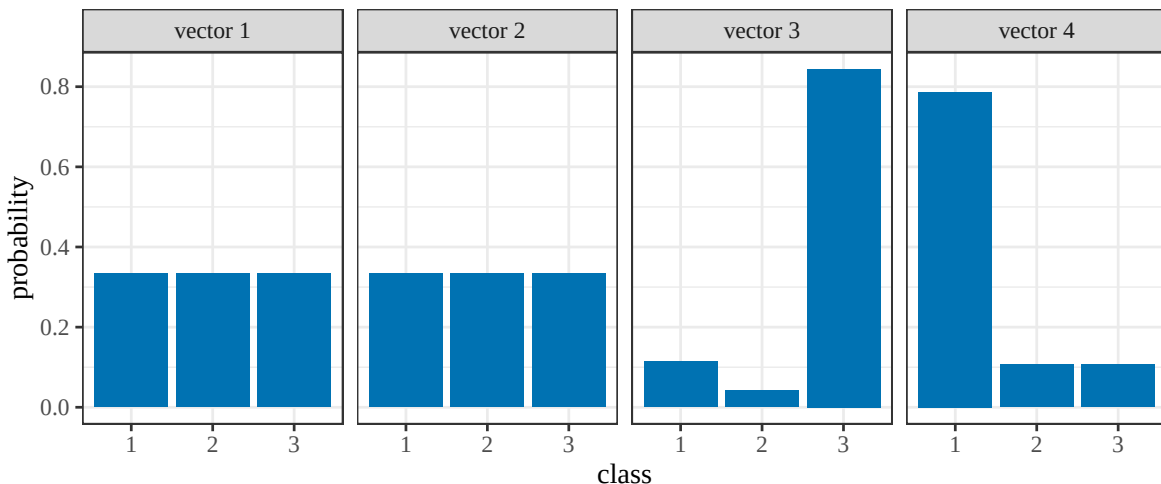


Figure 31: The probability mass functions^b of the four softmax vectors. Vectors 1 and 2 give the same probabilities, one third each. Vector 3, from the scores $(0, -1, 2)^\top$, puts most of its mass on the third class, and vector 4, from $(2, 0, 0)^\top$, on the first.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>
^b<https://morrison-lab.github.io/pds/random-variables.html#def-pmf>

Exercise 12.2 (Comparing softmax probability vectors (part 2)). Use the setup of Exercise 12.1. Give all four to three decimal places.

Solution

Solution 12.2.

	class 1	class 2	class 3
$\tilde{v}_1$	0.333	0.333	0.333
$\tilde{v}_2$	0.333	0.333	0.333
$\tilde{v}_3$	0.114	0.042	0.844
$\tilde{v}_4$	0.787	0.107	0.107

Exercise 12.3 (Comparing softmax probability vectors (part 3)). Use the setup of Exercise 12.1. In $\tilde{v}_3$, the largest score exceeds the smallest by 3. Give the ratio of the corresponding two probabilities, exactly, and state what determines it.

Solution

Solution 12.3. The ratio of any two softmax probabilities is

$$\begin{aligned}\frac{p_m}{p_k} &= \frac{e^{z_m}/Z}{e^{z_k}/Z} \\ &= e^{z_m - z_k},\end{aligned}$$

because the normalizer Z cancels. Here $z_3 - z_2 = 3$, so

$$\begin{aligned}p_3/p_2 &= e^3 \\ &= 20.09,\end{aligned}$$

which matches $0.844/0.042 = 20.1$.

This ratio depends only on the *difference* $z_m - z_k$ — not on the individual magnitudes of z_m and z_k , the number of classes, or the scores of the other classes. This property is the pairwise form of the shift invariance in Solution 12.1 and the multi-class generalization of Exercise 3.1: the difference between two class scores is the log-odds^a ratio between those two classes.

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

Exercise 12.4 (Comparing softmax probability vectors (part 4)). Use the setup of Exercise 12.1. Raising the first score of $\tilde{v}_4$ moves the output toward the indicator vector^a $(1, 0, 0)$. What limit must the scores take for the softmax output to equal $(1, 0, 0)$, and how does this relate to Exercise 11.1?

^a<https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-indicator-vector>

Solution

Solution 12.4. No finite score vector produces $(1, 0, 0)^\top$. Because $e^{z_m} > 0$ for every finite z_m , every softmax probability p_m is strictly positive, and the softmax output approaches $(1, 0, 0)^\top$ only in the

limit as $z_1 - z_2 \rightarrow +\infty$ and $z_1 - z_3 \rightarrow +\infty$.

This behavior is the C -class version of Exercise 11.1. When the training classes are linearly separable, maximizing the likelihood^a drives the score differences toward $+\infty$, so the coefficients grow without bound and no finite maximum likelihood^b estimate^c exists. For $C = 2$, this property reduces to Exercise 2.1: $\text{expit}(\eta) \in (0, 1)$ for all finite η .

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik>

^b<https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle>

^c<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

Exercise 12.5 (More than two classes). For C classes, give each class its own coefficient vector B_m and set

$$P(Y = m \mid \tilde{x}) = \frac{\exp\{B_m^\top \tilde{x}\}}{\sum_{k=1}^C \exp\{B_k^\top \tilde{x}\}},$$

the **softmax**.

Set $C = 2$ and show that this expression^a reduces to $P(Y = 1 \mid \tilde{x}) = \text{expit}(\vec{\beta} \cdot \tilde{x})$. Express $\vec{\beta}$ in terms of B_0 and B_1 .

^a<https://morrison-lab.github.io/mds/expressions-algorithms.html#def-expression>

Solution

Solution 12.5. With two classes the denominator has two terms, so

$$\begin{aligned} P(Y = 1 \mid \tilde{x}) &= \frac{e^{B_1^\top \tilde{x}}}{e^{B_0^\top \tilde{x}} + e^{B_1^\top \tilde{x}}} \\ &= \frac{1}{1 + e^{-(B_1 - B_0)^\top \tilde{x}}} \\ &= \text{expit}((B_1 - B_0)^\top \tilde{x}), \end{aligned}$$

after dividing numerator and denominator by $e^{B_0^\top \tilde{x}}$. Thus $\vec{\beta} = B_1 - B_0$: the single coefficient vector of the binary model is the *difference* between the two class coefficient vectors, not either vector alone.

This relation explains why every **logistic regression** coefficient is a comparison between classes: as shown in Exercise 3.1, β_j is a change in log odds^a, and log odds compare one class against another.

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

Exercise 12.6 (More than two classes (part 2)). Use the setup of Exercise 12.5.

Show that replacing every B_m by $B_m + \cdot$, for any fixed vector $\cdot$, leaves all C probabilities unchanged. How many free parameters does the model actually have? Give two ways to remove the excess, and say which one Exercise 12.5 was implicitly using.

Solution

Solution 12.6. Adding $\cdot$ to each B_k multiplies every exponential in the numerator and in all C terms of the denominator by the same factor $e^{\cdot^\top \tilde{x}}$, which cancels:

$$\begin{aligned} \frac{e^{(B_m + \cdot)^\top \tilde{x}}}{\sum_k e^{(B_k + \cdot)^\top \tilde{x}}} &= \frac{e^{\cdot^\top \tilde{x}} e^{B_m^\top \tilde{x}}}{e^{\cdot^\top \tilde{x}} \sum_k e^{B_k^\top \tilde{x}}} \\ &= P(Y = m \mid \tilde{x}). \end{aligned}$$

The model is therefore *overparameterized*. It has $C(p + 1)$ parameters, but only $(C - 1)(p + 1)$ of them are identified: adding a common vector $\cdot$ to all C columns moves B along a $(p + 1)$ -dimensional subspace

on which the likelihood^a is constant^b. Along this subspace the Hessian^c is singular, so an unregularized Newton solver cannot factorize the Hessian without a constraint.

Two standard ways to remove this non-identifiability are:

- **Designate a reference class:** set $B_1 = \vec{0}$ and interpret every other B_m relative to class 1. Solution 12.5 used this convention for $C = 2$ by setting $B_0 = \vec{0}$ and $\vec{\beta} = B_1$.
- **Add a ridge penalty.** Subtracting $\frac{\lambda}{2} \sum_m \|B_m\|^2$ from the log-likelihood^d makes the objective strictly concave and selects the unique representative with minimum squared Frobenius norm^e. Neural network classifiers typically retain all C columns and rely on weight decay or first-order optimization.

Retaining all C columns treats all classes symmetrically, uses the same gradient formula for every class, and stores the parameters in a single $(p + 1) \times C$ matrix, at the expense of $p + 1$ redundant parameters. Neither approach changes any fitted probability; they differ only in which parameter matrix B represents the fitted distribution^f, which is why different software packages can report different coefficients for the same fitted model.

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik>

^b<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

^c<https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian>

^d<https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik>

^e<https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-frobenius-norm>

^f<https://morrison-lab.github.io/pds/random-variables.html#def-cdf>

Exercise 12.7 (More than two classes (part 3)). Use the setup of Exercise 12.5.

Derive $\nabla_{B_m} \ell$ for a dataset of n observations, where $\ell = \sum_i \log P(Y = y_i | \tilde{x}_i)$. Some references write a factor of $1/n$ in front of this gradient. Say when that factor is correct and when it is a mistake.

Solution

Solution 12.7. The log-likelihood^a can be written as $\ell = \sum_i [B_{y_i}^\top \tilde{x}_i - \log \sum_k e^{B_k^\top \tilde{x}_i}]$.

The first term contributes $\tilde{x}_i$ to ∇_{B_m} when $y_i = m$ and $\vec{0}$ otherwise, which equals $\delta(y_i = m) \tilde{x}_i$. In the second term, only the $k = m$ summand depends on B_m , so

$$\begin{aligned} \nabla_{B_m} \log \sum_k e^{B_k^\top \tilde{x}_i} &= \frac{e^{B_m^\top \tilde{x}_i}}{\sum_k e^{B_k^\top \tilde{x}_i}} \tilde{x}_i \\ &= P(Y = m | \tilde{x}_i) \tilde{x}_i. \end{aligned}$$

Combining the two terms gives

$$\nabla_{B_m} \ell = \sum_{i=1}^n [\delta(y_i = m) - P(Y = m | \tilde{x}_i)] \tilde{x}_i,$$

which has the same form as the binary gradient: the **residual** times the feature vector, summed over observations.

Regarding the factor of $1/n$: including $1/n$ gives the gradient of the *mean*^b cross-entropy^c, whereas omitting $1/n$ gives the gradient of the *summed* cross-entropy (the negative log-likelihood). Because the log-likelihood is a sum over observations and the empirical risk is an average, their gradients differ by the factor $1/n$. Multiplying an objective by a positive constant^d does not change its maximizer or its score equations, so the two conventions yield the same maximum likelihood^e estimate^f.

However, the factor of $1/n$ does affect step sizes and curvature. Including $1/n$ in the gradient rescales the effective step size by $1/n$, as analyzed in the stochastic-gradient exercises of the gradient descent chapter, and rescales the Hessian^g by $1/n$, so standard errors^h computed from the rescaled Hessian without accounting for n would be off by a factor of $\sqrt{n}$. Specifying whether the objective is the sum or the average determines whether $1/n$ appears.

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik>

^b<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

^c<https://morrison-lab.github.io/pds/expectation.html#def-cross-entropy>
^d<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>
^e<https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle>
^f<https://morrison-lab.github.io/sds/estimation.html#def-estimate>
^g<https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian>
^h<https://morrison-lab.github.io/sds/estimation.html#def-SE>

Exercise 12.8 (More than two classes (part 4)). Use the setup of Exercise 12.5. Check Exercise 12.7 against Exercise 6.2 by specializing to $C = 2$ under the convention you chose in Exercise 12.6.

Solution

Solution 12.8. For $C = 2$ with class 0 fixed at $B_0 = \vec{0}$ and $B_1 = \vec{\beta}$, Solution 12.5 gives $P(Y = 1 | x_i) = \pi_i$, and $\delta(y_i = 1) = y_i$ because $y_i \in \{0, 1\}$. The gradient from Solution 12.7 for $m = 1$ then becomes

$$\nabla_{\vec{\beta}} \ell = \sum_i (y_i - \pi_i) x_i = X^\top (y - \pi),$$

which matches Exercise 6.2 up to the sign convention between maximizing ℓ and minimizing $-\ell$. Once $B_0 = \vec{0}$ is fixed, the $m = 0$ score equation is redundant: its residuals are the negatives of the $m = 1$ residuals because the two fitted probabilities sum to 1 and the two class indicators sum to 1. This redundancy reflects the reduction from $C(p + 1)$ parameters to $(C - 1)(p + 1)$ identified parameters in Solution 12.6.

Exercise 12.9 (More than two classes (part 5)). Use the setup of Exercise 12.5. Setting Exercise 12.7 to zero gives C score equations. Assuming $\tilde{x}$ has an intercept^a component, state in words what the intercept equation for class m says, and compare it with Exercise 6.5.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

Solution

Solution 12.9. The intercept^a entry of the m th score equation is

$$\sum_{i=1}^n \delta(y_i = m) = \sum_{i=1}^n P(Y = m | x_i),$$

which states that the observed count in class m equals the sum of the fitted probabilities assigned to class m . The fitted model therefore matches the sample count of every class.

Exercise 6.5 is the $C = 2$ special case, where matching the count of class 1 automatically matches the count of class 0 because the two probabilities sum to 1. With C classes, the intercept score equations provide $C - 1$ independent^b aggregate calibration identities.

Aggregate calibration is a property of the score equations at the maximum likelihood^c estimate^d, not a guarantee of predictive accuracy: a model can match every class total while assigning inaccurate probabilities to individual observations, which is why calibration and discrimination are evaluated separately in the metrics chapter.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

^b<https://morrison-lab.github.io/pds/independence.html#def-indpt>

^c<https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle>

^d<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

Exercise 12.10 (More than two classes (part 6)). Use the setup of Exercise 12.5. Fit the intercept-only model to 100 observations falling 10/30/60 across three classes. Give the fitted probabilities, and the coefficients under the convention of Exercise 12.6 with class 1 as the reference.

Solution

Solution 12.10. With only an intercept^a, every $x_i = 1$ and the fitted probabilities p_m are constant^b across observations. The score equation from Solution 12.9 is then $n_m = 100p_m$, so

$$\hat{p}_1 = 0.1, \quad \hat{p}_2 = 0.3, \quad \hat{p}_3 = 0.6,$$

which are the sample proportions.

Fixing class 1 as the reference class sets $b_1 = 0$, and since $p_m/p_1 = e^{b_m}$, we obtain

$$\begin{aligned} \hat{b}_2 &= \log 3 & \hat{b}_3 &= \log 6 \\ &= 1.0986, & &= 1.7918. \end{aligned}$$

As a check,

$$\begin{aligned} e^0 + e^{\log 3} + e^{\log 6} &= 1 + 3 + 6 \\ &= 10, \end{aligned}$$

and dividing the numerators 1, 3, 6 by 10 recovers 0.1, 0.3, 0.6.

Each coefficient is the log-odds^c of its class relative to the reference class, as in Exercise 3.1, so its numerical value depends on the choice of reference class, as in Exercise 7.5. If class 3 is chosen as the reference class instead ($b_3 = 0$), the same fitted model has parameters $\hat{b}_1 = \log \frac{1}{6}$ and $\hat{b}_2 = \log \frac{1}{2}$, while all fitted probabilities remain unchanged.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

^b<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

^c<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

i Read

GeeksforGeeks explains the softmax function^a [23].

^a<https://www.geeksforgeeks.org/deep-learning/the-role-of-softmax-in-neural-networks-detailed-explanation-and-applications/>

Exercise 12.11 (A softmax with large scores). Consider four classes with the score vector $\tilde{z} = (23, -100, -15, 20)^\top$.

Exponentiate each score, give the normalizer $Z = \sum_k e^{z_k}$, and give the four probabilities $\text{softmax}(\tilde{z})$, each to two significant figures.

Solution

Solution 12.11. Exponentiating gives 9.74×10^9 , 3.72×10^{-44} , 3.06×10^{-7} , and 4.85×10^8 . Only the first and the last are comparable in magnitude to the total $Z = 1.023 \times 10^{10}$, so dividing through gives

$$\text{softmax}(\tilde{z}) = (0.95, 3.6 \times 10^{-54}, 3.0 \times 10^{-17}, 0.047)^\top.$$

Exercise 12.12 (A softmax with large scores (part 2)). Use the setup of Exercise 12.11.

Say what is preserved of the scores' ordering and what is preserved of their spacing, using the gap of 3 between the two leading scores and the gap of 38 between the leading score and the third. Say what that has to do with the name *softmax*.

Solution

Solution 12.12. The ordering of the scores is preserved because the exponential function is strictly increasing, but the relative spacing changes: differences in scores become ratios of probabilities. The difference of 3 between the two leading scores becomes a factor of $e^3 \approx 20$ between their probabilities, while the difference of 38 between the leading score and the third gives the third class a probability of 3×10^{-17} . The name “softmax” refers to this behavior: a smooth approximation to the indicator of the maximum, which at this spacing assigns nearly all probability to the largest score.

Exercise 12.13 (A softmax with large scores (part 3)). Use the setup of Exercise 12.11.

In double precision e^z overflows to infinity just below $z = 710$. Say what a direct implementation returns for the scores $\tilde{z} + 700$, and use Exercise 12.6 to give an implementation that cannot overflow.

Solution

Solution 12.13. Adding 700 to every score makes e^{723} and e^{720} exceed the floating-point overflow threshold, and the normalizer overflows with them. Direct evaluation then divides infinity by infinity for the first and last classes, returning NaN, and divides a finite number by infinity for the other two classes, returning 0. The exact probability vector equals the vector in Solution 12.11, because adding a constant^a to every score leaves all softmax probabilities unchanged by Exercise 12.6.

That shift invariance also prevents overflow. Before exponentiating, subtract the maximum score $\max_k z_k$ from every entry. This shift leaves the probabilities unchanged and sets the largest exponent to $e^0 = 1$, so no term can overflow, and because the normalizer is at least 1, it cannot underflow to 0. Standard numerical implementations apply this shift automatically.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

Exercise 12.14 (Penalizing the softmax weights). Collect the class vectors as the columns of a $(p+1) \times C$ matrix B , and index its rows by $r = 0, 1, \dots, p$, keeping i for observations. Row 0 holds the intercepts and rows 1 through p the feature weights. Leave the intercepts out of the penalty, as Exercise 5.19 argued for the binary model, and maximize

$$\ell(B) - \lambda \sum_{r=1}^p \sum_{c=1}^C B_{rc}^2.$$

Derive the gradient of this objective with respect to B_m , using Exercise 12.7 for the gradient of ℓ .

Solution

Solution 12.14. Differentiation is linear, so

$$\nabla_{B_m} \left[\ell(B) - \lambda \sum_{r=1}^p \sum_{c=1}^C B_{rc}^2 \right] = \nabla_{B_m} \ell(B) - \lambda \nabla_{B_m} \sum_{r=1}^p \sum_{c=1}^C B_{rc}^2,$$

and the first term was computed in Exercise 12.7. In the second gradient, only the terms from column m and rows $r \geq 1$ have nonzero derivatives with respect to B_m , giving

$$\nabla_{B_m} \sum_{r=1}^p \sum_{c=1}^C B_{rc}^2 = 2\tilde{B}_m, \quad \tilde{B}_m = (0, B_{1m}, \dots, B_{pm})^\top,$$

which is B_m with its intercept^a entry set to zero. Combining the two terms gives

$$\nabla_{B_m} = \sum_{i=1}^n [\delta(y_i = m) - P(Y = m \mid \tilde{x}_i)] \tilde{x}_i - 2\lambda \tilde{B}_m.$$

The penalty shrinks each non-intercept entry of B_m toward 0 in proportion to its magnitude, while leaving the intercept entry unpenalized.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

Exercise 12.15 (Penalizing the softmax weights (part 2)). Use the setup of Exercise 12.14. Exercise 12.6 showed that adding the same vector to every column leaves the likelihood^a unchanged. State which of those shifts change the penalty, which representative of each equivalence class the penalized objective selects, and whether the remaining unpenalized direction in the intercept^b row affects the predicted probabilities.

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik>

^b<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

Solution

Solution 12.15. Let the shift vector be $= (c_0, c_1, \dots, c_p)^\top$. The penalty depends only on the feature rows $r \geq 1$, where it equals $\sum_{r \geq 1} \sum_c (B_{rc} + c_r)^2$. Because this sum depends on $c_1, \dots, c_p$, the penalty distinguishes shifts that yield the same likelihood^a. Setting the derivative with respect to c_r to zero gives $\sum_c (B_{rc} + c_r) = 0$, so the penalized objective is maximized by the representative with minimum squared norm, in which each feature row sums to zero across the C classes.

Neither the likelihood nor the penalty depends on the intercept^b shift c_0 , so adding a constant^c to all C intercepts leaves the penalized objective unchanged. This invariance does not affect any fitted probability, but it leaves a one-dimensional subspace along which the objective is constant, so an optimizer that requires a unique maximizer must still fix one class intercept (for example, to 0) or constrain the intercepts to sum to zero.

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik>

^b<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

^c<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

i Watch

- Hastie and Tibshirani extend logistic regression to several classes, from 5:43^a [4], video 4.4.
- Hutchinson teaches multinomial logistic regression and the softmax^b (20 min) [1].
- Starmer explains argmax and softmax^c [24].

^a<https://www.youtube.com/watch?v=sYDDk6R-be0&t=343s>

^bhttps://facultyweb.cs.wvu.edu/~hutchib2/video_lectures/data371/#mult_log_regression

^c<https://www.youtube.com/watch?v=KpKog-L9veg>

13 Modelling the features: discriminant analysis

Logistic regression models the label given the features. An alternative is to model the distribution⁶⁵ of the features within each class, and compute the posterior probability of the label by Bayes' rule. Choosing a normal distribution⁶⁶ for the features of each class produces linear and [quadratic discriminant analysis](#), which yield linear or quadratic decision boundaries fitted by maximizing the joint⁶⁷ likelihood rather than the conditional likelihood.

Exercise 13.1 (Classifying with Bayes' rule). Logistic regression models $\Pr[Y = k \mid \tilde{x}]$ directly. Suppose instead that we knew the two ingredients of the joint distribution:

⁶⁵<https://morrison-lab.github.io/pds/random-variables.html#def-cdf>

⁶⁶<https://morrison-lab.github.io/pds/random-variables.html#def-normal>

⁶⁷<https://morrison-lab.github.io/pds/random-variables.html#def-joint-distribution>

- the class prior $\pi_k = \Pr[Y = k]$, for each class $k = 1, \dots, K$;
 - the class-conditional density $f_k(\tilde{x})$ of the features within class k .
- Use Bayes' rule to express $\Pr[Y = k \mid \tilde{x}]$ in terms of π_k and f_k .

Solution

Solution 13.1. Bayes' rule, with the law of total probability in the denominator, gives

$$\Pr[Y = k \mid \tilde{x}] = \frac{\pi_k f_k(\tilde{x})}{\sum_{l=1}^K \pi_l f_l(\tilde{x})}.$$

Exercise 13.2 (Classifying with Bayes' rule (part 2)). Use the setup of Exercise 13.1. To classify $\tilde{x}$, we select the class with the largest posterior probability. Which factor in your formula can be omitted when comparing classes, and why?

Solution

Solution 13.2. The denominator is the same for every k , since it is the density of $\tilde{x}$ with the class summed out. It is positive by the assumption of Theorem 13.1, so the division is defined. Dividing every posterior by one common positive number does not change which is largest. So the predicted class is

$$\hat{y}(\tilde{x}) = \arg \max_k \pi_k f_k(\tilde{x}),$$

and it is enough to compare the products $\pi_k f_k(\tilde{x})$.

Exercise 13.3 (Classifying with Bayes' rule (part 3)). Use the setup of Exercise 13.1. Where would π_k and f_k come from in practice?

Solution

Solution 13.3. Both are estimated^a from the training data. The proportion of training points in class k estimates^b π_k . Estimating f_k amounts to choosing a family of densities for the features within a class and fitting it to the training points of that class. Each choice of family gives a different classifier. The rest of this section uses the Gaussian family.

^a<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

^b<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

Theorem 13.1 (The posterior from a prior and class densities). *Let class k have prior $\pi_k > 0$ and feature density $f_k(\tilde{x})$, for $k = 1, \dots, K$, and let $\tilde{x}$ be a point where the marginal density $\sum_{l=1}^K \pi_l f_l(\tilde{x})$ is positive. Then*

$$\Pr[Y = k \mid \tilde{x}] = \frac{\pi_k f_k(\tilde{x})}{\sum_{l=1}^K \pi_l f_l(\tilde{x})},$$

and the class with the largest posterior is $\hat{y}(\tilde{x}) = \arg \max_k \pi_k f_k(\tilde{x})$.

i Proof

Proof. This is the solution to Exercise 13.1 and Exercise 13.2: see Solution 13.2. □

Example 13.1 (A posterior from two class densities). Two classes have priors $\pi_1 = 0.3$ and $\pi_2 = 0.7$. At the observed $\tilde{x}$ the class densities are $f_1(\tilde{x}) = 0.5$ and $f_2(\tilde{x}) = 0.1$.

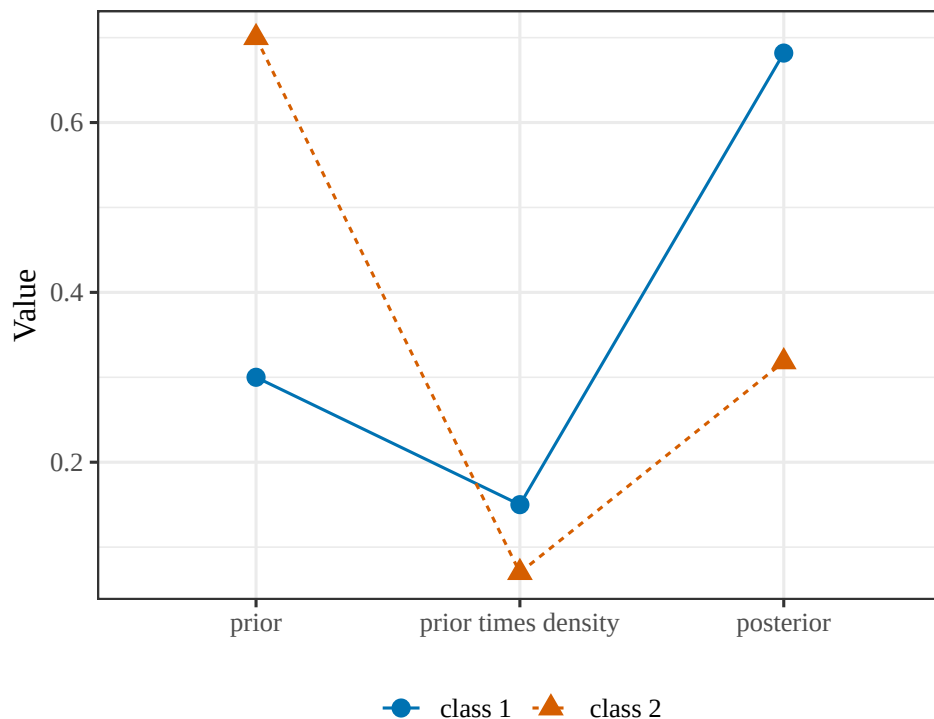
```
br_prior <- c(0.3, 0.7)
br_dens <- c(0.5, 0.1)
br_joint <- br_prior * br_dens
br_post <- br_joint / sum(br_joint)
rbind(joint = br_joint, posterior = br_post)
```

```
      [,1]      [,2]
joint 0.1500000 0.0700000
posterior 0.6818182 0.3181818
```

The products are 0.15 and 0.07, so by Theorem 13.1 the posteriors are 0.682 and 0.318. Class 1 has the larger posterior, although class 2 has the larger prior.

13.1 R

```
library(ggplot2)
bf_steps <- c("prior", "prior times density", "posterior")
bf <- data.frame(
  step = factor(rep(bf_steps, each = 2), levels = bf_steps),
  class = rep(c("class 1", "class 2"), times = 3),
  value = c(br_prior, br_joint, br_post)
)
ggplot(bf, aes(step, value, colour = class, shape = class, linetype = class, group = class)) +
  geom_line() +
  geom_point(size = 3) +
  scale_colour_manual(values = c("class 1" = "#0072B2", "class 2" = "#D55E00")) +
  labs(x = NULL, y = "Value", colour = NULL, shape = NULL, linetype = NULL)
```



(a)

13.2 Python

```
import matplotlib.pyplot as plt
import numpy as np

prior_bf = np.array([0.3, 0.7])
dens_bf = np.array([0.5, 0.1])
joint_bf = prior_bf * dens_bf
post_bf = joint_bf / joint_bf.sum()
steps_bf = ["prior", "prior times density", "posterior"]
values_bf = np.vstack([prior_bf, joint_bf, post_bf])
fig, ax = plt.subplots(figsize=(5, 4), layout="constrained")
for k, (label, colour, marker, style) in enumerate(
    [("class 1", "#0072B2", "o", "-"), ("class 2", "#D55E00", "^", "--")]
):
    ax.plot(steps_bf, values_bf[:, k], color=colour, marker=marker, markersize=8,
            linestyle=style, label=label)
ax.set_ylabel("Value")
fig.legend(loc="outside lower center", ncol=2)
plt.show()
```

Definition 13.1 (Generative classifier (plug-in Bayes classifier)). A **generative classifier** models the joint distribution of the label and the features through a class prior π_k and a class-conditional density $f_k(\underline{x})$ for each class k , fits both from data [25], and classifies by the largest posterior probability

$$\Pr[Y = k \mid \underline{x}] = \frac{\pi_k f_k(\underline{x})}{\sum_{l=1}^K \pi_l f_l(\underline{x})},$$

as in Theorem 13.1. A *discriminative classifier*, such as logistic regression, models $\Pr[Y = k \mid \underline{x}]$ directly and says nothing about how the features are distributed.

When π_k and f_k are the true distributions, the rule is called the *Bayes classifier*, and no classifier has a smaller error rate on new data [26, Ch. 2], [27]. A fitted generative classifier approximates it, so it is also called a **plug-in Bayes classifier**.

Example 13.2 (A Gaussian generative classifier for the stock market). Return to the **Smarket** data of Exercise 10.1. Model the previous day's return, **Lag1**, within each direction as a normal distribution, fitted to the days before 2005. Then classify a day whose previous return was 1%.

13.3 Python

```
import numpy as np
import pandas as pd
from scipy.stats import norm

da_py = pd.read_csv("data/smarket.csv")
da_train_py = da_py[da_py["Year"] < 2005]

by_dir_py = da_train_py.groupby("Direction")["Lag1"]
prior_py = by_dir_py.size() / len(da_train_py)
mean_py = by_dir_py.mean()
sd_py = by_dir_py.std()

x0_py = 1.0
joint_py = prior_py * norm.pdf(x0_py, mean_py, sd_py)
posterior_py = joint_py / joint_py.sum()
print(pd.DataFrame({"prior": prior_py, "mean": mean_py, "sd": sd_py,
                    "posterior": posterior_py}).round(4))
```

	prior	mean	sd	posterior
Direction				
Down	0.492	0.0428	1.2274	0.5059
Up	0.508	-0.0395	1.2317	0.4941

13.4 R

```

da <- read.csv("data/smarket.csv")
da_train <- da[da$Year < 2005, ]

prior <- prop.table(table(da_train$Direction))
class_mean <- tapply(da_train$Lag1, da_train$Direction, mean)
class_sd <- tapply(da_train$Lag1, da_train$Direction, sd)

x0 <- 1
joint <- prior * dnorm(x0, class_mean, class_sd)
posterior <- joint / sum(joint)
round(cbind(prior, mean = class_mean, sd = class_sd, posterior), 4)

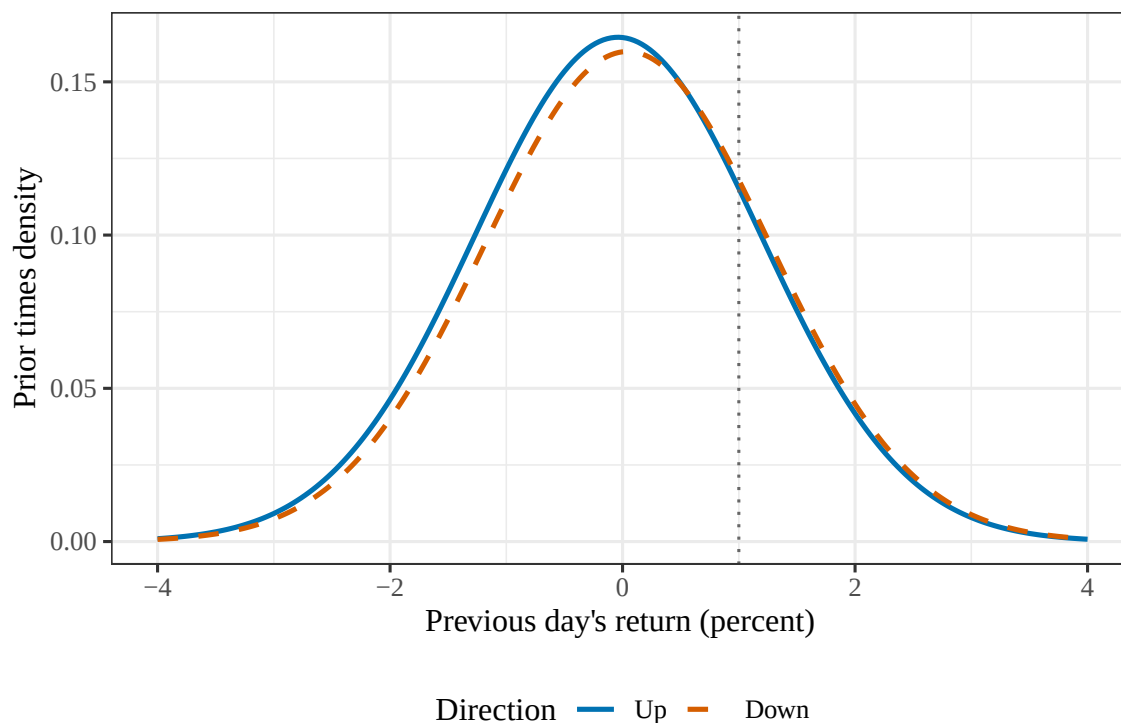
```

	prior	mean	sd	posterior
Down	0.492	0.0428	1.2274	0.5059
Up	0.508	-0.0395	1.2317	0.4941

The classes are nearly balanced, with 50.8% of the training days up. The mean of `Lag1` is -0.04 on days that went up and 0.043 on days that went down, and the standard deviations are 1.232 and 1.227, almost equal. A day that follows a 1% return has posterior probability 0.494 of going up, so the classifier predicts Down with almost no confidence. The two class-conditional densities overlap almost completely, which is the same difficulty that Figure 28 shows for logistic regression.

13.5 R

```
library(ggplot2)
gcf_grid <- seq(-4, 4, length.out = 241)
gcf_curves <- do.call(rbind, lapply(names(class_mean), function(dir) {
  data.frame(
    lag1 = gcf_grid,
    weighted = prior[[dir]] * dnorm(gcf_grid, class_mean[[dir]], class_sd[[dir]]),
    direction = dir
  )
}))
gcf_curves$direction <- factor(gcf_curves$direction, levels = c("Up", "Down"))
ggplot(gcf_curves, aes(lag1, weighted, colour = direction, linetype = direction)) +
  geom_line(linewidth = 0.9) +
  geom_vline(xintercept = x0, linetype = "dotted", colour = "grey40") +
  scale_colour_manual(values = c(Up = "#0072B2", Down = "#D55E00")) +
  scale_linetype_manual(values = c(Up = "solid", Down = "dashed")) +
  labs(
    x = "Previous day's return (percent)", y = "Prior times density",
    colour = "Direction", linetype = "Direction"
  )
)
```



(a)

13.6 Python

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import norm

grid_gcf = np.linspace(-4, 4, 241)
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
for direction, colour, style in [("Up", "#0072B2", "--"), ("Down", "#D55E00", "-")]:
    ax.plot(grid_gcf, prior_py[direction] * norm.pdf(grid_gcf, mean_py[direction], sd_py[direction]),
            color=colour, linestyle=style, linewidth=2, label=direction);
ax.axvline(x0_py, color="grey", linestyle=":", lw=2);
ax.set_xlabel("Previous day's return (percent)");
ax.set_ylabel("Prior times density");
```

Exercise 13.4 (Gaussian classes with one covariance matrix). Let $x \in \mathbb{R}^p$, and suppose that within class k the features are multivariate normal, $\tilde{x} \mid Y = k \sim N(\tilde{\mu}_k, \Sigma)$, with a mean $\tilde{\mu}_k$ for each class and one covariance matrix Σ shared by all of them. The density is

$$f_k(\tilde{x}) = \frac{1}{(2\pi)^{p/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\tilde{x} - \tilde{\mu}_k)^\top (\Sigma)^{-1} (\tilde{x} - \tilde{\mu}_k)\right).$$

Take the logarithm of $\pi_k f_k(\tilde{x})$, and expand the quadratic form.

Solution

Solution 13.4. Assume, as in Theorem 13.2, that Σ is positive definite and every $\pi_k > 0$. Then $(\Sigma)^{-1}$ exists, because a positive definite matrix is invertible^a, and $\log|\Sigma|$ exists, $\log \pi_k$ is finite, and every normal density is positive, so every product $\pi_k f_k(\tilde{x})$ is positive and has a logarithm. Take the logarithm of the product, which turns it into a sum:

$$\begin{aligned} \log(\pi_k f_k(\tilde{x})) &= \log \pi_k - \frac{p}{2} \log(2\pi) - \frac{1}{2} \log|\Sigma| - \frac{1}{2} (\tilde{x} - \tilde{\mu}_k)^\top (\Sigma)^{-1} (\tilde{x} - \tilde{\mu}_k) && \text{(log of a product, log of an exponential)} \\ &= \log \pi_k - \frac{p}{2} \log(2\pi) - \frac{1}{2} \log|\Sigma| - \frac{1}{2} \tilde{x}^\top (\Sigma)^{-1} \tilde{x} + \tilde{x}^\top (\Sigma)^{-1} \tilde{\mu}_k - \frac{1}{2} \tilde{\mu}_k^\top (\Sigma)^{-1} \tilde{\mu}_k && \text{(expand; } (\Sigma)^{-1} \text{ symmetric, so } \tilde{\mu}_k^\top (\Sigma)^{-1} \tilde{\mu}_k = \tilde{\mu}_k^\top (\Sigma)^{-1} \tilde{\mu}_k) \end{aligned}$$

^a<https://morrison-lab.github.io/mds/linear-algebra-diagonalization.html#thm-pd-invertible>

Exercise 13.5 (Gaussian classes with one covariance matrix (part 2)). Use the setup of Exercise 13.4. Drop every term that is the same for all classes, and denote the remaining expression^a by $\delta_k(\tilde{x})$. Why does dropping those terms preserve $\arg \max_k \pi_k f_k(\tilde{x})$?

^a<https://morrison-lab.github.io/mds/expressions-algorithms.html#def-expression>

Solution

Solution 13.5. Three terms, $-\frac{p}{2} \log(2\pi)$, $-\frac{1}{2} \log|\Sigma|$ and $-\frac{1}{2} \tilde{x}^\top (\Sigma)^{-1} \tilde{x}$, do not depend on k . Adding the same number to every class's score does not change which score is largest, so they can be dropped. The score that remains is

$$\delta_k(\tilde{x}) = \tilde{x}^\top (\Sigma)^{-1} \tilde{\mu}_k - \frac{1}{2} \tilde{\mu}_k^\top (\Sigma)^{-1} \tilde{\mu}_k + \log \pi_k.$$

Exercise 13.6 (Gaussian classes with one covariance matrix (part 3)). Use the setup of Exercise 13.4. Is $\delta_k(\tilde{x})$ linear in $\tilde{x}$?

Solution

Solution 13.6. Yes. The quadratic term $\tilde{x}^\top (\Sigma)^{-1} \tilde{x}$ is the same in every class and was dropped, which is possible only because the covariance matrix^a is shared. The score is a constant^b plus the dot product^c of $\tilde{x}$ with the fixed vector $(\Sigma)^{-1} \tilde{\mu}_k$.

^a<https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x>

^b<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

^c<https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-dot-product>

Theorem 13.2 (The linear discriminant function). Suppose that within class k the p features are normal, $x \mid Y = k \sim N(\vec{\mu}_k, \Sigma)$, with one covariance matrix^a Σ shared by all classes, which is positive definite^b, and prior $\pi_k > 0$ for every class. Then the marginal density $\sum_l \pi_l f_l(\tilde{x})$ is positive at every $\tilde{x}$, and $\arg \max_k \pi_k f_k(\tilde{x}) = \arg \max_k \delta_k(\tilde{x})$, where

$$\delta_k(\tilde{x}) = \tilde{x}^\top (\Sigma)^{-1} \vec{\mu}_k - \frac{1}{2} \vec{\mu}_k^\top (\Sigma)^{-1} \vec{\mu}_k + \log \pi_k.$$

Each δ_k is linear in $\tilde{x}$. More precisely, $\log(\pi_k f_k(\tilde{x})) = \delta_k(\tilde{x}) + c(\tilde{x})$, where $c(\tilde{x}) = -\frac{p}{2} \log(2\pi) - \frac{1}{2} \log|\Sigma| - \frac{1}{2} \tilde{x}^\top (\Sigma)^{-1} \tilde{x}$ is the same for every class.

^a<https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x>

^b<https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-positive-definite>

Proof

Proof. This is the solution to Exercise 13.4: see Solution 13.4. □

Example 13.3 (The linear discriminant with one feature). Let $p = 1$, with class means^a $\vec{\mu}_0 = 0$ and $\vec{\mu}_1 = 2$, shared variance^b $\Sigma = 1$, and priors $\pi_0 = 0.6$ and $\pi_1 = 0.4$. Classify the point $x = 1.2$.

```
ld_m <- c(0, 2)
ld_pi <- c(0.6, 0.4)
ld_x <- 1.2
ld_delta <- ld_x * ld_m - ld_m^2 / 2 + log(ld_pi)
ld_post <- ld_pi * dnorm(ld_x, ld_m, 1)
rbind(delta = ld_delta, pi_times_f = ld_post)
```

```
      [,1]      [,2]
delta  -0.5108256 -0.5162907
pi_times_f 0.1165116 0.1158766
```

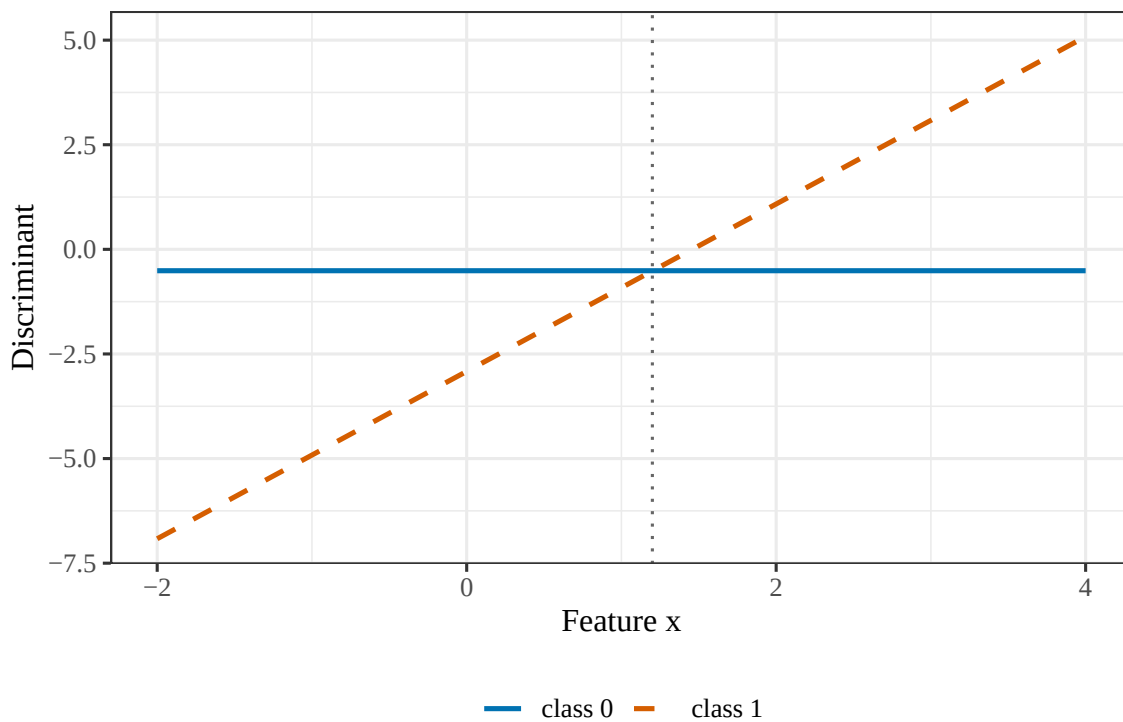
The discriminants are -0.511 and -0.516, so Theorem 13.2 predicts class 0. The products $\pi_k f_k(\tilde{x})$ choose the same class: 0.

^a<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

^b<https://morrison-lab.github.io/pds/variance-covariance.html#def-variance>

13.7 R

```
library(ggplot2)
ldf_grid <- seq(-2, 4, length.out = 121)
ldf_lines <- do.call(rbind, lapply(1:2, function(k) {
  data.frame(
    x = ldf_grid,
    delta = ldf_grid * ld_m[k] - ld_m[k]^2 / 2 + log(ld_pi[k]),
    class = paste("class", k - 1)
  )
}))
ggplot(ldf_lines, aes(x, delta, colour = class, linetype = class)) +
  geom_line(linewidth = 0.9) +
  geom_vline(xintercept = ld_x, linetype = "dotted", colour = "grey40") +
  scale_colour_manual(values = c("class 0" = "#0072B2", "class 1" = "#D55E00")) +
  scale_linetype_manual(values = c("class 0" = "solid", "class 1" = "dashed")) +
  labs(x = "Feature x", y = "Discriminant", colour = NULL, linetype = NULL)
```



(a)

13.8 Python

```
import matplotlib.pyplot as plt
import numpy as np

grid_ldf = np.linspace(-2, 4, 121)
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
for k, colour, style in [(0, "#0072B2", "-"), (1, "#D55E00", "--")]:
    ax.plot(grid_ldf, grid_ldf * r.ld_m[k] - r.ld_m[k] ** 2 / 2 + np.log(r.ld_pi[k]),
            color=colour, linestyle=style, linewidth=2, label=f"class {k}");
ax.axvline(r.ld_x, color="grey", linestyle=":");
ax.set_xlabel("Feature x");
ax.set_ylabel("Discriminant");
fig.legend(loc="outside lower center", ncol=2);
plt.show()
```

Definition 13.2 (Linear discriminant analysis (LDA)). *Linear discriminant analysis* is the generative classifier of Definition 13.1 with a multivariate normal density in each class and one covariance matrix Σ shared by all classes. It predicts the class with the largest *linear discriminant function* [28]

$$\delta_k(\tilde{x}) = \tilde{x}^\top (\Sigma)^{-1} \tilde{\mu}_k - \frac{1}{2} \tilde{\mu}_k^\top (\Sigma)^{-1} \tilde{\mu}_k + \log \pi_k,$$

which is Theorem 13.2. The parameters are estimated from n training points, n_k of them in class k :

- $\hat{\pi}_k = n_k/n$, the class proportion;
- $\hat{\mu}_k$, the average of the feature vectors in class k ;
- $\hat{\Sigma} = \frac{1}{n-K} \sum_{k=1}^K \sum_{i: y_i=k} (\tilde{x}_i - \hat{\mu}_k)(\tilde{x}_i - \hat{\mu}_k)^\top$, the *pooled* within-class covariance matrix.

Maximum likelihood divides the pooled sum by n rather than $n - K$. Software follows the unbiased $n - K$ version, as here. “Fisher’s linear discriminant” names a closely related method that finds the direction best separating the class means, and it gives the same directions in the two-class case.

Example 13.4 (LDA for one predictor). Fit LDA to the `Smarket` days before 2005 with `Lag1` as the only predictor, so $p = 1$ and Σ is a single variance. Compute the pooled variance and the discriminant functions at $x_0 = 1$ by hand, and compare the resulting posterior probabilities with a library’s.

13.9 Python

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

n_k_py = by_dir_py.size()
pooled_py = ((n_k_py - 1) * sd_py**2).sum() / (n_k_py.sum() - 2)
delta_py = x0_py * mean_py / pooled_py - mean_py**2 / (2 * pooled_py) + np.log(prior_py)
by_hand_py = np.exp(delta_py) / np.exp(delta_py).sum()

lda_py = LinearDiscriminantAnalysis().fit(da_train_py[["Lag1"]], da_train_py["Direction"])
library_py = lda_py.predict_proba(pd.DataFrame({"Lag1": [x0_py]}))[0]
print("pooled variance:", round(float(pooled_py), 4))

pooled variance: 1.5119
print("by hand:", by_hand_py.round(4).to_dict())

by hand: {'Down': 0.5056, 'Up': 0.4944}
print("library:", dict(zip(lda_py.classes_.tolist(), library_py.round(4).tolist()))))

library: {'Down': 0.5056, 'Up': 0.4944}
```

13.10 R

```
n_k <- table(da_train$Direction)
pooled <- sum((n_k - 1) * class_sd^2) / (sum(n_k) - 2)
delta <- x0 * class_mean / pooled - class_mean^2 / (2 * pooled) + log(prior)
by_hand <- exp(delta) / sum(exp(delta))

lda_fit <- MASS::lda(Direction ~ Lag1, data = da_train)
library_post <- predict(lda_fit, data.frame(Lag1 = x0))$posterior
cat("pooled variance:", round(pooled, 4), "\n")

pooled variance: 1.5119
round(rbind(by_hand = by_hand, library = library_post[1, ]), 4)

      Down      Up
by_hand 0.5056 0.4944
```

library 0.5056 0.4944

The pooled variance is 1.512, between the two class variances 1.507 and 1.517. The two discriminant functions differ by only 0.022 at x_0 , and the hand computation and the library agree in all four printed decimals. The posterior probability of Up is 0.4944, slightly under one half, as in Example 13.2. The small difference from that example's posterior comes from replacing the two class variances by their pool.

Exercise 13.7 (The log odds of LDA). Consider two classes, $Y \in \{0, 1\}$, under the LDA model of Definition 13.2. Compare with Exercise 3.1, where the log odds were *assumed* linear. Express $\log \frac{\Pr[Y=1|\tilde{x}]}{\Pr[Y=0|\tilde{x}]}$ in terms of the discriminant functions δ_1 and δ_0 .

Solution

Solution 13.7. Both posteriors share the denominator $\sum_l \pi_l f_l(\tilde{x})$ from Theorem 13.1, which cancels in the ratio. Then $\log(\pi_k f_k(\tilde{x})) = \delta_k(\tilde{x}) + c(\tilde{x})$, where $c(\tilde{x})$ collects the terms that Theorem 13.2 drops and is the same for both classes. So

$$\log \frac{\Pr[Y = 1 | \tilde{x}]}{\Pr[Y = 0 | \tilde{x}]} = \log \frac{\pi_1 f_1(\tilde{x})}{\pi_0 f_0(\tilde{x})} = \delta_1(\tilde{x}) - \delta_0(\tilde{x}).$$

Exercise 13.8 (The log odds of LDA (part 2)). Use the setup of Exercise 13.7. Show that it equals $\beta_0 + \vec{\beta} \cdot \tilde{x}$, and give β_0 and $\vec{\beta}$ in terms of the parameters of the model.

Solution

Solution 13.8. Subtract the two discriminant functions:

$$\begin{aligned} \delta_1(\tilde{x}) - \delta_0(\tilde{x}) &= \tilde{x}^\top (\Sigma)^{-1} (\vec{\mu}_1 - \vec{\mu}_0) - \frac{1}{2} (\vec{\mu}_1^\top (\Sigma)^{-1} \vec{\mu}_1 - \vec{\mu}_0^\top (\Sigma)^{-1} \vec{\mu}_0) + \log \frac{\pi_1}{\pi_0} \quad (\text{definition of } \delta_k) \\ &= \tilde{x}^\top (\Sigma)^{-1} (\vec{\mu}_1 - \vec{\mu}_0) - \frac{1}{2} (\vec{\mu}_1 + \vec{\mu}_0)^\top (\Sigma)^{-1} (\vec{\mu}_1 - \vec{\mu}_0) + \log \frac{\pi_1}{\pi_0} \quad (\text{difference of quadratic forms, } (\Sigma)^{-1} \text{ symmetric}) \end{aligned}$$

This has the form $\beta_0 + \vec{\beta} \cdot \tilde{x}$ with

$$\vec{\beta} = (\Sigma)^{-1} (\vec{\mu}_1 - \vec{\mu}_0), \quad \beta_0 = \log \frac{\pi_1}{\pi_0} - \frac{1}{2} (\vec{\mu}_1 + \vec{\mu}_0)^\top \vec{\beta}.$$

Exercise 13.9 (The log odds of LDA (part 3)). Use the setup of Exercise 13.7. What does this say about $\Pr[Y = 1 | \tilde{x}]$?

Solution

Solution 13.9. Theorem 3.3 turns a linear log odds^a into the probability, which gives

$$\Pr[Y = 1 | \tilde{x}] = \text{expit}(\beta_0 + \vec{\beta} \cdot \tilde{x}).$$

This is the [logistic regression model](#), with these particular coefficients.

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

Theorem 13.3 (LDA implies a logistic model). *Under the two-class LDA model of Definition 13.2, with Σ positive definite and $\pi_0, \pi_1 > 0$,*

$$\Pr[Y = 1 \mid \tilde{x}] = \text{expit}(\beta_0 + \vec{\beta} \cdot \tilde{x}), \quad \vec{\beta} = (\Sigma)^{-1}(\vec{\mu}_1 - \vec{\mu}_0), \quad \beta_0 = \log \frac{\pi_1}{\pi_0} - \frac{1}{2}(\vec{\mu}_1 + \vec{\mu}_0)^\top \vec{\beta}.$$

The decision boundary $\beta_0 + \vec{\beta} \cdot \tilde{x} = 0$ at the threshold 1/2 is a hyperplane with normal vector $\vec{\beta}$, as in Exercise 4.1.

i Proof

Proof. By Solution 13.7. □

Example 13.5 (LDA and logistic regression on the same days). For the one-predictor LDA fit of Example 13.4, compute β_0 and β_1 from Theorem 13.3, and compare them with the coefficients of a logistic regression of Up on Lag1 fitted to the same days.

13.11 Python

```
import statsmodels.formula.api as smf
import statsmodels.api as sm

b1_py = (mean_py["Up"] - mean_py["Down"]) / pooled_py
mid_py = (mean_py["Up"] + mean_py["Down"]) / 2
b0_py = np.log(prior_py["Up"] / prior_py["Down"]) - mid_py * b1_py

da_train_py = da_train_py.assign(up=(da_train_py["Direction"] == "Up").astype(int))
logit_py = smf.glm("up ~ Lag1", data=da_train_py, family=sm.families.Binomial()).fit()
print("LDA:      ", round(float(b0_py), 4), round(float(b1_py), 4))

LDA:      0.0322 -0.0545

print("logistic:", logit_py.params.round(4).tolist())

logistic: [0.0322, -0.0546]
```

13.12 R

```
b1 <- (class_mean[["Up"]] - class_mean[["Down"]]) / pooled
mid <- (class_mean[["Up"]] + class_mean[["Down"]]) / 2
b0 <- log(prior[["Up"]] / prior[["Down"]]) - mid * b1

logit_fit <- glm(
  I(Direction == "Up") ~ Lag1,
  family = binomial, data = da_train
)
rbind(LDA = c(b0, b1), logistic = coef(logit_fit)) |> round(4)

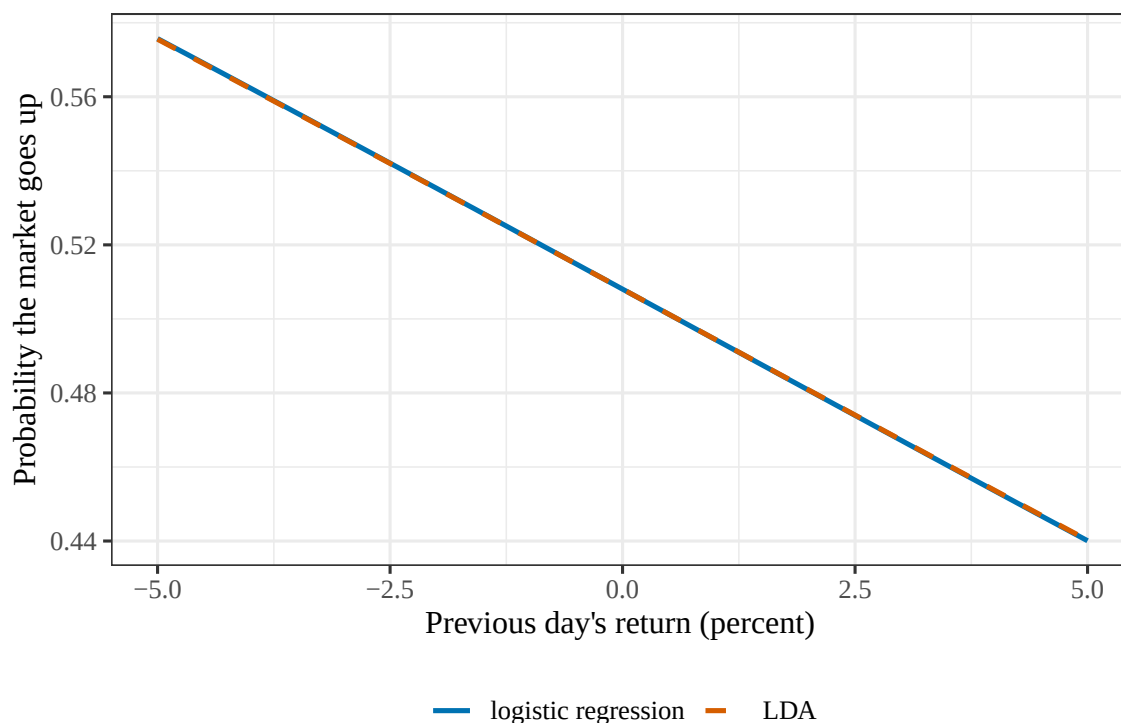
      (Intercept)      Lag1
LDA      0.0322 -0.0545
logistic  0.0322 -0.0546
```

The two lines nearly coincide: LDA gives a slope of -0.0545 and logistic regression -0.0546, a difference of 0.3% of the logistic slope, and the intercepts are 0.0322 and 0.0322. Both slopes are negative, but the slope is tiny next to its scale, since a one-point rise in yesterday's return moves the log odds by about 0.05. The closeness is typical of data that are close to normal with equal variances in the two classes, which is the situation LDA assumes. The fits can differ more when the assumption fails, for example for

a strongly skewed predictor.

13.13 R

```
library(ggplot2)
llo_grid <- seq(-5, 5, length.out = 201)
llo_curves <- data.frame(
  lag1 = rep(llo_grid, 2),
  prob = c(plogis(b0 + b1 * llo_grid), plogis(coef(logit_fit)[[1]] + coef(logit_fit)[[2]] * llo_grid))
  model = rep(c("LDA", "logistic regression"), each = length(llo_grid))
)
llo_curves$model <- factor(llo_curves$model, levels = c("logistic regression", "LDA"))
ggplot(llo_curves, aes(lag1, prob, colour = model, linetype = model)) +
  geom_line(linewidth = 0.9) +
  scale_colour_manual(values = c("logistic regression" = "#0072B2", LDA = "#D55E00")) +
  scale_linetype_manual(values = c("logistic regression" = "solid", LDA = "dashed")) +
  labs(
    x = "Previous day's return (percent)", y = "Probability the market goes up",
    colour = NULL, linetype = NULL
  )
)
```



(a)

13.14 Python

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.special import expit

grid_llo = np.linspace(-5, 5, 201)
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.plot(grid_llo, expit(logit_py.params.iloc[0] + logit_py.params.iloc[1] * grid_llo),
        color="#0072B2", linestyle="-", linewidth=2, label="logistic regression");
ax.plot(grid_llo, expit(b0_py + b1_py * grid_llo),
        color="#D55E00", linestyle="--", linewidth=2, label="LDA");
ax.set_xlabel("Previous day's return (percent)");
ax.set_ylabel("Probability the market goes up");
fig.legend(loc="outside lower center", ncol=2);
plt.show()
```

Exercise 13.10 (Same form, different fitting criterion). By Theorem 13.3, two-class LDA and logistic regression give a posterior probability of the same functional form. Each method chooses its parameters to maximize a likelihood^a. State the quantity LDA maximizes and the quantity logistic regression maximizes, in terms of π_k , f_k and $\Pr[Y = 1 \mid \tilde{x}]$.

^a<https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik>

Solution

Solution 13.10. LDA maximizes the *joint* likelihood of the labels and the features,

$$\prod_{i=1}^n \pi_{y_i} f_{y_i}(\tilde{x}_i),$$

and logistic regression maximizes the *conditional* likelihood of the labels given the features,

$$\prod_{i=1}^n \Pr[Y = y_i \mid \tilde{x}_i].$$

The LDA estimates are the closed-form class proportions, class means and pooled covariance of Definition 13.2. The logistic estimates need the iterative methods of this chapter. Writing the joint likelihood as the conditional likelihood times $\prod_i \Pr(\tilde{x}_i)$ shows that LDA also fits the marginal distribution of the features, which logistic regression ignores.

Exercise 13.11 (Same form, different fitting criterion (part 2)). Use the setup of Exercise 13.10. Does a logistic model with log odds^a linear in $\tilde{x}$ imply that the classes are Gaussian with a common covariance matrix^b? Give a counterexample or a proof.

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

^b<https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x>

Solution

Solution 13.11. No. Consider a single binary feature $X \in \{0, 1\}$ with $\Pr[X = 1 \mid Y = k] = q_k$. The log odds^a of $Y = 1$ are $\log \frac{\pi_1}{\pi_0} + X \log \frac{q_1}{q_0} + (1 - X) \log \frac{1-q_1}{1-q_0}$, which is linear in X , even though X is not Gaussian. Many other families of class-conditional densities also yield linear log odds, so [logistic regression](#) makes weaker distributional assumptions than [LDA](#).

^a<https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds>

Exercise 13.12 (Same form, different fitting criterion (part 3)). Use the setup of Exercise 13.10. Which method is more affected when the features are not Gaussian, and which uses the data more efficiently when they are?

Solution

Solution 13.12. When the Gaussian assumption fails, [LDA](#)'s estimates^a are fitted to a misspecified density, and its boundary can have higher error than [logistic regression](#)'s. When the Gaussian assumption holds, LDA is asymptotically more efficient, because it estimates the full joint distribution^b of features and labels. In practice the two methods often produce similar boundaries [29, Sec. 4.4.5].

^a<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

^b<https://morrison-lab.github.io/pds/random-variables.html#def-joint-distribution>

Exercise 13.13 (Same form, different fitting criterion (part 4)). Use the setup of Exercise 13.10. Recall Exercise 11.1. What happens to the LDA parameters when the two classes are perfectly separated?

Solution

Solution 13.13. The parameter estimates^a remain finite. The sample class means^b and the pooled sample covariance matrix^c are well-defined and finite whenever the pooled covariance matrix is invertible^d, so LDA produces a finite decision boundary even when the classes are linearly separable and the logistic regression coefficients diverge (Exercise 11.1).

^a<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

^b<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

^c<https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x>

^d<https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-matrix-inverse>

Exercise 13.14 (Gaussian classes with their own covariance matrices). Repeat Exercise 13.4, but let each class have its own covariance matrix: $\tilde{x} \mid Y = k \sim N(\tilde{\mu}_k, \Sigma_k)$. Which terms of $\log(\pi_k f_k(\tilde{x}))$ can still be dropped because they are the same for all classes?

Solution

Solution 13.14. Assume, as in Theorem 13.4, that every Σ_k is positive definite and every $\pi_k > 0$, so that each $(\Sigma_k)^{-1}$ exists, because a positive definite matrix is invertible^a, and each $\log|\Sigma_k|$ and $\log \pi_k$ exists and every product $\pi_k f_k(\tilde{x})$ is positive. Only $-\frac{p}{2} \log(2\pi)$. The term $-\frac{1}{2} \log|\Sigma_k|$ now depends on k , and so does the quadratic form $(\tilde{x} - \tilde{\mu}_k)^\top (\Sigma_k)^{-1} (\tilde{x} - \tilde{\mu}_k)$, because Σ_k is inside it. In particular $-\frac{1}{2} \tilde{x}^\top (\Sigma_k)^{-1} \tilde{x}$ no longer cancels between classes.

^a<https://morrison-lab.github.io/mds/linear-algebra-diagonalization.html#thm-pd-invertible>

Exercise 13.15 (Gaussian classes with their own covariance matrices (part 2)). Use the setup of Exercise 13.14. Give the remaining discriminant function $\delta_k(\tilde{x})$.

Solution

Solution 13.15. With the constant^a dropped,

$$\delta_k(\tilde{x}) = -\frac{1}{2} \log|\Sigma_k| - \frac{1}{2} (\tilde{x} - \tilde{\mu}_k)^\top (\Sigma_k)^{-1} (\tilde{x} - \tilde{\mu}_k) + \log \pi_k.$$

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

Exercise 13.16 (Gaussian classes with their own covariance matrices (part 3)). Use the setup of Exercise 13.14. Is it linear in $\tilde{x}$?

Solution

Solution 13.16. It is not linear. It contains $-\frac{1}{2} \tilde{x}^\top (\Sigma_k)^{-1} \tilde{x}$, a quadratic form in $\tilde{x}$.

Exercise 13.17 (Gaussian classes with their own covariance matrices (part 4)). Use the setup of Exercise 13.14.

What geometric surface is the set $\{x : \delta_k(x) = \delta_l(x)\}$ where two classes have equal discriminant scores?

Solution

Solution 13.17. The difference $\delta_k(x) - \delta_l(x)$ is a polynomial^a of degree^b two in the coordinates of x , so the boundary $\{x : \delta_k(x) = \delta_l(x)\}$ is a quadric surface: an ellipse, a parabola, a hyperbola, or a pair of lines when $p = 2$, and a hyperplane if $\Sigma_k = \Sigma_l$.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-polynomial>

^b<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-polynomial-degree>

Exercise 13.18 (Gaussian classes with their own covariance matrices (part 5)). Use the setup of Exercise 13.14.

How many such points can there be when $p = 1$?

Solution

Solution 13.18. When $p = 1$, the condition $\delta_k(x) = \delta_l(x)$ is a quadratic equation in x , so the boundary consists of at most two points.

Theorem 13.4 (The quadratic discriminant function). *Suppose that within class k the features are normal, $x | Y = k \sim N(\tilde{\mu}_k, \Sigma_k)$, with its own covariance matrix Σ_k , which is positive definite, and prior $\pi_k > 0$ for every class. Then the marginal density $\sum_l \pi_l f_l(x)$ is positive at every x , and $\arg \max_k \pi_k f_k(x) = \arg \max_k \delta_k(x)$, where*

$$\delta_k(x) = -\frac{1}{2} \log |\Sigma_k| - \frac{1}{2} (x - \tilde{\mu}_k)^\top (\Sigma_k)^{-1} (x - \tilde{\mu}_k) + \log \pi_k.$$

Each δ_k is quadratic in x , so the set where two classes have equal discriminant scores is a quadric surface.

i Proof

Proof. This is the solution to Exercise 13.14: see Solution 13.14. □

Example 13.6 (The quadratic discriminant with one feature). Let $p = 1$, with two classes that have the same mean^a $\tilde{\mu}_0 = \tilde{\mu}_1 = 0$, variances^b $\Sigma_0 = 1$ and $\Sigma_1 = 4$, and equal priors. Classify $x = 0.5$ and $x = 3$.

```
qd_var <- c(1, 4)
qd_delta <- function(x) -log(qd_var) / 2 - x^2 / (2 * qd_var) + log(0.5)
qd_x <- c(0.5, 3)
rbind(
  x = qd_x,
  class_by_delta = vapply(qd_x, \ (x) which.max(qd_delta(x)) - 1, numeric(1)),
  class_by_density = vapply(qd_x, \ (x) which.max(0.5 * dnorm(x, 0, sqrt(qd_var))) - 1, numeric(1))
)
```

```
      [,1] [,2]
x      0.5   3
class_by_delta 0.0 1
class_by_density 0.0 1
```

By Theorem 13.4 the discriminant and the density rule agree. The narrower class has the larger discriminant near the center and the wider class has the larger discriminant far from it, so $\delta_0 = \delta_1$ at

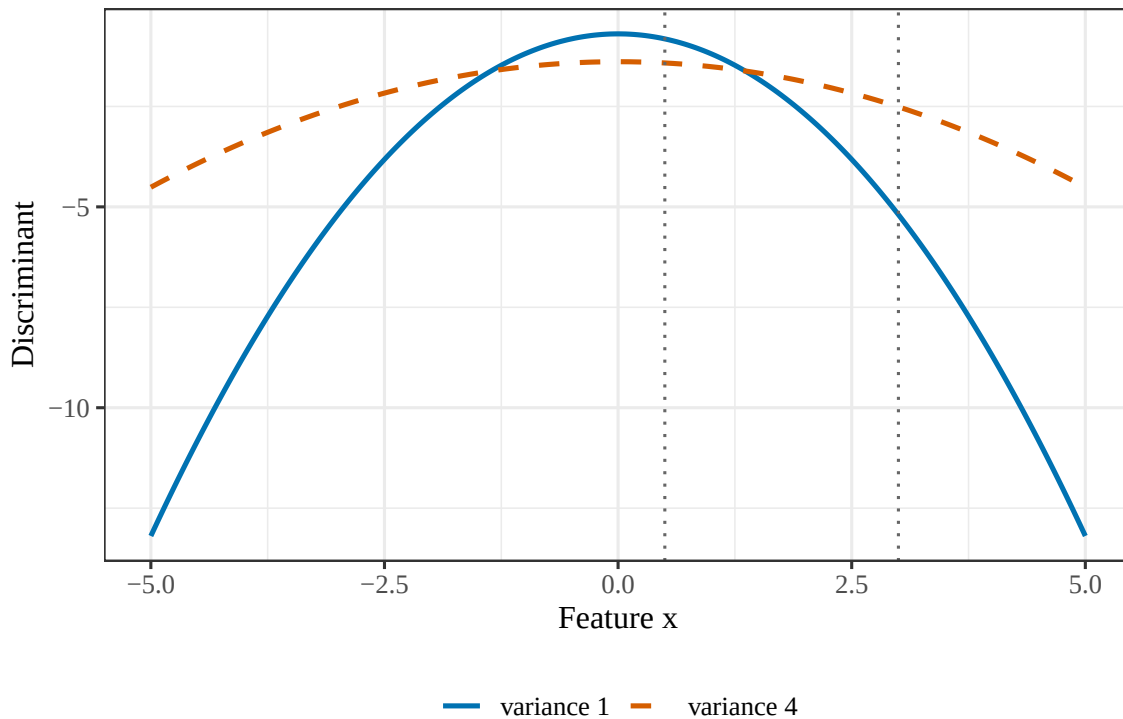
two points, the roots of a quadratic equation in x . No linear discriminant could give this rule, because the two classes share a mean.

^a<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

^b<https://morrison-lab.github.io/pds/variance-covariance.html#def-variance>

13.15 R

```
library(ggplot2)
qdf_grid <- seq(-5, 5, length.out = 241)
qdf_curves <- data.frame(
  x = rep(qdf_grid, 2),
  delta = c(sapply(qdf_grid, \(x) qd_delta(x)[1]), sapply(qdf_grid, \(x) qd_delta(x)[2])),
  class = rep(c("variance 1", "variance 4"), each = length(qdf_grid))
)
ggplot(qdf_curves, aes(x, delta, colour = class, linetype = class)) +
  geom_line(linewidth = 0.9) +
  geom_vline(xintercept = qd_x, linetype = "dotted", colour = "grey40") +
  scale_colour_manual(values = c("variance 1" = "#0072B2", "variance 4" = "#D55E00")) +
  scale_linetype_manual(values = c("variance 1" = "solid", "variance 4" = "dashed")) +
  labs(x = "Feature x", y = "Discriminant", colour = NULL, linetype = NULL)
```



(a)

13.16 Python

```
import matplotlib.pyplot as plt
import numpy as np

var_qdf = np.asarray(r.qd_var)
grid_qdf = np.linspace(-5, 5, 241)
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
for k, colour, style in [(0, "#0072B2", "-"), (1, "#D55E00", "--")]:
    delta_qdf = -np.log(var_qdf[k]) / 2 - grid_qdf ** 2 / (2 * var_qdf[k]) + np.log(0.5)
    ax.plot(grid_qdf, delta_qdf, color=colour, linestyle=style, linewidth=2,
            label=f"variance {var_qdf[k]:.0f}");
for x_qdf in np.atleast_1d(r.qd_x):
    ax.axvline(x_qdf, color="grey", linestyle=":");
ax.set_xlabel("Feature x");
ax.set_ylabel("Discriminant");
fig.legend(loc="outside lower center", ncol=2);
plt.show()
```

Definition 13.3 (Quadratic discriminant analysis (QDA)). *Quadratic discriminant analysis* is the generative classifier of Definition 13.1 with a multivariate normal density in each class and a separate covariance matrix Σ_k for each class. It predicts the class with the largest *quadratic discriminant function* [30]

$$\delta_k(\tilde{x}) = -\frac{1}{2} \log |\Sigma_k| - \frac{1}{2} (\tilde{x} - \tilde{\mu}_k)^\top (\Sigma_k)^{-1} (\tilde{x} - \tilde{\mu}_k) + \log \pi_k,$$

which is Theorem 13.4. The priors and means are estimated as in Definition 13.2. Each $\hat{\Sigma}_k$ is the sample covariance matrix of the training points in class k alone, with divisor $n_k - 1$. LDA is the special case of QDA with $\Sigma_1 = \dots = \Sigma_K$.

Example 13.7 (The quadratic boundary for one predictor). Give the two directions of the Smarket days before 2005 their own variances^a, as in Example 13.2, and find where the Up and Down discriminant functions are equal. Setting $\delta_{\text{Up}}(x) = \delta_{\text{Down}}(x)$ and collecting terms gives a quadratic equation in x .

13.17 Python

```
def delta_qda_py(x, k):
    return (-np.log(sd_py[k]) - (x - mean_py[k]) ** 2 / (2 * sd_py[k] ** 2)
            + np.log(prior_py[k]))

# delta_Up - delta_Down is quadratic in x; fit its three coefficients exactly
grid_py = np.array([-1.0, 0.0, 1.0])
coef_py = np.polyfit(grid_py, delta_qda_py(grid_py, "Up") - delta_qda_py(grid_py, "Down"), 2)
roots_py = np.sort(np.roots(coef_py))
print("roots:", roots_py.round(3))

roots: [ 0.539 23.441]

for x in [-1.0, 1.0, 30.0]:
    up = delta_qda_py(x, "Up") > delta_qda_py(x, "Down")
    print(x, "->", "Up" if up else "Down")

-1.0 -> Up
1.0 -> Down
30.0 -> Up
```

13.18 R

```
delta_qda <- function(x, k, mean, sd, prior) {
  -log(sd[[k]]) - (x - mean[[k]])^2 / (2 * sd[[k]]^2) + log(prior[[k]])
}

delta_up <- function(x) delta_qda(x, "Up", class_mean, class_sd, prior)
delta_down <- function(x) delta_qda(x, "Down", class_mean, class_sd, prior)

# delta_Up - delta_Down is quadratic in x; fit its three coefficients exactly
grid <- c(-1, 0, 1)
gap <- delta_up(grid) - delta_down(grid)
roots <- sort(Re(polyroot(coef(lm(gap ~ grid + I(grid^2))))))
cat("roots:", round(roots, 3), "\n")

roots: 0.539 23.441

for (x in c(-1, 1, 30)) {
  pred_class <- if (delta_up(x) > delta_down(x)) "Up" else "Down"
  cat(x, "->", pred_class, "\n")
}
```

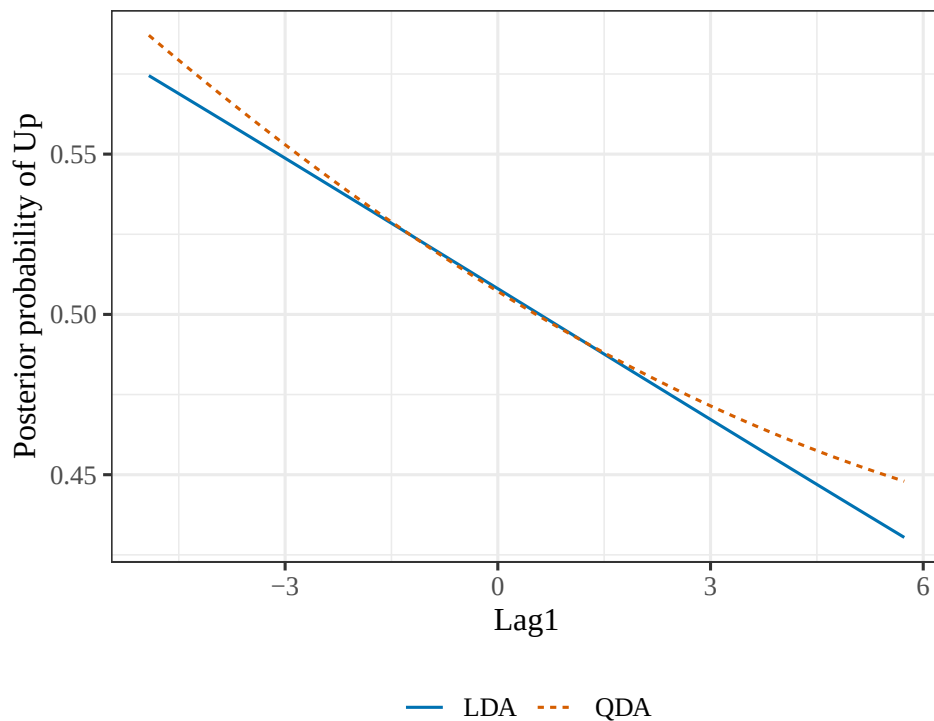
```
-1 -> Up  
1 -> Down  
30 -> Up
```

The two discriminant functions are equal at $x = 0.54$ and $x = 23.44$. The first root is in the middle of the data, where the returns range from -4.9 to 5.7. The second root is far outside that range, so within any realistic range of **Lag1** the boundary is a single point, as for LDA. The curvature enters only because the Up variance is slightly larger than the Down variance. With the variances this close the quadratic coefficient is tiny, and QDA reduces almost exactly to LDA.

^a<https://morrison-lab.github.io/pds/variance-covariance.html#def-variance>

13.19 R

```
library(ggplot2)
lq_x <- seq(min(da_train$Lag1), max(da_train$Lag1), length.out = 300)
lq_post <- function(delta_fun) {
  up <- delta_fun(lq_x, "Up")
  down <- delta_fun(lq_x, "Down")
  exp(up) / (exp(up) + exp(down))
}
lq_lda <- function(x, k) x * class_mean[[k]] / pooled - class_mean[[k]]^2 / (2 * pooled) + log(prior[[k]])
lq_qda <- function(x, k) delta_qda(x, k, class_mean, class_sd, prior)
lq <- data.frame(
  lag1 = rep(lq_x, 2),
  posterior = c(lq_post(lq_lda), lq_post(lq_qda)),
  method = rep(c("LDA", "QDA"), each = length(lq_x))
)
ggplot(lq, aes(lag1, posterior, colour = method, linetype = method)) +
  geom_line() +
  scale_colour_manual(values = c(LDA = "#0072B2", QDA = "#D55E00")) +
  labs(x = "Lag1", y = "Posterior probability of Up", colour = NULL, linetype = NULL)
```



(a)

13.20 Python

```
import matplotlib.pyplot as plt
import numpy as np

x_lq = np.linspace(da_train_py["Lag1"].min(), da_train_py["Lag1"].max(), 300)

def post_lq(delta_up, delta_down):
    return np.exp(delta_up) / (np.exp(delta_up) + np.exp(delta_down))

def delta_lda_py(x, k):
    return x * mean_py[k] / pooled_py - mean_py[k] ** 2 / (2 * pooled_py) + np.log(prior_py[k])

fig, ax = plt.subplots(figsize=(5, 4), layout="constrained")
ax.plot(x_lq, post_lq(delta_lda_py(x_lq, "Up"), delta_lda_py(x_lq, "Down")),
```

Exercise 13.19 (Counting parameters). Consider a classification problem with K classes and p features. A symmetric^a $p \times p$ matrix has $p(p+1)/2$ free entries. Count the free parameters of LDA (Definition 13.2).

^a<https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-symmetric-matrix>

Solution

Solution 13.19. Every generative model has $K - 1$ free priors, since the priors sum to one, and Kp class means^a. The models differ in the covariance matrices.

LDA has one covariance matrix^b: $(K - 1) + Kp + p(p + 1)/2$ parameters.

^a<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

^b<https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x>

Exercise 13.20 (Counting parameters (part 2)). Use the setup of Exercise 13.19. Count the free parameters of QDA (Definition 13.3).

Solution

Solution 13.20. QDA has K of them: $(K - 1) + Kp + K p(p + 1)/2$ parameters.

Exercise 13.21 (Counting parameters (part 3)). Use the setup of Exercise 13.19. Count the free parameters of the naive Bayes assumption of the probability chapter with a normal density for each feature in each class. This makes each class covariance matrix^a diagonal^b.

^a<https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x>

^b<https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-diagonal-matrix>

Solution

Solution 13.21. Gaussian naive Bayes has K diagonal^a matrices, which is Kp variances^b: $(K - 1) + 2Kp$ parameters.

^a<https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-diagonal-matrix>

^b<https://morrison-lab.github.io/pds/variance-covariance.html#def-variance>

Exercise 13.22 (Counting parameters (part 4)). Use the setup of Exercise 13.19. Count the free parameters of multinomial logistic regression with $K - 1$ coefficient vectors, each with an intercept^a.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept>

Solution

Solution 13.22. Multinomial logistic regression has $(K - 1)(p + 1)$ parameters.

Exercise 13.23 (Counting parameters (part 5)). Use the setup of Exercise 13.19. Which counts grow fastest in p ?

Solution

Solution 13.23. As p grows, the LDA count grows like $p^2/2$, and the QDA count like $Kp^2/2$. The naive Bayes and logistic counts grow only linearly in p . Each QDA covariance matrix^a must be estimated^b from the n_k points of its own class, so QDA's variance^c is the largest of these models when classes are small.

^a<https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x>

^b<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

^c<https://morrison-lab.github.io/pds/variance-covariance.html#def-variance>

Theorem 13.5 (Parameter counts of the classifiers). *With K classes and p features, the number of free parameters is*

Model	Parameters
LDA (Definition 13.2)	$(K - 1) + Kp + p(p + 1)/2$
QDA (Definition 13.3)	$(K - 1) + Kp + Kp(p + 1)/2$
Gaussian naive Bayes	$(K - 1) + 2Kp$
Multinomial logistic regression	$(K - 1)(p + 1)$

i Proof

Proof. This is the solution to Exercise 13.19: see Solution 13.19. □

Example 13.8 (Parameters for two classes). Count the parameters for $K = 2$ classes and several numbers of features, including the $p = 2$ of the Lag1–Lag2 model fitted later in this section and the $p = 6$ of Exercise 10.1.

13.21 Python

```
K_py = 2
D_py = np.array([2, 6, 50, 100])
counts_py = pd.DataFrame({
    "D": D_py,
    "LDA": (K_py - 1) + K_py * D_py + D_py * (D_py + 1) // 2,
    "QDA": (K_py - 1) + K_py * D_py + K_py * D_py * (D_py + 1) // 2,
    "Gaussian naive Bayes": (K_py - 1) + 2 * K_py * D_py,
    "logistic": (K_py - 1) * (D_py + 1),
})
print(counts_py.to_string(index=False))
```

D	LDA	QDA	Gaussian naive Bayes	logistic
2	8	11	9	3
6	34	55	25	7
50	1376	2651	201	51
100	5251	10301	401	101

13.22 R

```

k_classes <- 2
d_features <- c(2, 6, 50, 100)
counts <- data.frame(
  D = d_features,
  LDA = (k_classes - 1) + k_classes * d_features +
    d_features * (d_features + 1) / 2,
  QDA = (k_classes - 1) + k_classes * d_features +
    k_classes * d_features * (d_features + 1) / 2,
  `Gaussian naive Bayes` = (k_classes - 1) + 2 * k_classes * d_features,
  logistic = (k_classes - 1) * (d_features + 1),
  check.names = FALSE
)
counts

```

```

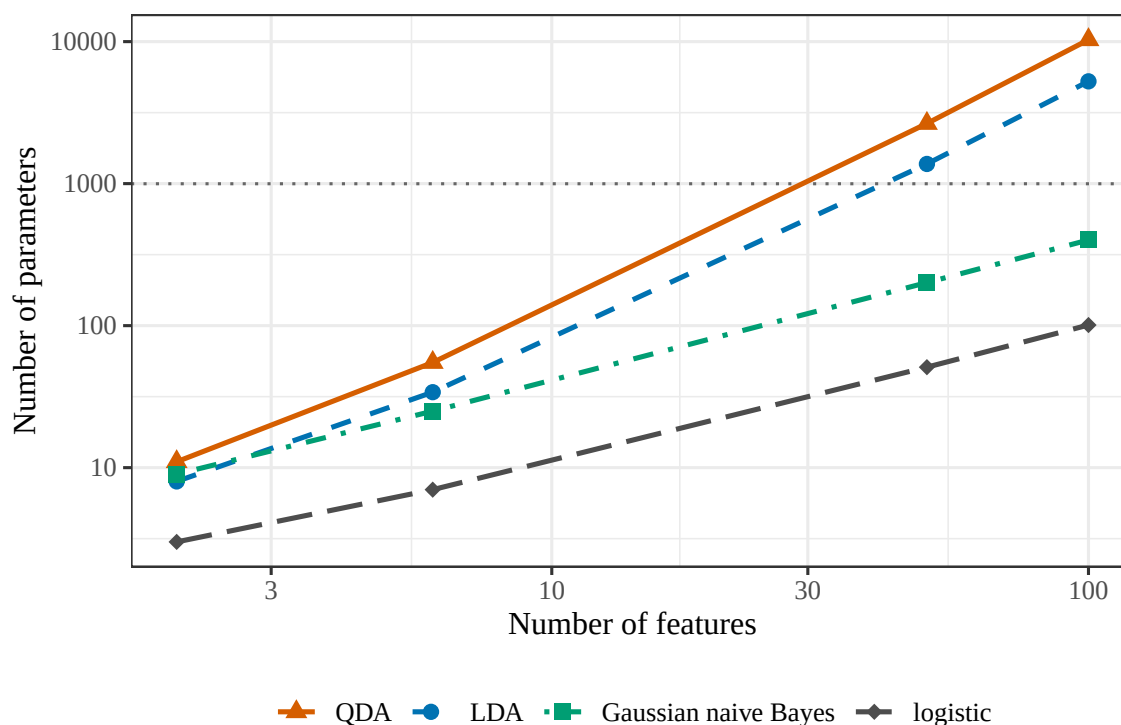
# A tibble: 4 x 5
      D    LDA    QDA `Gaussian naive Bayes` logistic
  <dbl> <dbl> <dbl>          <dbl>      <dbl>
1     2     8    11              9         3
2     6    34    55             25         7
3    50  1376  2651            201        51
4   100  5251 10301            401       101

```

At $p = 50$ the QDA model has 2651 parameters and LDA has 1376, against 51 for logistic regression. The Smarket training set has 998 days, which is 0.38 observations per QDA parameter at $p = 50$. At $p = 100$ QDA has 10,301 parameters, more than the 998 training days.

13.23 R

```
library(ggplot2)
dpf_long <- reshape(
  counts, direction = "long", idvar = "D",
  varying = list(c("LDA", "QDA", "Gaussian naive Bayes", "logistic")),
  v.names = "parameters", timevar = "method",
  times = c("LDA", "QDA", "Gaussian naive Bayes", "logistic")
)
dpf_long$method <- factor(dpf_long$method, levels = c("QDA", "LDA", "Gaussian naive Bayes", "logistic"))
ggplot(dpf_long, aes(D, parameters, colour = method, linetype = method, shape = method)) +
  geom_hline(yintercept = nrow(da_train), linetype = "dotted", colour = "grey40") +
  geom_line(linewidth = 0.9) +
  geom_point(size = 2.5) +
  scale_x_log10() +
  scale_y_log10() +
  scale_colour_manual(values = c(
    QDA = "#D55E00", LDA = "#0072B2", `Gaussian naive Bayes` = "#009E73", logistic = "grey30"
  )) +
  scale_linetype_manual(values = c(
    QDA = "solid", LDA = "dashed", `Gaussian naive Bayes` = "dotdash", logistic = "longdash"
  )) +
  scale_shape_manual(values = c(QDA = 17, LDA = 16, `Gaussian naive Bayes` = 15, logistic = 18)) +
  labs(x = "Number of features", y = "Number of parameters", colour = NULL, linetype = NULL, shape = NULL)
```



(a)

13.24 Python

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.axhline(len(da_train_py), color="grey", linestyle=":");
for method, colour, style, marker in [
    ("QDA", "#D55E00", "-", "^"), ("LDA", "#0072B2", "--", "o"),
    ("Gaussian naive Bayes", "#009E73", "-.", "s"), ("logistic", "#4d4d4d", (0, (8, 3)), "D"),
]:
    ax.plot(counts_py["D"], counts_py[method], color=colour, linestyle=style, marker=marker,
```

Exercise 13.24 (LDA and QDA on the stock market). Use the `Smarket` days of Exercise 10.1. Fit LDA and QDA for the direction of the market with `Lag1` and `Lag2` as the only predictors, on the days before 2005. Predict each day of 2005 and tabulate the predictions against what happened. What are the fitted class proportions and class means^a?

^a<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean>

Solution

Solution 13.24.

13.25 Python

```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

features_py = ["Lag1", "Lag2"]
da_test_py = da_py[da_py["Year"] == 2005]
lda2_py = LinearDiscriminantAnalysis().fit(da_train_py[features_py], da_train_py["Direction"])
qda2_py = QuadraticDiscriminantAnalysis().fit(da_train_py[features_py], da_train_py["Direction"])

print("priors:", dict(zip(lda2_py.classes_.tolist(), lda2_py.priors_.round(3).tolist())))
priors: {'Down': 0.492, 'Up': 0.508}
print(pd.DataFrame(lda2_py.means_, index=lda2_py.classes_, columns=features_py).round(3))
```

	Lag1	Lag2
Down	0.043	0.034
Up	-0.040	-0.031

```
for name, model in [("LDA", lda2_py), ("QDA", qda2_py)]:
    pred = model.predict(da_test_py[features_py])
    acc_py = float((pred == da_test_py["Direction"]).mean())
    # A 95% interval for the accuracy: the proportion plus or minus 1.96 standard errors.
    half_py = 1.96 * np.sqrt(acc_py * (1 - acc_py) / len(da_test_py))
    print(name, "accuracy:", round(acc_py, 3), [round(float(acc_py - half_py), 3), round(float(acc_py + half_py), 3)])
    print(pd.crosstab(pred, da_test_py["Direction"], rownames=["predicted"], colnames=["actual"])))

LDA accuracy: 0.56 [0.498, 0.621]
actual      Down  Up
predicted
Down         35  35
Up           76 106
QDA accuracy: 0.599 [0.539, 0.66]
actual      Down  Up
predicted
Down         30  20
Up           81 121
```

13.26 R

```
da_test <- da[da$Year == 2005, ]
lda2 <- MASS::lda(Direction ~ Lag1 + Lag2, data = da_train)
qda2 <- MASS::qda(Direction ~ Lag1 + Lag2, data = da_train)

lda2$prior
```

	Down	Up
	0.491984	0.508016

```
lda2$means
```

```
          Lag1      Lag2
Down 0.04279022 0.03389409
Up   -0.03954635 -0.03132544
```

```
pred_lda <- predict(lda2, da_test)$class
pred_qda <- predict(qda2, da_test)$class
table(LDA = pred_lda, actual = da_test$Direction)
```

```
      actual
LDA   Down Up
Down   35  35
Up     76 106
```

```
table(QDA = pred_qda, actual = da_test$Direction)
```

```
      actual
QDA   Down Up
Down   30  20
Up     81 121
```

```
acc <- c(
  LDA = mean(pred_lda == da_test$Direction),
  QDA = mean(pred_qda == da_test$Direction),
  always_up = mean(da_test$Direction == "Up")
)
round(acc, 3)
```

```
      LDA      QDA always_up
0.560    0.599    0.560
```

```
# 95% intervals for each accuracy, and for the paired differences between rules
# scored on the same days (wald_ci and paired_ci come from the logistic regression above).
ci_acc <- sapply(acc, wald_ci, n = nrow(da_test))
hit_lda <- pred_lda == da_test$Direction
hit_qda <- pred_qda == da_test$Direction
hit_up <- da_test$Direction == "Up"
ci_qda_up <- paired_ci(hit_qda, hit_up)
ci_qda_lda <- paired_ci(hit_qda, hit_lda)
```

The classes are balanced, with 50.8% of the training days up. The means of both lags are slightly negative on days that went up and slightly positive on days that went down, -0.04 and 0.043 for **Lag1**, a gap of 0.08 against a standard deviation of about 1.23. The classes overlap almost completely.

Exercise 13.25 (LDA and QDA on the stock market (part 2)). Use the setup of Exercise 13.24. How often is each model right on the 2005 days? Compare with always predicting Up, and with the [logistic regression](#) of Solution 10.2.

Solution

Solution 13.25. LDA is right on 56% of the 252 days in 2005 (95% interval [49.8, 62.1]%), QDA on 59.9% ([53.9, 66]%), and always predicting Up on 56% ([49.8, 62.1]%). LDA does no better than the constant^a rule, and the [logistic regression](#) of Solution 10.2, with more predictors, was right on only 48%. QDA exceeds the accuracy of the constant rule by 4 percentage points.

^a<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

Exercise 13.26 (LDA and QDA on the stock market (part 3)). Use the setup of Exercise 13.24. How much weight does the accuracy difference between the two models deserve, given that it comes from one year of data?

Solution

Solution 13.26. Little. Each interval above is the accuracy plus or minus 1.96 standard errors^a, of about 3.1 percentage points for QDA on 252 days, and the intervals for the three rules overlap almost entirely. The rules are scored on the same days, so the sharper comparison is day by day: QDA exceeds the accuracy of the constant^b rule by 4 percentage points (95% interval $[-1.5, 9.5]$), which includes 0, and exceeds that of LDA by 4 percentage points ($[0.5, 7.4]$), which excludes 0. These intervals reflect sampling variation across the test days in 2005 rather than across the training set, and they do not account for comparing several rules on the same days before reporting the best. Establishing that QDA has higher out-of-sample accuracy would require more test days or repeated splits.

^a<https://morrison-lab.github.io/sds/estimation.html#def-SE>

^b<https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function>

```

library(ggplot2)
grid <- expand.grid(
  Lag1 = seq(-4, 4, length.out = 120),
  Lag2 = seq(-4, 4, length.out = 120)
)
grid_post <- rbind(
  transform(grid, model = "LDA", p_up = predict(lda2, grid)$posterior[, "Up"]),
  transform(grid, model = "QDA", p_up = predict(qda2, grid)$posterior[, "Up"])
)
ggplot(grid_post) +
  aes(Lag1, Lag2) +
  geom_raster(aes(fill = p_up)) +
  geom_contour(aes(z = p_up), breaks = 0.5, colour = "black") +
  geom_point(
    data = da_test,
    aes(shape = Direction),
    colour = "grey20", alpha = 0.7
  ) +
  scale_fill_gradient2(midpoint = 0.5, limits = c(0, 1), name = "P(Up)") +
  facet_wrap(~model) +
  coord_cartesian(xlim = c(-4, 4), ylim = c(-4, 4))

```

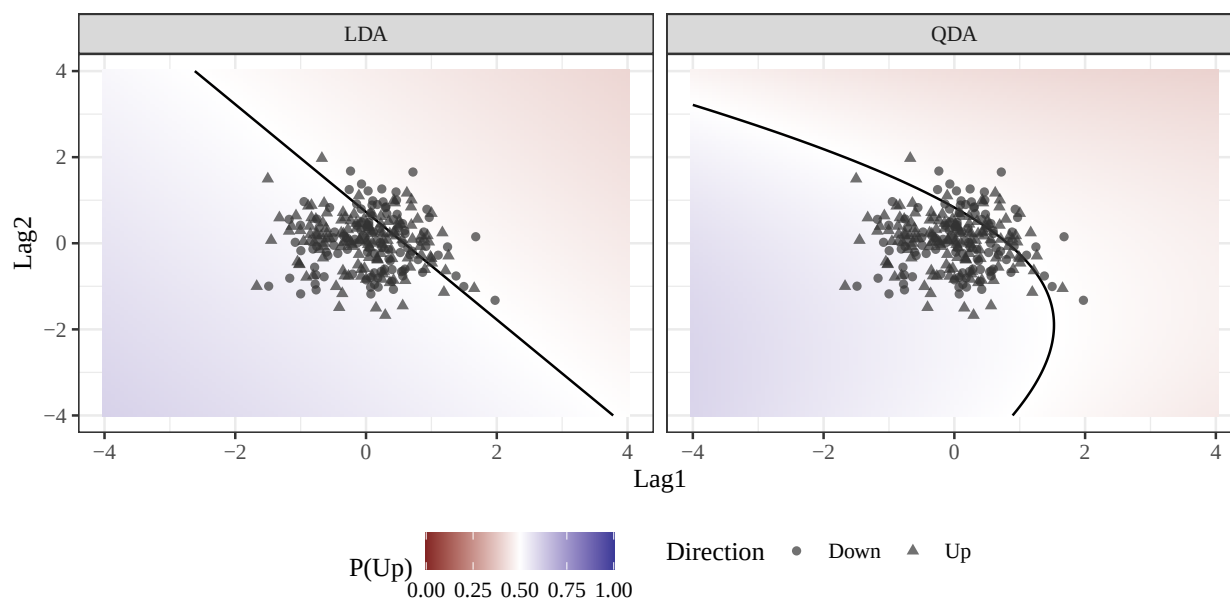


Figure 39: Predicted probability of an up day from the LDA and QDA fits of Solution 13.26, over the plane of the two lagged returns. The black line and curve are the decision boundaries, where the probability is one half. The points are the days of 2005.

Exercise 13.27 (Choosing among the classifiers). LDA, QDA, Gaussian naive Bayes and logistic regression are all linear or quadratic classifiers fitted to the same data. For each situation, say which you would try first, and why.
There are $n = 60$ observations and $p = 40$ features.

Solution

Solution 13.27. LDA, Gaussian naive Bayes or a regularized version of logistic regression. The pooled covariance matrix of LDA has 820 free entries, estimated from 58 degrees of freedom of within-class variation, and with about 30 points per class a QDA class covariance matrix is singular, since that is fewer than $p = 40$ points (a sample covariance matrix from $n \leq p$ points is singular^a). The diagonal^b covariance^c of naive Bayes has only p variances per class, so it is the most stable.

^a<https://morrison-lab.github.io/mds/linear-algebra-rank-nullity.html#cor-sample-covariance-singular>

^b<https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-diagonal-matrix>

^c<https://morrison-lab.github.io/pds/variance-covariance.html#def-cov>

Exercise 13.28 (Choosing among the classifiers (part 2)). Use the setup of Exercise 13.27. There are $n = 100,000$ observations and $p = 3$ features, and a plot shows that the within-class spread of the features differs a lot between classes.

Solution

Solution 13.28. QDA. With 3 features QDA has 19 parameters and LDA 13, by Theorem 13.5, and each is estimated^a from thousands of points. The unequal spreads are the structure that LDA's single covariance matrix^b cannot represent, so LDA is biased here and QDA is not.

^a<https://morrison-lab.github.io/sds/estimation.html#def-estimate>

^b<https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x>

Exercise 13.29 (Choosing among the classifiers (part 3)). Use the setup of Exercise 13.27. Some features are binary indicators, such as whether a word appears in an email.

Solution

Solution 13.29. Not LDA or QDA. Their density is Gaussian, which is a poor model of a binary feature. [Logistic regression](#) handles binary features without a model for them, and naive Bayes can use a Bernoulli density for each one, as in the probability chapter.

Exercise 13.30 (Choosing among the classifiers (part 4)). Use the setup of Exercise 13.27. How is the choice a bias-variance trade-off?

Solution

Solution 13.30. Moving from naive Bayes to LDA to QDA adds parameters, so it lowers bias^a when the classes really differ in their covariances^b and raises variance^c. LDA accepts bias from forcing one covariance matrix^d on all classes in exchange for variance that grows only with one matrix. QDA is the better choice when n is large relative to the number of parameters of Theorem 13.5, and LDA when it is small [26, Ch. 4]. Regularized discriminant analysis interpolates between the two by shrinking each $\hat{\Sigma}_k$ toward the pooled $\hat{\Sigma}$ [29, Sec. 4.3].

^a<https://morrison-lab.github.io/sds/estimation.html#def-bias>

^b<https://morrison-lab.github.io/pds/variance-covariance.html#def-cov>

^c<https://morrison-lab.github.io/pds/variance-covariance.html#def-variance>

^d<https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x>

i Read

James et al. [26], Chapter 4, derives these classifiers and compares them with logistic regression. Hastie et al. [29], Section 4.3, covers discriminant analysis, including its regularized versions.

14 Further reading

- James et al. [26], Chapter 4
- Hastie et al. [29], Section 4.4
- Zhou [31] — Biostat M257 slides on the Newton step and Fisher scoring, with a fraud-detection example; the statistical treatment is on the Inference for Logistic Regression⁶⁹ page of the statistics notes
- Zhou [32], Classification⁷⁰ — slides on *An Introduction to Statistical Learning*, Chapter 4, from logistic regression to discriminant analysis and ROC curves, with code in both R and Python
- Starmer [33], on logistic regression — an illustrated walk-through with little algebra
- Weissman [34] — the website of Stanford’s information theory course EE 276, which covers entropy and mutual information and lists readings⁷¹; the entropy page⁷² of the *Information Theory for Data Science* notes defines surprisal, conditional entropy, and mutual information

i Watch

The Data Science Learning Community’s book clubs discuss *An Introduction to Statistical Learning*, Chapter 4, in recorded meetings:

- the R book club, in part 1^a and part 2^b [35];
- the Python book club, in part 1^c and part 2^d [36].

^a<https://www.youtube.com/watch?v=noEW2H2N-s&list=PL3x6DOfs2NGjlpKMiWsDcKL2nngwXROq>

^b<https://www.youtube.com/watch?v=qF6BR3R9XeM&list=PL3x6DOfs2NGjlpKMiWsDcKL2nngwXROq>

^chttps://www.youtube.com/watch?v=W2_Al4g89hg&list=PL3x6DOfs2NGgbSNh0UG87rrM-z5LiKLZp

^d<https://www.youtube.com/watch?v=UpNBjSLhIDE&list=PL3x6DOfs2NGgbSNh0UG87rrM-z5LiKLZp>

References

- [1] B. Hutchinson, “DATA 471/571 (machine learning) and CSCI 481/581 (deep learning) video lectures.” Western Washington University. Accessed: Sep. 28, 2026. [Online]. Available: https://facultyweb.cs.wvu.edu/~hutchib2/video_lectures/data371/
- [2] R. H. Rosenman, R. J. Brand, C. D. Jenkins, M. Friedman, R. Straus, and M. Wurm, “Coronary heart disease in the Western Collaborative Group Study: Final follow-up experience of 8 1/2 years,” *JAMA*, vol. 233, no. 8, pp. 872–877, 1975, doi: 10.1001/jama.1975.03260080034016⁷³.
- [3] C. M. Bishop, *Pattern recognition and machine learning*. in Information science and statistics. New York: Springer, 2006.
- [4] T. Hastie, R. Tibshirani, and J. Taylor, “Statistical learning with python.” Stanford Online, lecture videos on YouTube, 2023. Available: <https://www.youtube.com/playlist?list=PLoROMvodv4rPP6braWoRt5UCXYZ71GZIQ>
- [5] J. Starmer, “Logistic regression.” StatQuest, video on YouTube, 2018. Available: <https://www.youtube.com/watch?v=yIYKR4sgzI8>
- [6] GeeksforGeeks, “Logistic regression in machine learning.” GeeksforGeeks, tutorial article, 2026. Available: <https://www.geeksforgeeks.org/machine-learning/understanding-logistic-regression/>

⁶⁹<https://morrison-lab.github.io/sds/logistic-inference.html>

⁷⁰<https://ucla-biostat-212a.github.io/2026winter/slides/04-classification/classification.html>

⁷¹<https://web.stanford.edu/class/ee276/reading.html>

⁷²<https://morrison-lab.github.io/ids/entropy.html>

⁷³<https://doi.org/10.1001/jama.1975.03260080034016>

- [7] GeeksforGeeks, “Sigmoid function.” GeeksforGeeks, tutorial article, 2025. Available: <https://www.geeksforgeeks.org/machine-learning/derivative-of-the-sigmoid-function/>
- [8] N. Jaques, “CSE 446/546: Machine learning, lecture slides.” University of Washington, 2026. Accessed: Oct. 12, 2026. [Online]. Available: <https://courses.cs.washington.edu/courses/cse446/26sp/schedule/lectures/>
- [9] A. Ng *et al.*, “CS229: Machine learning, autumn 2018.” Stanford Online, lecture videos on YouTube, 2018. Available: <https://www.youtube.com/playlist?list=PLoROMvodv4rMiGQp3WXShtMGgzqpfVfbU>
- [10] K. Weinberger, “CS4780: Machine learning for intelligent systems, spring 2017.” Cornell University, lecture videos on YouTube, 2017. Available: <https://www.youtube.com/playlist?list=PLl8OIHZGYOQ7bkVbuRthEsaLr7bONzbXS>
- [11] J. Starmer, “Odds and log(odds), clearly explained.” StatQuest, video on YouTube, 2018. Available: <https://www.youtube.com/watch?v=ARfXDSkQf1Y>
- [12] J. Starmer, “Odds ratios and log(odds ratios), clearly explained.” StatQuest, video on YouTube, 2018. Available: <https://www.youtube.com/watch?v=8nm0G-1uJzA>
- [13] J. Starmer, “Logistic regression details Pt1: coefficients.” StatQuest, video on YouTube, 2018. Available: <https://www.youtube.com/watch?v=vN5cNN2-HWE>
- [14] GeeksforGeeks, “Log odds.” GeeksforGeeks, tutorial article, 2022. Available: <https://www.geeksforgeeks.org/machine-learning/log-odds/>
- [15] Wikipedia contributors, “Cross-entropy — Wikipedia, the free encyclopedia.” 2026. Available: <https://en.wikipedia.org/wiki/Cross-entropy>
- [16] C. E. Shannon, “A mathematical theory of communication,” *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948, doi: 10.1002/j.1538-7305.1948.tb01338.x⁷⁴.
- [17] J. Starmer, “Logistic regression details, part 2: Maximum likelihood.” StatQuest, video on YouTube, 2018. Available: <https://www.youtube.com/watch?v=BfKan1aSG0>
- [18] J. Starmer, “Neural networks part 6: Cross entropy.” StatQuest, video on YouTube, 2021. Available: <https://www.youtube.com/watch?v=6ArSys5qHAU>
- [19] GeeksforGeeks, “Binary cross entropy/log loss for binary classification.” GeeksforGeeks, tutorial article, 2025. Available: <https://www.geeksforgeeks.org/deep-learning/binary-cross-entropy-log-loss-for-binary-classification/>
- [20] GeeksforGeeks, “Newton’s method in machine learning.” GeeksforGeeks, tutorial article, 2025. Available: <https://www.geeksforgeeks.org/machine-learning/newtons-method-in-machine-learning/>
- [21] G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor, *An introduction to statistical learning: With applications in Python*. Springer, 2023. doi: 10.1007/978-3-031-38747-0⁷⁵.
- [22] S. Rosset, J. Zhu, and T. Hastie, “Boosting as a regularized path to a maximum margin classifier,” *Journal of Machine Learning Research*, vol. 5, pp. 941–973, 2004, Available: <https://www.jmlr.org/papers/volume5/rosset04a/rosset04a.pdf>
- [23] GeeksforGeeks, “Softmax activation function in neural networks.” GeeksforGeeks, tutorial article, 2025. Available: <https://www.geeksforgeeks.org/deep-learning/the-role-of-softmax-in-neural-networks-detailed-explanation-and-applications/>
- [24] J. Starmer, “Neural networks part 5: ArgMax and SoftMax.” StatQuest, video on YouTube, 2021. Available: <https://www.youtube.com/watch?v=KpKog-L9veg>
- [25] Wikipedia contributors, “Generative model — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Generative_model
- [26] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning with applications in R*, 2nd ed. Springer, 2021. Available: <https://www.statlearning.com/>
- [27] Wikipedia contributors, “Bayes classifier — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Bayes_classifier
- [28] Wikipedia contributors, “Linear discriminant analysis — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Linear_discriminant_analysis
- [29] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: Data mining, inference, and prediction*, 2nd ed. Springer, 2009. Available: <https://hastie.su.domains/ElemStatLearn/>

⁷⁴<https://doi.org/10.1002/j.1538-7305.1948.tb01338.x>

⁷⁵<https://doi.org/10.1007/978-3-031-38747-0>

- [30] Wikipedia contributors, “Quadratic classifier — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Quadratic_classifier
- [31] H. Zhou, “Newton’s method and Fisher scoring.” Lecture notes for Biostat M257, Statistical Computing, University of California, Los Angeles, 2025. Available: <https://ucla-biostat-257.github.io/2025spring/slides/23-newton/newton.html>
- [32] J. Zhou, “Biostat 212A: Statistical machine learning a, winter 2026.” UCLA, course webpage with lecture slides, 2026. Available: <https://ucla-biostat-212a.github.io/2026winter/>
- [33] J. Starmer, *The StatQuest illustrated guide to machine learning*. StatQuest Publications, 2022. Available: <https://statquest.org/statquest-store/>
- [34] T. Weissman, “EE 276, information theory: Winter 2025–26 course website.” Course website, 2026. Available: <https://web.stanford.edu/class/ee276/>
- [35] Data Science Learning Community, “Introduction to statistical learning using R (islr06).” Book club meeting recordings on YouTube, 2023. Available: <https://www.youtube.com/playlist?list=PL3x6DOfs2NGjlpKMiWsDcKL2nngwXROq>
- [36] Data Science Learning Community, “Introduction to statistical learning using Python (islp03).” Book club meeting recordings on YouTube, 2025. Available: <https://www.youtube.com/playlist?list=PL3x6DOfs2NGgbSNh0UG87rrM-z5LiKLZp>