Fitting Logistic Regression

Last modified: 2026-11-19

```
# The Python chunks import these packages; reticulate provisions them.
library(reticulate) # nolint: undesirable_function_linter
py_packages <- c(
  "numpy", "pandas", "statsmodels", "scipy", "scikit-learn", "matplotlib",
  "plotly"
)
py_require(py_packages)
```

Source

This material was a section of the logistic regression page. It carries the same sources: Brian Hutchinson's Fall 2025 lecture notes, Kameron Decker Harris' 2022 weekly notes and Logan Sizemore's Spring 2025 lectures, as described there.

1 Fitting the model by maximum likelihood

The form of the model is now fixed, and the parameter vector $\vec{\beta}$ remains to be estimated[1]. Logistic regression is a probability[2] model, so maximum likelihood[3] gives a criterion for choosing $\vec{\beta}$: the value that makes the observed labels most probable.

Maximum likelihood[4] also determines the loss function. Exercise 1.1, Exercise 1.2 and Exercise 1.3 derive that loss, Exercise 1.5 and Exercise 1.8 show why squared-error loss on the probabilities is unsuitable, and Exercise 1.7 graphs[5] one observation[6]'s contribution to the log-likelihood[7] for each label.

Exercise 1.1 (Probability of a binary outcome in one formula)**.** The log-odds exercise specified the *model*: $\Pr[Y = 1 \mid \underset{\sim}{x}] = \operatorname{expit}\left(\vec{\beta} \cdot \underset{\sim}{x}\right)$. It did not specify how to estimate[a] $\vec{\beta}$. Let $\pi_i = \operatorname{expit}\left(\vec{\beta} \cdot \underset{\sim}{x}_i\right)$ be the probability that the model assigns to observation i having label 1.
Express $\Pr[Y_i = y_i \mid \underset{\sim}{x}_i]$ 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 1.1.

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

At $y_i = 1$, the second factor is $(1 - \pi_i)^0 = 1$, so the expression[a] equals π_i. At $y_i = 0$, the first factor

[1] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[2] https://morrison-lab.github.io/pds/probability-basics.html#def-probability
[3] https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle
[4] https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle
[5] https://morrison-lab.github.io/mcs/functions.html#def-graph
[6] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-observation
[7] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik

is $\pi_i^0 = 1$, so the expression equals $1 - \pi_i$. Each value is the probability that the model gives to that outcome, so the single expression covers both cases.
Each exponent is either 0 or 1, so one factor equals 1, and the expression equals the factor for the observed outcome.

[a] https://morrison-lab.github.io/mcs/expressions-algorithms.html#def-expression

Lemma 1.1 (Probability mass of one Bernoulli outcome)**.** *If* $Y_i \mid \underset{\sim}{x}_i \sim \text{Bernoulli}(\pi_i)$ *and* $y_i \in \{0, 1\}$*, then*

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

Proof

Proof. By Solution 1.1. □

Example 1.1 (The probability mass of one outcome)**.** Let $\pi_i = 0.8$. Lemma 1.1 gives the probabilities of the two outcomes, which equal the values from R's `dbinom` and SciPy's `bernoulli.pmf`:

1.0.1 R

```
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
```

1.0.2 Python

```
from scipy.stats import bernoulli

p_bpmf = 0.8
print([p_bpmf**1 * (1 - p_bpmf)**0, p_bpmf**0 * (1 - p_bpmf)**1])
```

```
[0.8, 0.19999999999999996]
```

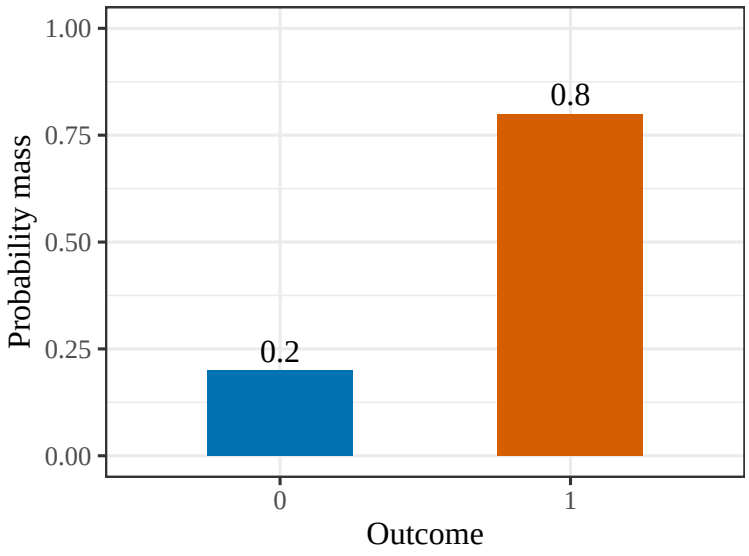
```
print(bernoulli.pmf([1, 0], p_bpmf))
```

```
[0.8 0.2]
```

Figure 1 plots the two values.

1.0.3 R

```r
bpmf_outcomes <- c(0, 1)
bpmf_mass <- dbinom(bpmf_outcomes, size = 1, prob = 0.8)
bpmf_y <- factor(bpmf_outcomes)
bpmf_fig <- tibble::tibble(y = bpmf_y, mass = bpmf_mass)
bpmf_fills <- c("0" = "#0072B2", "1" = "#D55E00")
bpmf_limits <- c(0, 1)
bpmf_fig |>
  ggplot2::ggplot(ggplot2::aes(y, mass, fill = y)) +
  ggplot2::geom_col(width = 0.5) +
  ggplot2::geom_text(ggplot2::aes(label = mass), vjust = -0.4) +
  ggplot2::scale_fill_manual(values = bpmf_fills, guide = "none") +
  ggplot2::scale_y_continuous(limits = bpmf_limits) +
  ggplot2::labs(x = "Outcome", y = "Probability mass")
```

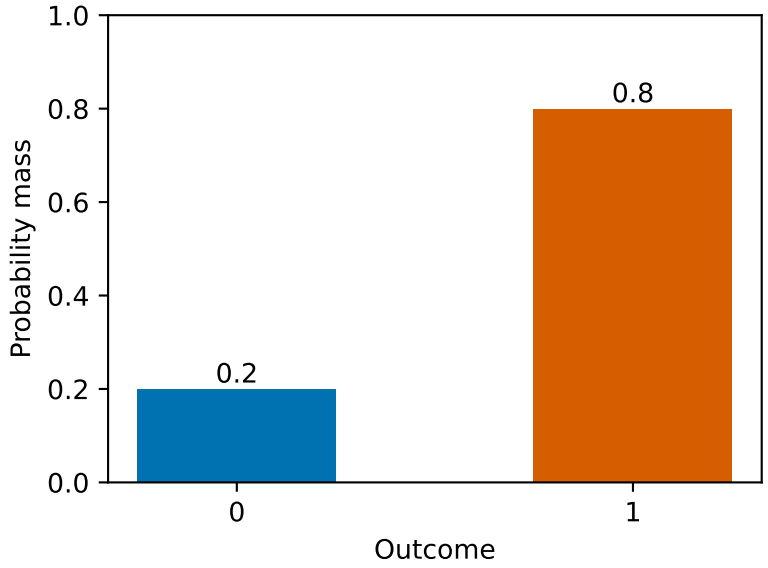


(a)

1.0.4 Python

```
import matplotlib.pyplot as plt
from scipy.stats import bernoulli

fig, ax = plt.subplots(figsize=(4, 3), layout="constrained")
masses_bpmf = bernoulli.pmf([0, 1], 0.8)
bars_bpmf = ax.bar(["0", "1"], masses_bpmf, color=["#0072B2", "#D55E00"], width=0.5);
ax.bar_label(bars_bpmf, fmt="%.1f");
ax.set_ylim(0, 1);
ax.set_xlabel("Outcome");
ax.set_ylabel("Probability mass");
plt.show()
```



(b)

Figure 1: The probability masses of the two outcomes for $\pi_i = 0.8$. The outcome 1 has probability 0.8 and the outcome 0 has probability 0.2.

Exercise 1.2 (Likelihood of the dataset)**.** Using the single-observation probability from Exercise 1.1, express the likelihood[a] $L(\vec{\beta})$ of the full dataset in terms of the $\pi_i = \operatorname{expit}\left(\vec{\beta} \cdot \underset{\sim}{x}_i\right)$ and labels y_i, and state the assumption required.

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik

Solution

Solution 1.2.

$$L(\vec{\beta}) = \prod_{i=1}^{n} \pi_i^{y_i}(1-\pi_i)^{1-y_i}, \qquad \pi_i = \operatorname{expit}\left(\vec{\beta} \cdot \underset{\sim}{x}_i\right)$$

The product form requires that the observations be conditionally independent[a] given their features. Conditional independence[b] is a different assumption from independence[c] of the Y_i themselves, and neither assumption implies the other. Two patients with similar features have similar π_i, so their outcomes can be correlated when the features are not held fixed, and the model allows that correlation[d]. Conditional independence rules out datasets in which y_1 carries information about y_2 beyond what $\underset{\sim}{x}_1$ and $\underset{\sim}{x}_2$ provide. Two patients from the same household can share exposures that the features do not record, and repeated measurements on the same subject, or students within one classroom, are

further examples. Such data violate conditional independence and need a method that accounts for the grouping, such as a model with a shared random effect, or standard errors[e] that allow for correlation within groups.

[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
[d] https://morrison-lab.github.io/pds/variance-covariance.html#def-correlation
[e] https://morrison-lab.github.io/sds/estimation.html#def-SE

Lemma 1.2 (Likelihood of conditionally independent Bernoulli outcomes)**.** *If $Y_1, \dots, Y_n$ are conditionally independent[a] given $\underset{\sim}{x}_1, \dots, \underset{\sim}{x}_n$ with $Y_i \mid \underset{\sim}{x}_i \sim \text{Bernoulli}(\pi_i)$, 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

Proof

Proof. By Solution 1.2. □

Example 1.2 (The likelihood of three outcomes)**.** Consider the outcomes $(1, 0, 1)$ with $(\pi_1, \pi_2, \pi_3) = (0.9, 0.2, 0.7)$. Lemma 1.2 gives the product of the three probability masses:

1.0.5 R

```
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
```

1.0.6 Python

```
import numpy as np

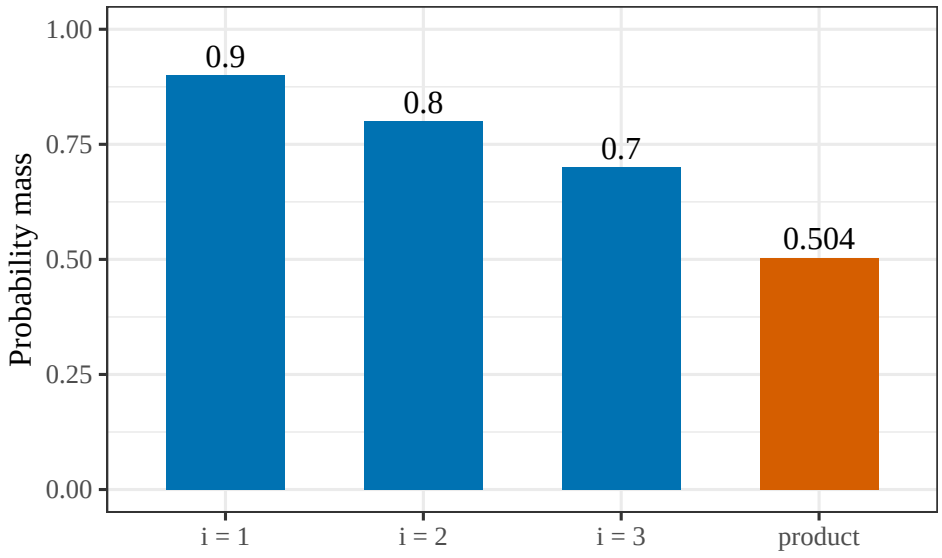
y_bl = np.array([1, 0, 1])
p_bl = np.array([0.9, 0.2, 0.7])
print(np.prod(p_bl**y_bl * (1 - p_bl)**(1 - y_bl)))
```

```
0.504
```

Figure 2 plots each factor and their product.

1.0.7 R

```r
bl_names <- c("i = 1", "i = 2", "i = 3", "product")
bl_y <- c(1, 0, 1)
bl_p <- c(0.9, 0.2, 0.7)
bl_mass <- bl_p^bl_y * (1 - bl_p)^(1 - bl_y)
bl_product <- prod(bl_mass)
bl_values <- c(bl_mass, bl_product)
bl_shown <- round(bl_values, 3)
bl_what <- factor(bl_names, levels = bl_names)
bl_kinds <- c("observation", "observation", "observation", "product")
bl_fig <- tibble::tibble(what = bl_what, value = bl_values, kind = bl_kinds)
bl_fills <- c(observation = "#0072B2", product = "#D55E00")
bl_limits <- c(0, 1)
bl_fig |>
  ggplot2::ggplot(ggplot2::aes(what, value, fill = kind)) +
  ggplot2::geom_col(width = 0.6) +
  ggplot2::geom_text(ggplot2::aes(label = bl_shown), vjust = -0.4) +
  ggplot2::scale_fill_manual(values = bl_fills, guide = "none") +
  ggplot2::scale_y_continuous(limits = bl_limits) +
  ggplot2::labs(x = NULL, y = "Probability mass")
```

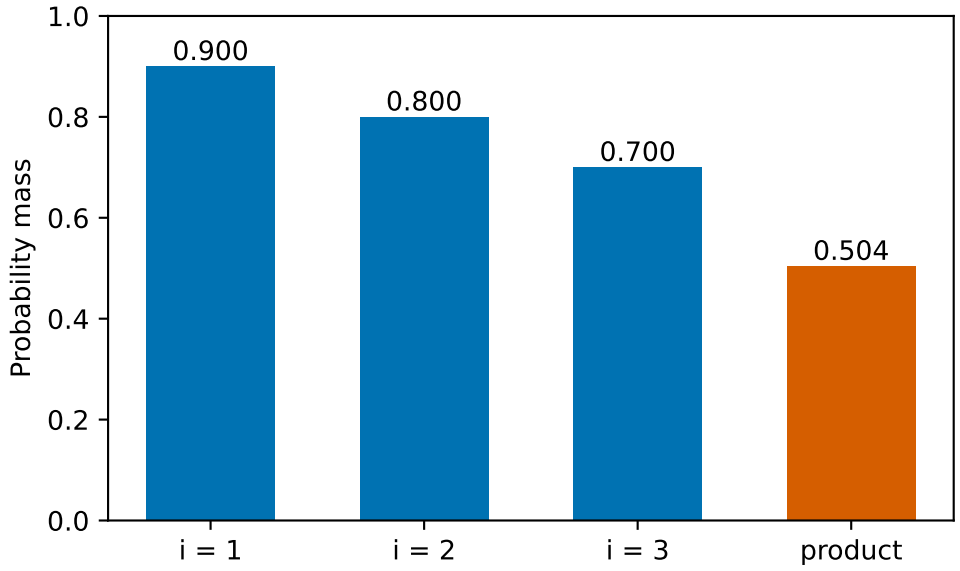


(a)

1.0.8 Python

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

y_blf = np.array([1, 0, 1])
p_blf = np.array([0.9, 0.2, 0.7])
mass_blf = p_blf**y_blf * (1 - p_blf)**(1 - y_blf)
fig, ax = plt.subplots(figsize=(5, 3), layout="constrained")
bars_blf = ax.bar(["i = 1", "i = 2", "i = 3", "product"], [*mass_blf, np.prod(mass_blf)],
                  color=["#0072B2"] * 3 + ["#D55E00"], width=0.6);
ax.bar_label(bars_blf, fmt="%.3f");
ax.set_ylim(0, 1);
ax.set_ylabel("Probability mass");
plt.show()
```



(b)

Figure 2: The probability mass of each observed outcome (blue) and the likelihood[a], their product (vermilion).

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik

Exercise 1.3 (Log-likelihood of the dataset)**.** Take the logarithm of the likelihood[a] $L(\vec{\beta})$ from Exercise 1.2 and simplify.

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik

Solution

Solution 1.3.

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

The second line uses the likelihood[a] from Solution 1.2.
Maximizing $\ell(\vec{\beta})$ is equivalent to minimizing $-\ell(\vec{\beta})$, which is the total log loss. Minimizing it adds no assumption to the maximum likelihood[b] fit for conditionally independent[c] Bernoulli[d] outcomes.

[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/pds/independence.html#def-cind
[d] https://morrison-lab.github.io/pds/random-variables.html#def-bernoulli

Exercise 1.4 (Log-likelihood of the dataset (part 1b))**.** Use the setup of Exercise 1.3.
Give two reasons to work with the log-likelihood[a] $\log L(\vec{\beta})$ rather than the likelihood[b] $L(\vec{\beta})$ of Exercise 1.2.

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik
[b] https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik

Solution

Solution 1.4. This continues Solution 1.3.
There are two reasons to work with $\ell(\vec{\beta})$.

1. *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 size.
2. *Optimization.* log is strictly increasing, so it does not change the maximizer, and a sum is differentiated term by term: the gradient of $\ell(\vec{\beta})$ is the sum of n per-observation gradients. Stochastic gradient descent uses a few of its terms at each step.

Lemma 1.3 (Log-likelihood of conditionally independent Bernoulli outcomes)**.** *Under the assumptions of Lemma 1.2, 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

Proof

Proof. By Solution 1.3. □

Example 1.3 (The log-likelihood of three outcomes)**.** With the outcomes and probabilities of Example 1.2, Lemma 1.3 gives the log-likelihood[a] as a sum, which equals the log of the likelihood[b]:

1.0.9 R

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

```
[1] -0.685179
```

```
log(prod(p_bll^y_bll * (1 - p_bll)^(1 - y_bll)))
```

```
[1] -0.685179
```

1.0.10 Python

```
import numpy as np

y_bll = np.array([1, 0, 1])
p_bll = np.array([0.9, 0.2, 0.7])
print(np.sum(y_bll * np.log(p_bll) + (1 - y_bll) * np.log(1 - p_bll)))
```

```
-0.6851790109107685
```

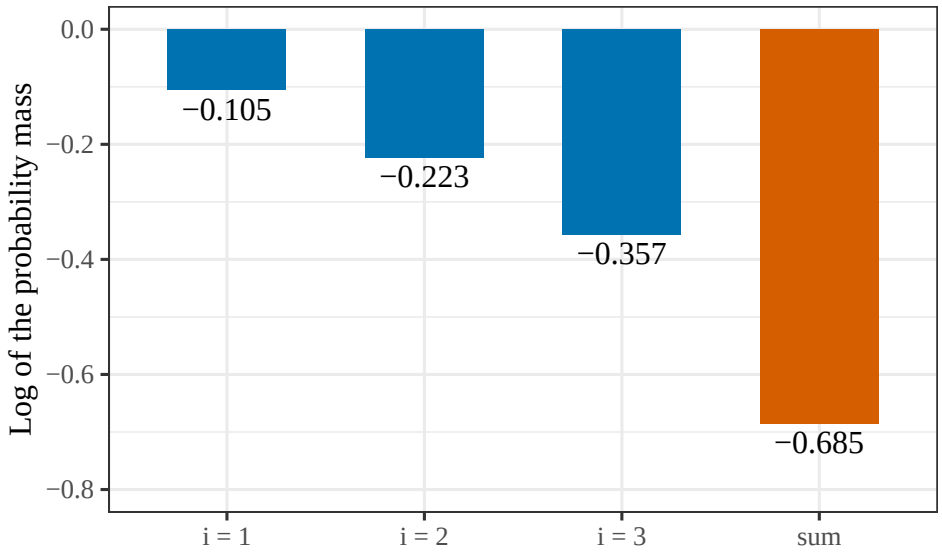
```
print(np.log(np.prod(p_bll**y_bll * (1 - p_bll)**(1 - y_bll))))
```

```
-0.6851790109107684
```

Figure 3 plots each term and their sum.

1.0.11 R

```r
bll_names <- c("i = 1", "i = 2", "i = 3", "sum")
bll_y <- c(1, 0, 1)
bll_p <- c(0.9, 0.2, 0.7)
bll_logmass <- bll_y * log(bll_p) + (1 - bll_y) * log(1 - bll_p)
bll_sum <- sum(bll_logmass)
bll_values <- c(bll_logmass, bll_sum)
bll_shown <- round(bll_values, 3)
bll_what <- factor(bll_names, levels = bll_names)
bll_kinds <- c("observation", "observation", "observation", "sum")
bll_fig <- tibble::tibble(what = bll_what, value = bll_values, kind = bll_kinds)
bll_fills <- c(observation = "#0072B2", sum = "#D55E00")
bll_limits <- c(-0.8, 0)
bll_fig |>
  ggplot2::ggplot(ggplot2::aes(what, value, fill = kind)) +
  ggplot2::geom_col(width = 0.6) +
  ggplot2::geom_text(ggplot2::aes(label = bll_shown), vjust = 1.4) +
  ggplot2::scale_fill_manual(values = bll_fills, guide = "none") +
  ggplot2::scale_y_continuous(limits = bll_limits) +
  ggplot2::labs(x = NULL, y = "Log of the probability mass")
```

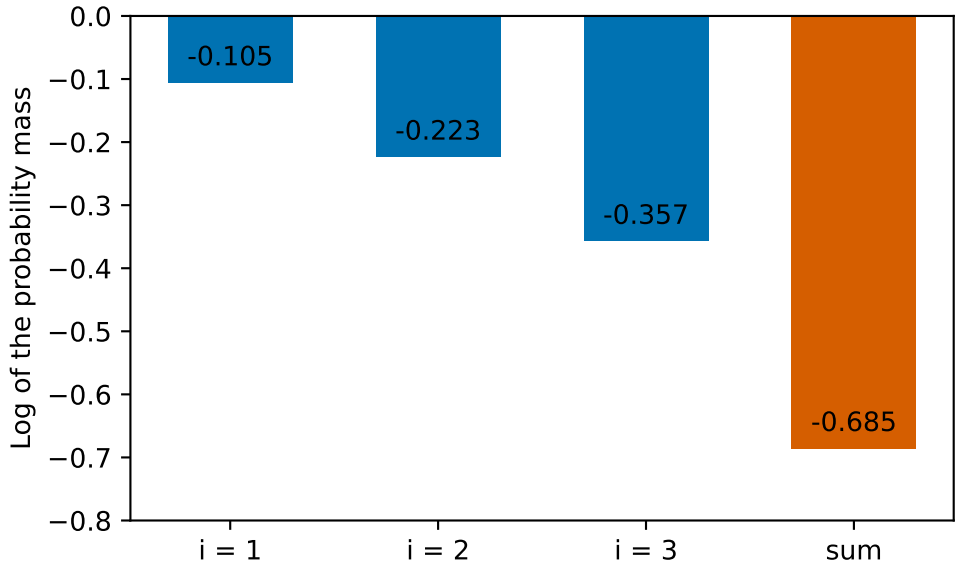


(a)

1.0.12 Python

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

y_bllf = np.array([1, 0, 1])
p_bllf = np.array([0.9, 0.2, 0.7])
logmass_bllf = y_bllf * np.log(p_bllf) + (1 - y_bllf) * np.log(1 - p_bllf)
fig, ax = plt.subplots(figsize=(5, 3), layout="constrained")
bars_bllf = ax.bar(["i = 1", "i = 2", "i = 3", "sum"], [*logmass_bllf, logmass_bllf.sum()],
                   color=["#0072B2"] * 3 + ["#D55E00"], width=0.6);
ax.bar_label(bars_bllf, fmt="%.3f", label_type="edge", padding=-14);
ax.set_ylim(-0.8, 0);
ax.set_ylabel("Log of the probability mass");
plt.show()
```



(b)

Figure 3: The log of each observation's probability mass (blue) and the log-likelihood[c], their sum (vermilion).

[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/sds/intro-MLEs.html#def-loglik

Exercise 1.5 (Negative log-likelihood on a positive observation)**.** Consider one observation with label $y_i = 1$ in the log-likelihood[a] of Exercise 1.3. This exercise works with $-\log L(\vec{\beta})$, the negative of the log-likelihood, because it treats that negative as a loss to compare with squared-error loss, and a loss is minimized rather than maximized. Give its contribution to $-\log L(\vec{\beta})$ at $\pi_i = 0.9$, 0.5, 0.1, and 0.01, to three significant figures.

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik

Solution

Solution 1.5. With $y_i = 1$, observation i's contribution to $-\log L(\vec{\beta})$ is $-\log \pi_i$:

```r
llp_prob <- c(0.9, 0.5, 0.1, 0.01)
knitr::kable(
  data.frame(
    prob = llp_prob,
    loss = formatC(-log(llp_prob), digits = 3, format = "fg", flag = "#")
  ),
  col.names = c("$\\prob_i$", "$-\\log \\prob_i$")
)
```

π_i	$-\log \pi_i$
0.90	0.105
0.50	0.693
0.10	2.30
0.01	4.61

Exercise 1.6 (Negative log-likelihood on a positive observation (part 1b))**.** Use the setup of Exercise 1.5.
State what happens to the contribution of an observation with $y_i = 1$ to $-\log L$ as $\pi_i \to 0$.

Solution

Solution 1.6. This continues Solution 1.5.
For an observation with $y_i = 1$, the loss $-\log \pi_i$ of Solution 1.5 increases without bound as $\pi_i \to 0$. The model never gives $\pi_i = 0$ exactly, because $\pi_i = \operatorname{expit}\left(\vec{\beta} \cdot \underset{\sim}{x}_i\right)$ is in $(0, 1)$, but no finite upper bound holds for the loss of an observation with $y_i = 1$.

Exercise 1.7 (Graphing one observation's log-likelihood contribution)**.** In the log-likelihood[a] of Lemma 1.3, observation[b] i with label y_i contributes $y_i \log \pi_i + (1 - y_i) \log(1 - \pi_i)$. Graph[c] this contribution against π_i over the interval[d] $(0, 1)$, as one curve for $y_i = 1$ and one for $y_i = 0$.

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik
[b] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-observation
[c] https://morrison-lab.github.io/mcs/functions.html#def-graph
[d] https://morrison-lab.github.io/mcs/sets.html#def-interval

Solution

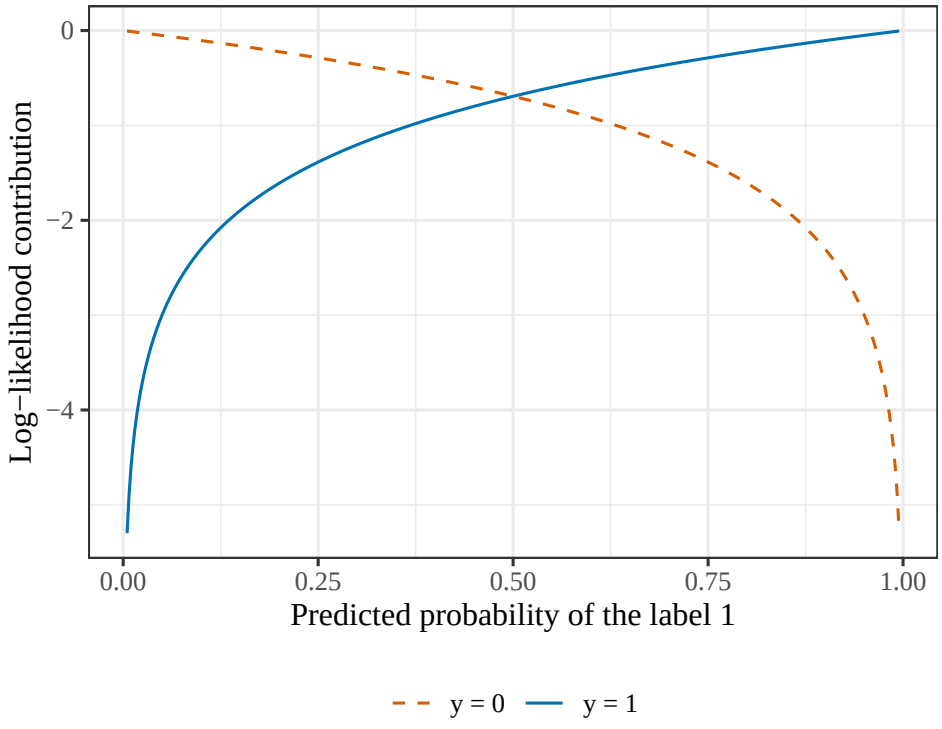
Solution 1.7. For label $y_i = 1$ the contribution is $\log \pi_i$, and for $y_i = 0$ it is $\log(1 - \pi_i)$. Figure 4 shows both curves.

- $y_i = 1$**:** the curve increases in π_i, approaches 0 as $\pi_i \to 1$, and tends to $-\infty$ as $\pi_i \to 0$.
- $y_i = 0$**:** the curve decreases in π_i, approaches 0 as $\pi_i \to 0$, and tends to $-\infty$ as $\pi_i \to 1$.

Each curve is the other reflected about $\pi_i = 1/2$. Both curves are negative on all of $(0, 1)$, and each approaches 0 only as the probability[a] that the model gives the observed label approaches 1.

1.0.13 R

```r
lcc_prob <- seq(0.005, 0.995, length.out = 400)
lcc_loglik_pos <- log(lcc_prob)
lcc_prob_neg <- 1 - lcc_prob
lcc_loglik_neg <- log(lcc_prob_neg)
lcc_pos <- tibble::tibble(
  prob = lcc_prob,
  contribution = lcc_loglik_pos,
  label = "y = 1"
)
lcc_neg <- tibble::tibble(
  prob = lcc_prob,
  contribution = lcc_loglik_neg,
  label = "y = 0"
)
lcc_curves <- dplyr::bind_rows(lcc_pos, lcc_neg)
lcc_colours <- c("y = 1" = "#0072B2", "y = 0" = "#D55E00")
lcc_linetypes <- c("y = 1" = "solid", "y = 0" = "dashed")
lcc_curves |>
  ggplot2::ggplot() +
  ggplot2::aes(
    x = prob,
    y = contribution,
    colour = label,
    linetype = label
  ) +
  ggplot2::geom_line() +
  ggplot2::scale_colour_manual(values = lcc_colours) +
  ggplot2::scale_linetype_manual(values = lcc_linetypes) +
  ggplot2::labs(
    x = "Predicted probability of the label 1",
    y = "Log-likelihood contribution",
    colour = NULL,
    linetype = NULL
  )
```

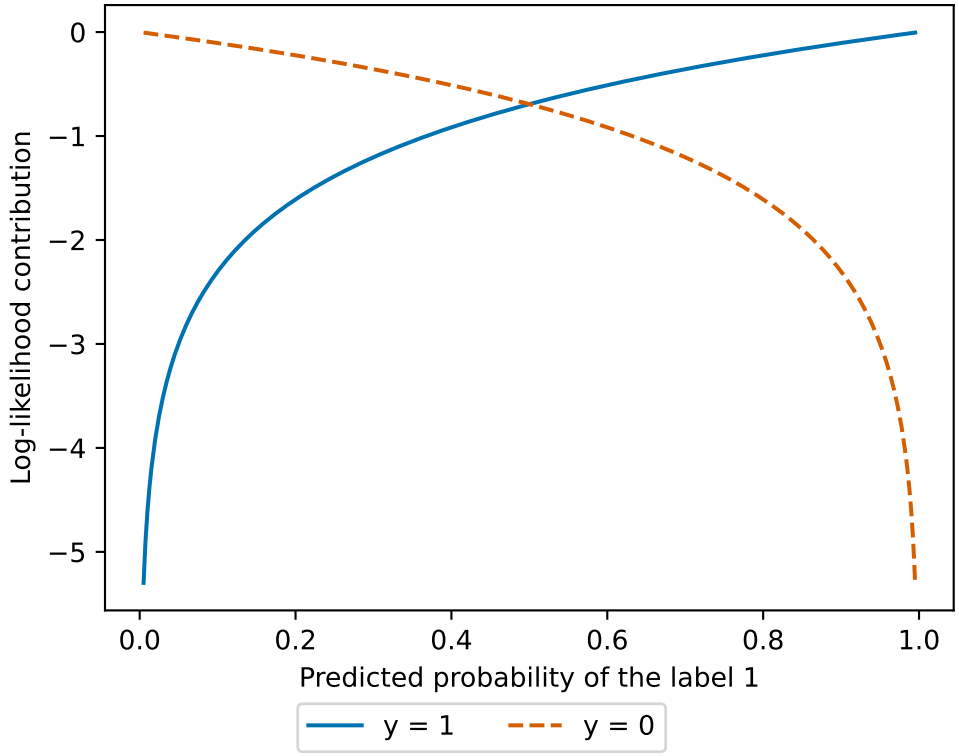

0
−2
−4
Log−likelihood contribution
0.00
0.25
0.50
0.75
1.00
Predicted probability of the label 1
y = 0
y = 1

(a)

1.0.14 Python

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

lcc_prob = np.linspace(0.005, 0.995, 400)
fig, ax = plt.subplots(figsize=(5, 4), layout="constrained")
ax.plot(lcc_prob, np.log(lcc_prob), color="#0072B2", linestyle="-", label="y = 1");
ax.plot(lcc_prob, np.log(1 - lcc_prob), color="#D55E00", linestyle="--", label="y = 0");
ax.set_xlabel("Predicted probability of the label 1");
ax.set_ylabel("Log-likelihood contribution");
fig.legend(loc="outside lower center", ncol=2);
plt.show()
```



(b)

Figure 4: One observation[b]'s contribution to the log-likelihood[c] of Lemma 1.3, for each label, as the predicted probability[d] π_i moves across $(0, 1)$. Each curve approaches 0 as the probability[e] of the observed label approaches 1, and tends to $-\infty$ as that probability approaches 0.

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-probability
[b] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-observation
[c] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik
[d] https://morrison-lab.github.io/sds/classification.html#def-predicted-probability
[e] https://morrison-lab.github.io/pds/probability-basics.html#def-probability

Exercise 1.8 (Why squared error on the probabilities is unsuitable)**.** Under squared-error loss, the positive observation of Exercise 1.5 contributes $(\pi_i - y_i)^2$, which never exceeds 1, however small π_i is. Explain what that upper bound allows a fitted model to do without incurring a large loss.

Solution

Solution 1.8. The squared-error loss on a single observation is at most 1, so the loss on one misclassified observation stays below 1 even as $\pi_i \to 0$. Suppose that letting $\pi_i \to 0$ on one observation with $y_i = 1$ reduces the sum of squared errors on the other observations by more than 1. Then the total squared error decreases, so a fit can give an observed outcome a probability near 0 at a cost of less than 1. Under the log loss, $-\log \pi_i \to \infty$ as $\pi_i \to 0$ (Solution 1.6), so no finite reduction in loss on the other observations can offset $\pi_i \to 0$ on an observation with $y_i = 1$.

The case against squared error is not that it rewards reporting a wrong probability. Squared error on a probability is the *Brier score*, and its expected value[a] is minimized by reporting the true probability, as the expected value[b] of the log loss is. Squared error has three other drawbacks:

- it is bounded above;
- it has no maximum-likelihood[c] interpretation, which is what supplies standard errors[d] and likelihood ratio tests[e] for the fitted coefficients;
- its derivative[f] with respect to the log-odds[g] $\vec{\beta} \cdot \underset{\sim}{x}_i$ has the logistic function[h]'s derivative as a factor, which is close to 0 when the log-odds is far from 0, so the gradient is close to 0 on badly misclassified observations.

Squared error on π_i is neither a plausible loss for Bernoulli[i] outcomes nor a friendly loss when paired with the logistic function[j], and the log loss is both.

[a] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[b] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[c] https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle
[d] https://morrison-lab.github.io/sds/estimation.html#def-SE
[e] https://morrison-lab.github.io/sds/intro-MLEs.html#def-lrt-stat
[f] https://morrison-lab.github.io/mds/calculus-derivatives.html#def-derivative
[g] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds
[h] https://morrison-lab.github.io/mcs/functions.html#def-logistic-function
[i] https://morrison-lab.github.io/pds/random-variables.html#def-bernoulli
[j] https://morrison-lab.github.io/mcs/functions.html#def-logistic-function

Exercise 1.9 (Why squared error on the probabilities is unsuitable (part 1b))**.** Use the setup of Exercise 1.8.

State which of the two requirements in the classifier requirements exercise the log loss satisfies.

Solution

Solution 1.9. This continues Solution 1.8.

The log loss satisfies the second requirement of the classifier requirements exercise, saturation on the side of the true label. On that side, $-\log \pi_i$ decreases to 0 as $\pi_i \to 1$, so raising π_i further on an observation that already has π_i near 1 reduces the loss by at most $-\log \pi_i$, which is near 0. On the wrong side, the loss increases without bound.

Definition 1.1 (Log loss)**.** For a label y_i and a model that predicts the probability of each value of the label given $\underset{\sim}{x}_i$, the per-observation **log loss** is the negative logarithm of the probability that the model gives to the observed label,

$$C_i = -\log \Pr(Y_i = y_i \mid \underset{\sim}{x}_i).$$

Here C_i is the name we give this loss for observation i. For a binary label, we write $\pi_i = \Pr(Y_i = 1 \mid \underset{\sim}{x}_i)$ for the predicted probability of label 1, and $L(y_i, \pi_i) = C_i$.

Example 1.4 (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 $\pi_i = 0.8, 0.6, 0.2, 0.01$:

1.0.15 R

```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
```

1.0.16 Python

```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))
```

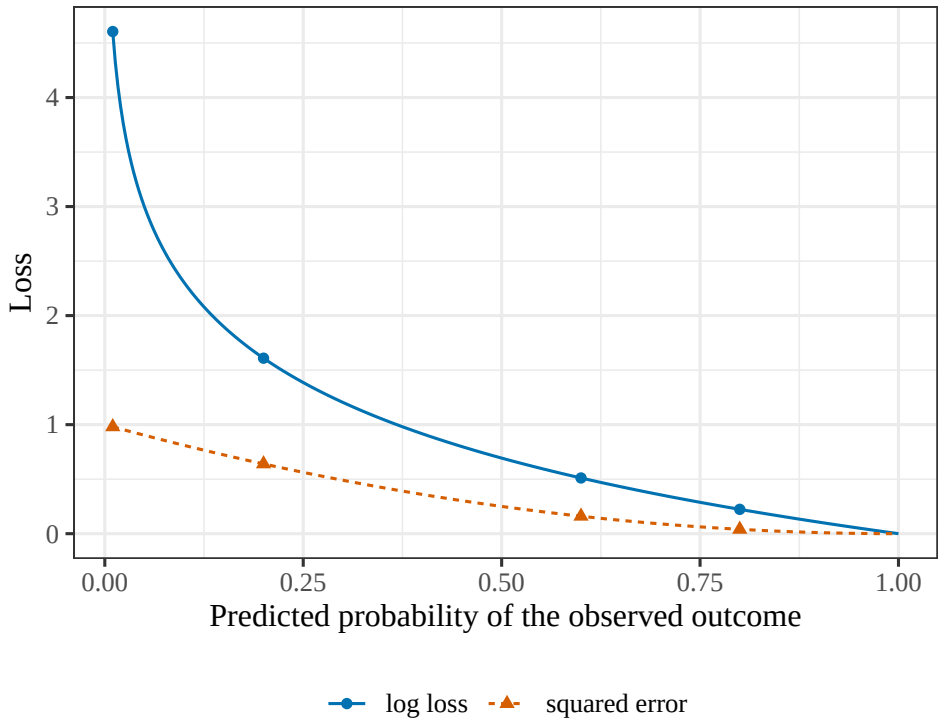
```
   prob  log_loss  squared_error
0  0.80     0.223           0.04
1  0.60     0.511           0.16
2  0.20     1.609           0.64
3  0.01     4.605           0.98
```

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 $\pi_i \to 0$.

Figure 5 plots the log loss and the squared error against the probability.

1.0.17 R

```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
```

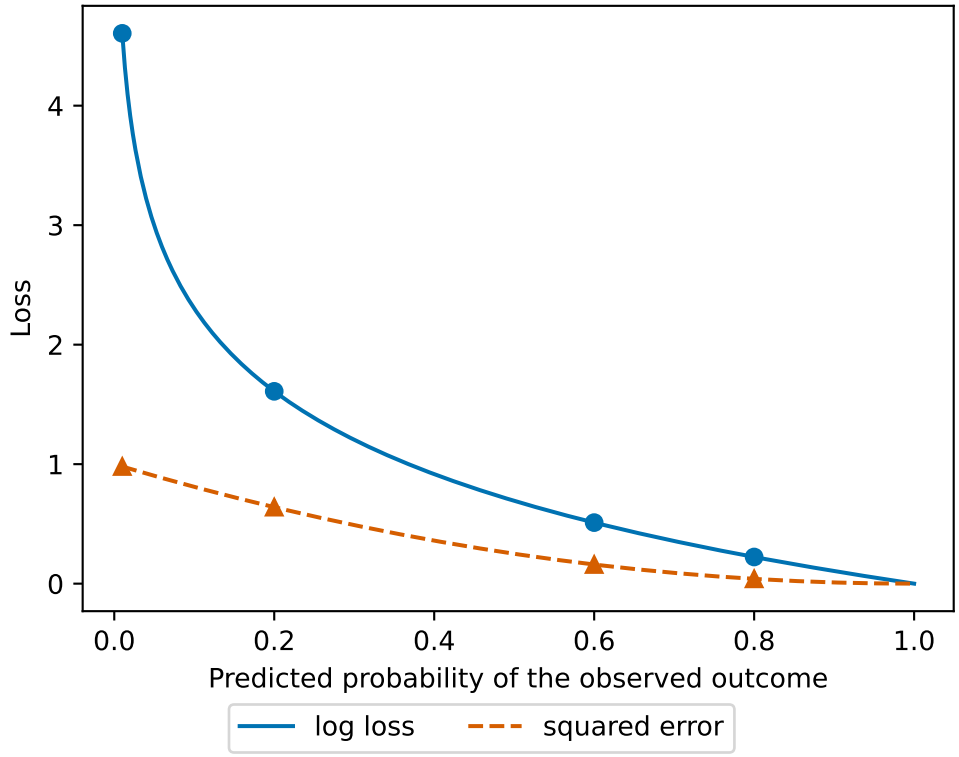


(a)

1.0.18 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()
```



(b)

Figure 5: The log loss and the squared error of Example 1.4 for a positive case, as the probability given to the positive class moves from 0 to 1, with the four values of the example marked. The squared error stays below 1, while the log loss grows without bound as the probability approaches 0.

Theorem 1.1 (Binary form of the log loss)**.** *For a label* $y_i \in \{0, 1\}$ *and a predicted probability* $\pi_i \in (0, 1)$*, the log loss is*

$$C_i = -y_i \log \pi_i - (1 - y_i) \log(1 - \pi_i).$$

Proof

Proof. The model gives probability π_i to label 1 and $1-\pi_i$ to label 0. Because $y_i \in \{0, 1\}$, the observed label is one of the two:

- **Case** $y_i = 1$: $C_i = -\log \pi_i$. This is the formula, because $1 - y_i = 0$.
- **Case** $y_i = 0$: $C_i = -\log(1 - \pi_i)$. This is the formula, because $y_i = 0$.

□

Example 1.5 (Binary form of the log loss for a label of zero)**.** Suppose $y_i = 0$, and the model predicts $\pi_i = 0.8$. The model gives probability $1-0.8 = 0.2$ to the observed label, so the log loss is $C_i = -\log 0.2$. The formula of Theorem 1.1 gives the same value,

$$C_i = -0 \cdot \log 0.8 - (1 - 0) \log(1 - 0.8) = -\log 0.2.$$

Definition 1.2 (Cross-entropy loss)**.** For a label $y_i \in \{1, \dots, K\}$ and a model that predicts the probability $q_{ik} = \Pr(Y_i = k \mid \underset{\sim}{x}_i)$ of each value k, let $\underset{\sim}{q}_i = (q_{i1}, \dots, q_{iK})$ be the model's distribution[a], and let $\underset{\sim}{p}_i$ be the distribution that puts probability 1 on the observed label, so $p_{ik} = 1$ when $k = y_i$, and $p_{ik} = 0$ otherwise. The per-observation **cross-entropy loss** is the cross-entropy[b] of $\underset{\sim}{q}_i$ relative to $\underset{\sim}{p}_i$,

$$H(\underset{\sim}{p}_i, \underset{\sim}{q}_i).$$

For a binary label coded 0 and 1, the two entries of each vector are listed in that order.

[a] https://morrison-lab.github.io/pds/random-variables.html#def-cdf
[b] https://morrison-lab.github.io/pds/expectation.html#def-cross-entropy

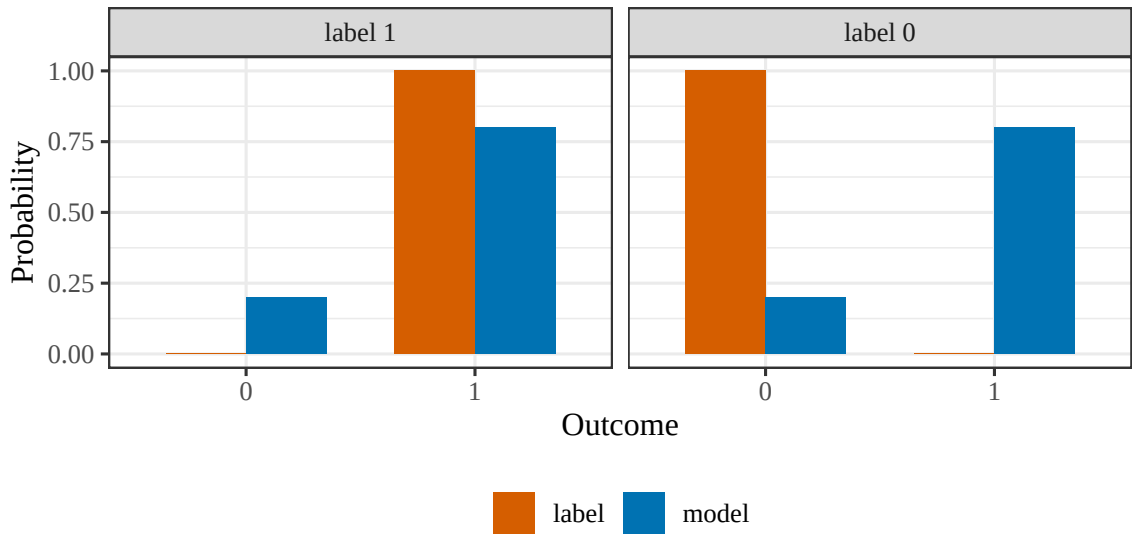
Example 1.6 (Label and model distributions for one observation)**.** Suppose $y_i = 1$, and the model predicts $\pi_i = 0.8$. The distribution[a] of the observed label is $\underset{\sim}{p}_i = (0, 1)$, and the model's distribution is $\underset{\sim}{q}_i = (0.2, 0.8)$. The cross-entropy loss is

$$H(\underset{\sim}{p}_i, \underset{\sim}{q}_i) = -0 \cdot \log 0.2 - 1 \cdot \log 0.8 = -\log 0.8,$$

where the term with probability 0 is taken to be 0.
Figure 6 plots both cases.

1.0.19 R

```r
cel_labels <- c("label 1", "label 0")
cel_label <- rep(cel_labels, each = 4)
cel_zero_one <- c(0, 1)
cel_outcome_raw <- rep(cel_zero_one, times = 4)
cel_outcome <- factor(cel_outcome_raw)
cel_sources <- c("label", "model")
cel_who_once <- rep(cel_sources, each = 2)
cel_who <- rep(cel_who_once, times = 2)
cel_mass <- c(0, 1, 0.2, 0.8, 1, 0, 0.2, 0.8)
cel_panel <- factor(cel_label, levels = cel_labels)
cel_fig <- tibble::tibble(
  label = cel_panel,
  outcome = cel_outcome,
  who = cel_who,
  mass = cel_mass
)
cel_fills <- c(label = "#D55E00", model = "#0072B2")
cel_fig |>
  ggplot2::ggplot(ggplot2::aes(outcome, mass, fill = who)) +
  ggplot2::geom_col(position = "dodge", width = 0.7) +
  ggplot2::facet_wrap(~label) +
  ggplot2::scale_fill_manual(values = cel_fills) +
  ggplot2::labs(x = "Outcome", y = "Probability", fill = NULL)
```

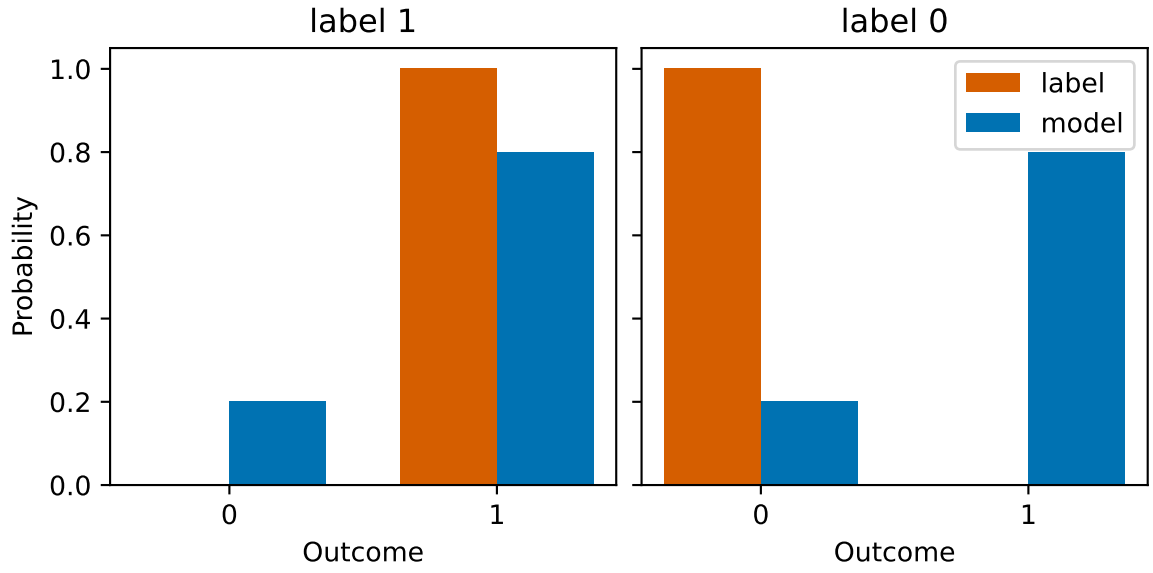


(a)

1.0.20 Python

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

fig, axes = plt.subplots(1, 2, figsize=(6, 3), sharey=True, layout="constrained")
pos_cel = np.arange(2)
for ax, title, p_label in zip(axes, ["label 1", "label 0"], [[0, 1], [1, 0]]):
    ax.bar(pos_cel - 0.18, p_label, width=0.36, color="#D55E00", label="label");
    ax.bar(pos_cel + 0.18, [0.2, 0.8], width=0.36, color="#0072B2", label="model");
    ax.set_xticks(pos_cel, ["0", "1"]);
    ax.set_title(title);
    ax.set_xlabel("Outcome");
axes[0].set_ylabel("Probability");
axes[1].legend();
plt.show()
```



(b)

Figure 6: The distribution[b] of the label (vermilion) and of the model (blue) over the two outcomes, for a label of 1 and a label of 0, with the model's probability 0.8.

[a] https://morrison-lab.github.io/pds/random-variables.html#def-cdf
[b] https://morrison-lab.github.io/pds/random-variables.html#def-cdf

Theorem 1.2 (The cross-entropy loss equals the log loss)**.** *For a label $y_i \in \{1, \dots, K\}$, the cross-entropy loss equals the log loss,*

$$H(\underset{\sim}{p}_i, \underset{\sim}{q}_i) = C_i.$$

Proof

Proof. Let $\underset{\sim}{p}_i$ and $\underset{\sim}{q}_i$ be as in the definition. By the definition of cross-entropy[a],

$$
\begin{aligned}
H(\underset{\sim}{p}_i, \underset{\sim}{q}_i) &= -\sum_{k=1}^{K} p_{ik} \log q_{ik} && \text{(definition of cross-entropy)} \\
&= -\log q_{iy_i} && (p_{ik} = 0 \text{ unless } k = y_i, \text{ and } p_{iy_i} = 1) \\
&= -\log \Pr(Y_i = y_i \mid \underset{\sim}{x}_i) && \text{(definition of } q_{ik}) \\
&= C_i.
\end{aligned}
$$

The last line is the definition of the log loss. □

[a] https://morrison-lab.github.io/pds/expectation.html#def-cross-entropy

Example 1.7 (The cross-entropy loss equals the log loss for a label of zero)**.** Suppose $y_i = 0$, and the model predicts $\pi_i = 0.8$. Then $\underset{\sim}{p}_i = (1, 0)$ and $\underset{\sim}{q}_i = (0.2, 0.8)$, so the cross-entropy loss is

$$H(\underset{\sim}{p}_i, \underset{\sim}{q}_i) = -1 \cdot \log 0.2 - 0 \cdot \log 0.8 = -\log 0.2.$$

This equals the log loss $C_i = -\log 0.2$ of Example 1.5.
The right panel of Figure 6 plots this case.

Definition 1.3 (Total log loss)**.** For labels $y_1, \ldots, y_n$ and predicted probabilities $\pi_1, \ldots, \pi_n$ determined by the coefficients $\vec{\beta}$, the **total log loss** is the sum of the per-observation log losses,

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

Example 1.8 (Total log loss of the type A men under one shared probability)**.** Consider the WCGS data of the exercise on why not a line (`data/wcgs.csv`[a]), whose column `dibpat` records each man's behavior pattern, type A or type B, and subset to the men of type A (`dibpat` is `"A"`). Model coronary heart disease (`chd`) on age at enrollment with $\pi_i = \text{expit}\big(\beta_0 + \beta_1 \text{ age}_i\big)$, and take the coefficients $\beta_1 = 0$ and β_0 chosen so that every man gets the same probability $\bar{y}$, the proportion of these men with `chd` equal to 1. Evaluate each man's log loss C_i and the total log loss $C(\vec{\beta})$:

1.0.21 R

```
wcgs_a <- subset(read.csv("data/wcgs.csv"), dibpat == "A")
chd_a <- wcgs_a$chd
p_bar <- mean(chd_a)
loss_i <- -chd_a * log(p_bar) - (1 - chd_a) * log(1 - p_bar)
round(c(
  n = length(loss_i),
  loss_if_chd = -log(p_bar),
  loss_if_no_chd = -log(1 - p_bar),
  total = sum(loss_i)
), 3)
```

```
            n    loss_if_chd loss_if_no_chd          total
     1589.000          2.189          0.119        557.291
```

1.0.22 Python

```
import numpy as np
import pandas as pd

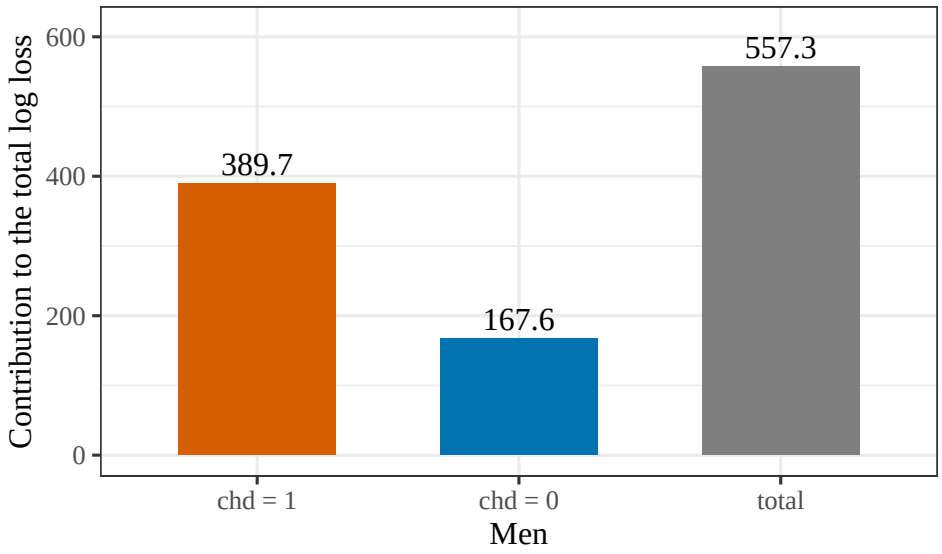
wcgs_all = pd.read_csv("data/wcgs.csv")
wcgs_a = wcgs_all[wcgs_all["dibpat"] == "A"]
chd_a = wcgs_a["chd"].to_numpy(float)
p_bar = chd_a.mean()
loss_i = -chd_a * np.log(p_bar) - (1.0 - chd_a) * np.log(1.0 - p_bar)
print(pd.Series({
    "n": len(loss_i),
    "loss_if_chd": -np.log(p_bar),
    "loss_if_no_chd": -np.log(1.0 - p_bar),
    "total": loss_i.sum(),
}).round(3))
```

```
n                 1589.000
loss_if_chd          2.189
loss_if_no_chd       0.119
total              557.291
dtype: float64
```

Here $\bar{y} = 0.112$. Each of the 178 men with `chd` equal to 1 contributes $-\log \bar{y} = 2.189$, and each of the other 1411 contributes $-\log(1 - \bar{y}) = 0.119$, so the total log loss is 557.3.
Figure 7 plots the two contributions and their total.

1.0.23 R

```r
tll_n1 <- sum(chd_a)
tll_n0 <- sum(1 - chd_a)
tll_contrib <- c(tll_n1 * -log(p_bar), tll_n0 * -log(1 - p_bar))
tll_total <- sum(tll_contrib)
tll_values <- c(tll_contrib, tll_total)
tll_shown <- round(tll_values, 1)
tll_names <- c("chd = 1", "chd = 0", "total")
tll_what <- factor(tll_names, levels = tll_names)
tll_fig <- tibble::tibble(what = tll_what, value = tll_values)
tll_fills <- c("chd = 1" = "#D55E00", "chd = 0" = "#0072B2", total = "grey50")
tll_limits <- c(0, 1.1 * tll_total)
tll_fig |>
  ggplot2::ggplot(ggplot2::aes(what, value, fill = what)) +
  ggplot2::geom_col(width = 0.6) +
  ggplot2::geom_text(ggplot2::aes(label = tll_shown), vjust = -0.4) +
  ggplot2::scale_fill_manual(values = tll_fills, guide = "none") +
  ggplot2::scale_y_continuous(limits = tll_limits) +
  ggplot2::labs(x = "Men", y = "Contribution to the total log loss")
```

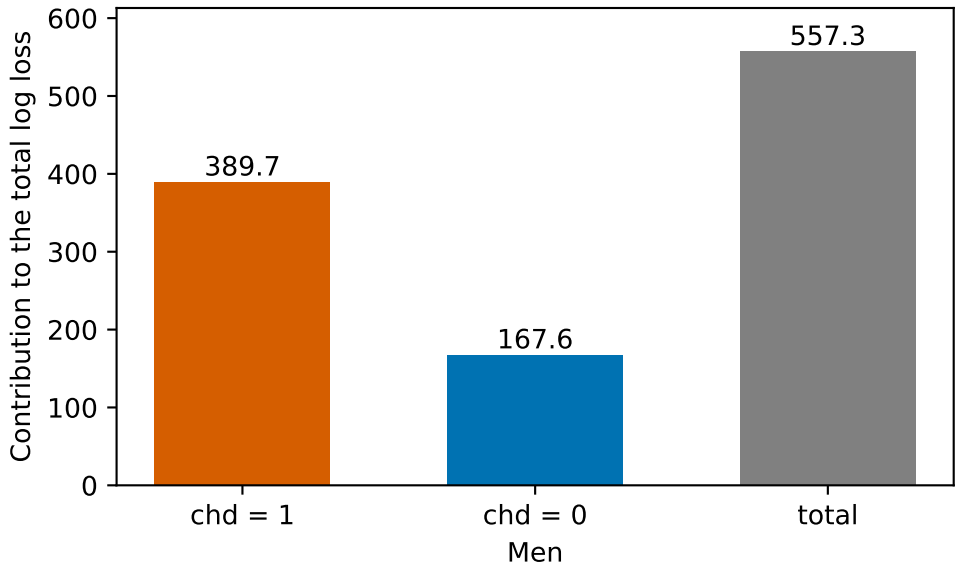


(a)

1.0.24 Python

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

contrib_tll = [chd_a.sum() * -np.log(p_bar), (1 - chd_a).sum() * -np.log(1 - p_bar)]
fig, ax = plt.subplots(figsize=(5, 3), layout="constrained")
bars_tll = ax.bar(["chd = 1", "chd = 0", "total"], [*contrib_tll, sum(contrib_tll)],
                  color=["#D55E00", "#0072B2", "grey"], width=0.6);
ax.bar_label(bars_tll, fmt="%.1f");
ax.set_ylim(0, 1.1 * sum(contrib_tll));
ax.set_xlabel("Men");
ax.set_ylabel("Contribution to the total log loss");
plt.show()
```



(b)

Figure 7: The contribution of the men with `chd` equal to 1 (vermilion) and equal to 0 (blue) to the total log loss of Example 1.8, and the total (gray).

[a] https://morrison-lab.github.io/lds/chapters/data/wcgs.csv

Remark (Per-observation, total and average log loss)

Remark 1.1 (Per-observation, total and average log loss). The average of the total log loss, $\hat{R}\left(\vec{\beta}\right) = \frac{1}{n}C(\vec{\beta})$, is the empirical risk under the log loss. 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 $\hat{R}\left(\vec{\beta}\right)$ [1]. Because the cross-entropy loss equals the log loss (Theorem 1.2), the sum of the cross-entropy losses equals the total log loss.

Proposition 1.1 (The total log loss is the negative log-likelihood)**.** *Under the assumptions of Lemma 1.2, the total log loss is the negative log-likelihood*[a]*:*

$$C(\vec{\beta}) = -\ell(\vec{\beta}).$$

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik

Proof

Proof. By Definition 1.3,

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

and the last sum is $\ell(\vec{\beta})$ by Lemma 1.3. □

Example 1.9 (The log-likelihood a fitting function reports is the negative total log loss)**.**
For the type A men of Example 1.8, fit two logistic regressions of `chd`, one with an intercept[a] only and one on age. For each fit, compute the total log loss of its fitted probabilities, and compare it with the log-likelihood[b] the software reports:

1.0.25 R

```r
fit_const <- glm(chd ~ 1, family = binomial, data = wcgs_a)
fit_age <- glm(chd ~ age, family = binomial, data = wcgs_a)

total_log_loss <- function(fit) {
  p <- fitted(fit)
  y <- fit$y
  sum(-y * log(p) - (1 - y) * log(1 - p))
}

loglik_check <- data.frame(
  total_log_loss = c(total_log_loss(fit_const), total_log_loss(fit_age)),
  neg_loglik = -c(as.numeric(logLik(fit_const)), as.numeric(logLik(fit_age))),
  row.names = c("intercept_only", "age")
)
round(loglik_check, 4)
```

```
# A tibble: 2 x 2
  total_log_loss neg_loglik
           <dbl>      <dbl>
1           557.       557.
2           544.       544.
```

1.0.26 Python

```python
import statsmodels.api as sm

designs = {
    "intercept_only": np.ones((len(chd_a), 1)),
    "age": sm.add_constant(wcgs_a["age"].to_numpy(float)),
}
rows = {}
for name, x in designs.items():
    fit = sm.GLM(chd_a, x, family=sm.families.Binomial()).fit()
    p = fit.predict()
    rows[name] = {
        "total_log_loss": np.sum(-chd_a * np.log(p) - (1.0 - chd_a) * np.log(1.0 - p)),
        "neg_loglik": -fit.llf,
    }
print(pd.DataFrame(rows).T.round(4))
```

```
               total_log_loss  neg_loglik
intercept_only        557.2912    557.2912
age                   543.5083    543.5083
```

In both rows, and in both languages, the two columns agree, as Proposition 1.1 states. The intercept-only fit gives every man the probability $\bar{y}$, so its row repeats the total of Example 1.8. Adding age lowers the total log loss to 543.5.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[b] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik

Theorem 1.3 (Minimizing the log loss is maximum likelihood estimation)**.** *Suppose the outcomes* $Y_1, \ldots, Y_n \in \{0, 1\}$ *are conditionally independent*[a] *given* $\underset{\sim}{x}_1, \ldots, \underset{\sim}{x}_n$*, with*

$$Y_i \mid \underset{\sim}{x}_i \sim \text{Bernoulli}(\pi_i), \qquad \pi_i = \text{expit}\left(\vec{\beta} \cdot \underset{\sim}{x}_i\right).$$

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

[a] https://morrison-lab.github.io/pds/independence.html#def-cind
[b] https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle

Proof

Proof. By assumption, the outcomes are conditionally independent[a] with $Y_i \mid \underset{\sim}{x}_i \sim \text{Bernoulli}(\pi_i)$, so Lemma 1.2 gives the likelihood[b] and Lemma 1.3 gives the log-likelihood[c] $\ell(\vec{\beta})$, which is $-C(\vec{\beta})$ by Proposition 1.1. Because log is strictly increasing, maximizing the likelihood[d] is equivalent to maximizing $\ell(\vec{\beta})$, which is equivalent to minimizing $C(\vec{\beta})$. Because $n > 0$, minimizing $C(\vec{\beta})$ is equivalent to minimizing $\frac{1}{n}C(\vec{\beta})$. □

[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 1.10 (Minimizing the log loss on the WCGS data gives the maximum likelihood fit)**.** Consider again the type A men of the WCGS data, as in Example 1.8, so that a single curve with two coefficients describes them. Model coronary heart disease (`chd`) on age at enrollment (`age`) with $\pi_i = \text{expit}\left(\beta_0 + \beta_1 \text{age}_i\right)$. Minimize the average log loss (empirical risk) $\hat{R}\left(\vec{\beta}\right) = \frac{1}{n}C(\vec{\beta})$ numerically from $\vec{\beta} = \vec{0}$, and compare with the maximum likelihood[a] fit:

1.0.27 R

```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
```

1.0.28 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))
```

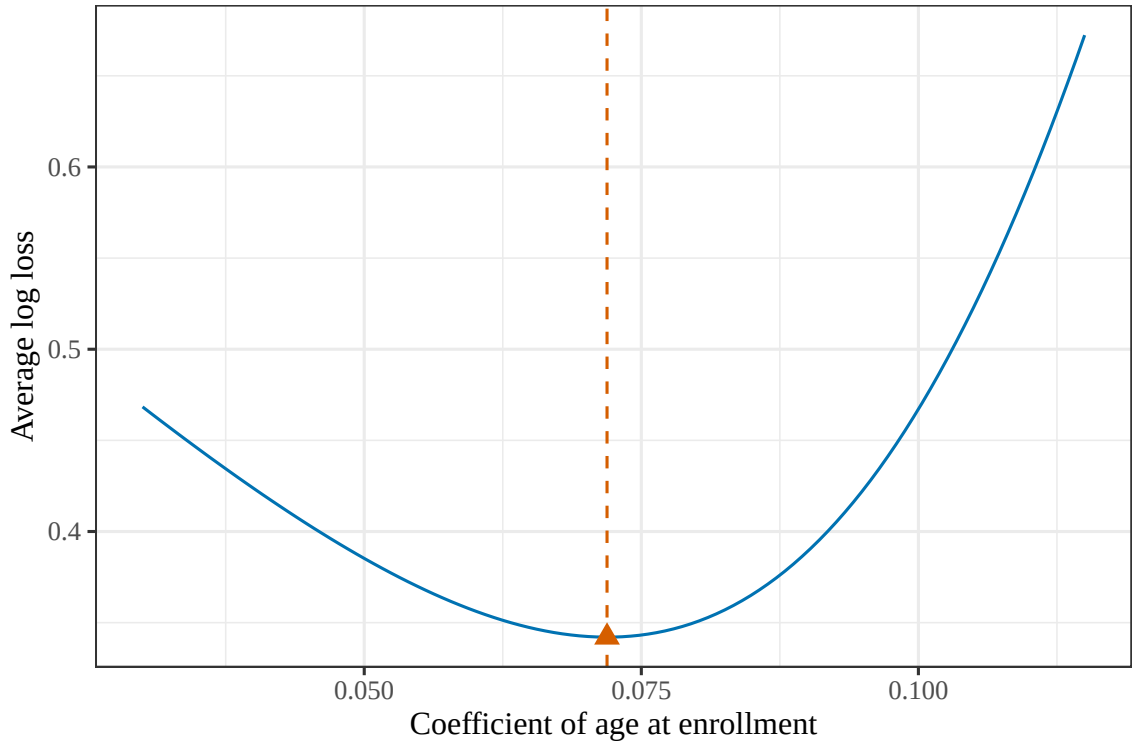
```
                    intercept       age
log_loss_minimizer  -5.498862  0.071907
mle                 -5.498862  0.071907
```

The numerical minimizer of $\hat{R}\left(\vec{\beta}\right)$ and the maximum likelihood[b] estimate[c] differ by at most 3.4e-07, as Theorem 1.3 states.

Figure 8 plots the average log loss against the coefficient of age at enrollment.

1.0.29 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")
```

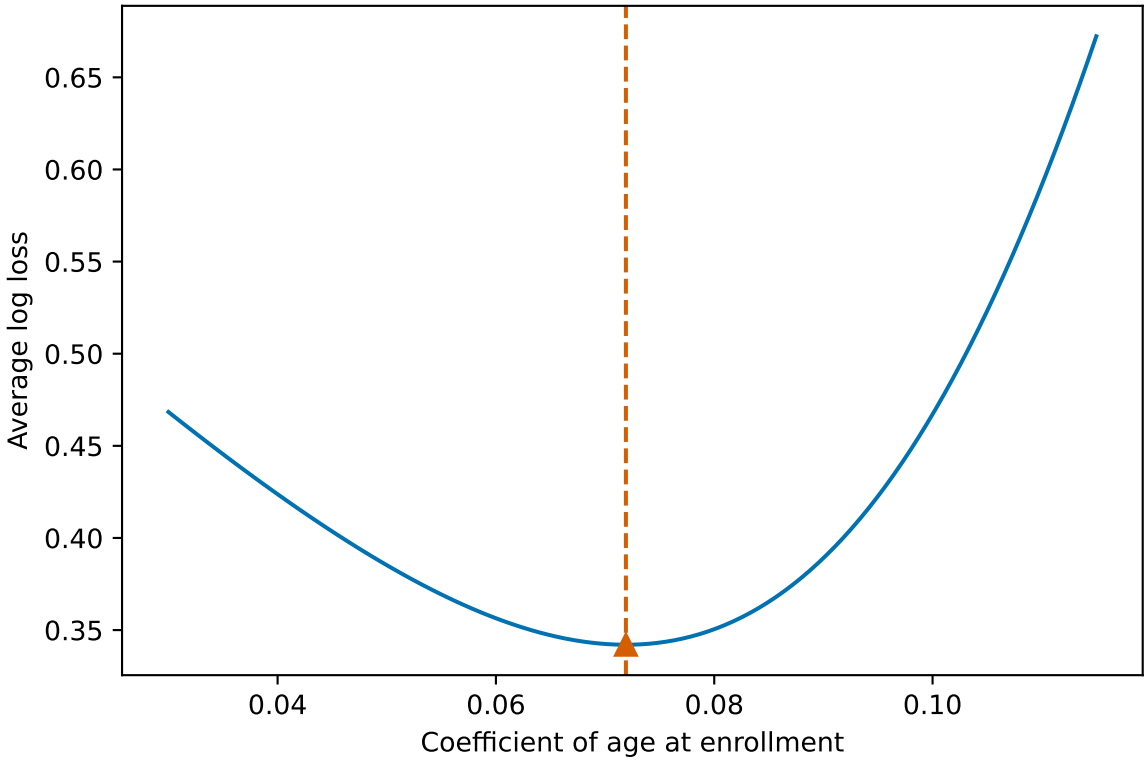


(a)

1.0.30 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()
```



(b)

Figure 8: The average log loss of Example 1.10, for the type A men, as the coefficient of age at enrollment varies, with the intercept[d] held at its fitted value. The numerical minimizer (triangle) and the maximum likelihood[e] coefficient (dashed line) coincide.

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle
[b] https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle
[c] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[d] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[e] https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle

Exercise 1.10 (The same loss, written two ways)**.** Exercise 1.3 gave the per-observation cost with labels $y \in \{0, 1\}$:

$$C(\eta, y) = -y \log \operatorname{expit}(\eta) - (1 - y) \log(1 - \operatorname{expit}(\eta)), \qquad \eta = \vec{\beta} \cdot \underset{\sim}{x}$$

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

Solution

Solution 1.10.
Check the two cases $y = 1$ and $y = 0$ separately. The steps use the definition $\operatorname{expit}(\eta) = 1/(1 + \mathrm{e}^{-\eta})$.
If $y = 1$, then $\tilde{y} = +1$, and

$$\begin{aligned}
C(\eta, 1) &= -1 \cdot \log \operatorname{expit}(\eta) - 0 \cdot \log(1 - \operatorname{expit}(\eta)) && \text{(substitute } y = 1\text{)} \\
&= -\log \operatorname{expit}(\eta) && \text{(the second term equals 0)} \\
&= -\log \frac{1}{1 + \mathrm{e}^{-\eta}} && \text{(definition of expit)} \\
&= \log(1 + \mathrm{e}^{-\eta}) && (\log(1/a) = -\log a) \\
&= \log\bigl(1 + \mathrm{e}^{-\tilde{y}\eta}\bigr) && (\tilde{y} = 1).
\end{aligned}$$

If $y = 0$, then $\tilde{y} = -1$. The identity[a] $1 - \operatorname{expit}(\eta) = \operatorname{expit}(-\eta)$ from part 2 of the sigmoid properties exercise gives

$$\begin{aligned}
C(\eta, 0) &= -0 \cdot \log \operatorname{expit}(\eta) - 1 \cdot \log(1 - \operatorname{expit}(\eta)) && \text{(substitute } y = 0\text{)} \\
&= -\log(1 - \operatorname{expit}(\eta)) && \text{(the first term equals 0)} \\
&= -\log \operatorname{expit}(-\eta) && (1 - \operatorname{expit}(\eta) = \operatorname{expit}(-\eta)) \\
&= -\log \frac{1}{1 + \mathrm{e}^{\eta}} && \text{(definition of expit)} \\
&= \log(1 + \mathrm{e}^{\eta}) && (\log(1/a) = -\log a) \\
&= \log\bigl(1 + \mathrm{e}^{-\tilde{y}\eta}\bigr) && (\tilde{y} = -1).
\end{aligned}$$

Both cases give the same expression[b], so the two formulas are the same loss function written under two label conventions. Under the $\{-1, +1\}$ convention, the loss depends on η and $\tilde{y}$ only through their product $\tilde{y}\eta$.

[a] https://morrison-lab.github.io/mds/algebra-basics.html#def-identity
[b] https://morrison-lab.github.io/mcs/expressions-algorithms.html#def-expression

Definition 1.4 (Margin of an observation)**.** The **margin** of an observation with label $y \in \{0, 1\}$ and log-odds[a] $\eta = \vec{\beta} \cdot \underset{\sim}{x}$ is

$$m \stackrel{\text{def}}{=} \tilde{y}\eta, \qquad \tilde{y} = 2y - 1 \in \{-1, +1\}.$$

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds

Example 1.11 (Margins of a man with a body mass index of 25)**.**
In the model of the logistic function example, a man with bmi = 25 has the log-odds[a] $\eta = -2.402$. If he has coronary heart disease, then $\tilde{y} = +1$, and his margin is $m = -2.402$, which is negative because his log-odds puts him on the side of the decision boundary where the model predicts no disease. If he does not have the disease, then $\tilde{y} = -1$, and his margin is $m = 2.402$, which is positive.

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds

Remark (Reading the margin)

Remark 1.2 (Reading the margin). The label $\tilde{y}$ is the label y recoded from 0 and 1 to -1 and $+1$ (Exercise 1.10), and the tilde keeps it distinct from y. The margin is positive when the log-odds[a] and the label have the same sign, so that the observation[b] is on the correct side of the decision boundary $\eta = 0$, and negative when it is on the wrong side. Its absolute value[c] $|m| = |\eta|$ is the distance of the log-odds from 0. The margin of a separating hyperplane, in the chapter on support vector machines, is a different quantity.

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds
[b] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-observation
[c] https://morrison-lab.github.io/mds/algebra-basics.html#def-absolute-value

Exercise 1.11 (The same loss, written two ways (part 2))**.** Use the setup of Exercise 1.10. Let $m = \tilde{y}\eta$ be the margin. Describe $\log(1 + \mathrm{e}^{-m})$ as a function of m: its monotonicity, its value at $m = 0$, and its behavior as $m \to +\infty$ and as $m \to -\infty$.

Solution

Solution 1.11. As a function of m, $\log(1+\mathrm{e}^{-m})$ is strictly decreasing, because e^{-m} is strictly decreasing and log is strictly increasing. A larger margin always gives a smaller loss, so the loss orders observations by their margins.
At $m = 0$ the loss is $\log 2$, which is the log loss of predicting the probability $\frac{1}{2}$ for the observed label.
As $m \to +\infty$, the loss decreases to 0, but it is strictly positive for every finite m. Increasing the margin of a correctly classified observation therefore always reduces the loss a little.
As $m \to -\infty$, the 1 in $1 + \mathrm{e}^{-m}$ becomes negligible, so $\log(1 + \mathrm{e}^{-m}) \approx \log \mathrm{e}^{-m}$, which equals $-m$. The loss therefore increases *linearly* with the size of a negative margin, whereas the least-squares loss $(1 - m)^2$ on the log-odds[a] increases quadratically. A single badly misclassified observation therefore has a bounded effect on the gradient under the log loss, and an unbounded effect under least squares on the log-odds.

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds

Exercise 1.12 (The same loss, written two ways (part 2b))**.** Use the setup of Exercise 1.10. Give the values of $\log(1 + \mathrm{e}^{-m})$ at $m = 0$ and at $m = \pm 2$ to three significant figures.

Solution

Solution 1.12. This continues Solution 1.11.

```
tlc_m <- c(-2, 0, 2)
tlc_exp <- exp(-tlc_m)
tlc_loss <- log1p(tlc_exp)
tlc_shown <- formatC(tlc_loss, digits = 3, format = "fg", flag = "#")
tlc_table <- tibble::tibble(m = tlc_m, loss = tlc_shown)
tlc_cols <- c("$m$", "$\\log(1 + \\e{-m})$")
knitr::kable(tlc_table, col.names = tlc_cols)
```

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

Exercise 1.13 (The same loss, written two ways (part 3))**.** Use the setup of Exercise 1.10.
The hinge loss $\max(0,\, 1-m)$ is the loss that support vector machines use. State one property that the hinge loss has and the logistic loss does not.

Solution

Solution 1.13. The hinge loss $\max(0, 1-m)$ equals 0 for every $m \geq 1$. 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.

Exercise 1.14 (The same loss, written two ways (part 3b))**.** Use the setup of Exercise 1.10.
State one property that the logistic loss has and the hinge loss does not.

Solution

Solution 1.14. This continues Solution 1.13.
The log loss is strictly positive for every finite m, so every observation contributes to the fit, with a gradient weight $\operatorname{expit}(-m)$ that decreases exponentially as $m \to +\infty$. It has derivatives of every order everywhere, whereas the hinge loss has a corner at $m = 1$ where it is not differentiable, so fitting it needs subgradient methods or constrained quadratic programming. The log loss is also the negative log-likelihood[a] of a Bernoulli[b] model, so minimizing it gives estimates[c] $\operatorname{expit}(\eta)$ of the probabilities, which are accurate when that model is correct. A support vector machine gives a linear function[d] of the features, and converting the linear function to a probability[e] needs a separate step after the fit.

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik
[b] https://morrison-lab.github.io/pds/random-variables.html#def-bernoulli
[c] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[d] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-linear-function
[e] https://morrison-lab.github.io/pds/probability-basics.html#def-probability

Exercise 1.15 (The same loss, written two ways (part 4))**.** Use the setup of Exercise 1.10.
The error rate[a] of a classifier is often the quantity of interest, and its per-observation cost is the zero–one loss $\delta(m \leq 0)$. Explain why it is not used for fitting.

[a] https://morrison-lab.github.io/sds/model-validation.html#def-error-rate

Solution

Solution 1.15. The zero–one loss $\delta(m \leq 0)$ takes only the values 0 and 1, so it is constant[a] in $\vec{\beta}$ except where some margin crosses 0. Its derivative with respect to $\vec{\beta}$ is zero wherever it exists, and it does not exist where $m = 0$, so the gradient gives no direction in which to change $\vec{\beta}$. The zero–one loss is also not convex[b], and minimizing the number of misclassified training points over classifiers with a hyperplane boundary is NP-hard, even approximately [2], abstract.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function
[b] https://morrison-lab.github.io/mds/algebra-sums.html#def-convex-function

Exercise 1.16 (The same loss, written two ways (part 4b))**.** Use the setup of Exercise 1.10.
Show that

$$\delta(m \leq 0) \leq \frac{1}{\log 2} \log(1 + \mathrm{e}^{-m}) \quad \text{for every } m.$$

Solution

Solution 1.16. This continues Solution 1.15.
To verify the upper bound, check the two cases $m \le 0$ and $m > 0$:

- If $m \le 0$, the left-hand side is 1. Because $e^{-m} \ge 1$, $\log(1 + e^{-m}) \ge \log 2$, so the right-hand side is at least $\log 2 / \log 2$, which equals 1.
- If $m > 0$, the left-hand side is 0, and the right-hand side is positive because $1 + e^{-m} > 1$.

The inequality therefore holds for every $m \in \mathbb{R}$.

Exercise 1.17 (The same loss, written two ways (part 4c))**.** Use the setup of Exercise 1.10.
Explain what reducing the right-hand side of the inequality of Exercise 1.16 therefore gains.

Solution

Solution 1.17. This continues Solution 1.15.
Averaging the inequality of Solution 1.16 over the training set shows that the mean[a] log loss divided by $\log 2$ is an upper bound on the error rate[b] on the training set. Minimizing the log loss therefore minimizes a differentiable, convex[c] upper bound on the error rate on the training set, which is called a *surrogate* for the zero–one loss.
The bound is not tight for positive margins. At $m = 2$, the rescaled log loss is $\log(1+e^{-2})/\log 2 = 0.183$, although the zero–one loss is 0. A fit can therefore reduce its log loss by increasing positive margins without changing any classification.

[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean
[b] https://morrison-lab.github.io/sds/model-validation.html#def-error-rate
[c] https://morrison-lab.github.io/mds/algebra-sums.html#def-convex-function

Theorem 1.4 (The log loss depends on the margin alone)**.** *Let $y \in \{0, 1\}$, $\tilde{y} = 2y - 1 \in \{-1, +1\}$ and $\eta = \vec{\beta} \cdot \underset{\sim}{x}$. Then the per-observation log loss,*

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

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

Proof

Proof. The equality is Solution 1.10, and the monotonicity is Solution 1.11. □

Example 1.12 (Evaluating the log loss from the log-odds and from the margin)**.** Consider the log-odds[a] $\eta = 2$, so $\pi = \operatorname{expit}(2)$ is 0.881, and compare the log loss under the two labels: $y = 1$, with $\tilde{y} = +1$ and margin $m = +2$, and $y = 0$, with $\tilde{y} = -1$ and margin $m = -2$.

1.0.31 R

```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
```

1.0.32 Python

```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 give 0.127 on the correctly classified observation ($m = +2$) and 2.13 on the misclassified one ($m = -2$).

Figure 9 compares four curves as functions of m.

```r
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",
    "Hinge: max(0, 1 - m)" = "dashed",
    "Least squares: (1 - m)^2" = "dotdash",
```

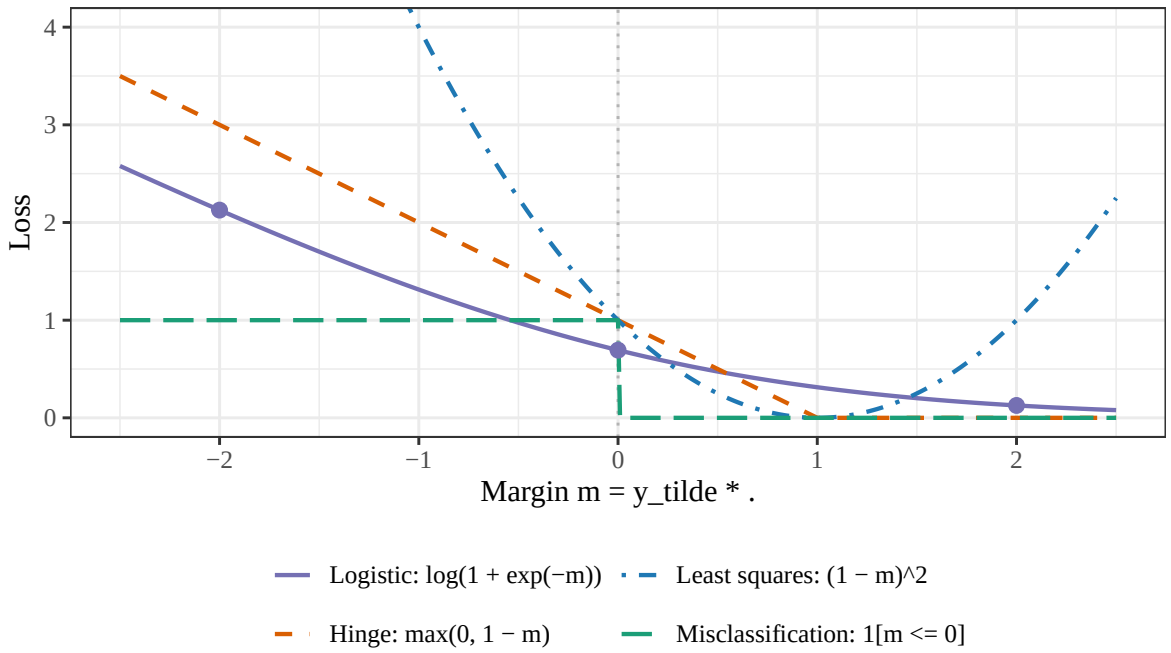


Figure 9: The log loss $\log(1+\mathrm{e}^{-m})$, the hinge loss $\max(0, 1-m)$, the least-squares loss $(1-m)^2$ and the misclassification loss $\delta(m \le 0)$ as functions of the margin $m = \tilde{y}\eta$. Dots mark the three log-loss values tabulated in Solution 1.12: $(-2, 2.13)$, $(0, 0.693)$ and $(2, 0.127)$.

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds

Remark (The log loss and least squares, compared through their derivatives)

Remark 1.3 (The log loss and least squares, compared through their derivatives). Fitting the log-odds[a] η to $\tilde{y} \in \{-1, +1\}$ by least squares uses the loss $(\tilde{y}-\eta)^2$, which equals $(1-m)^2$ because $\tilde{y}^2 = 1$. Its derivative with respect to η is $2(\eta - \tilde{y})$. The derivative of the log loss $\log(1+\mathrm{e}^{-\tilde{y}\eta})$ with respect to η is $-\tilde{y}\ \mathrm{expit}(-m)$.
Let $\tilde{y} = +1$:

- at $\eta = -4$, a confidently wrong prediction, the log-loss derivative is $-\mathrm{expit}(4)$, which is -0.982, and the least-squares derivative is $2(-4-1)$, which is -10;
- at $\eta = 4$, a confidently correct prediction, the log-loss derivative is $-\mathrm{expit}(-4)$, which is -0.018, and the least-squares derivative is $2(4-1)$, which is 6.

Squared error penalizes the prediction $\eta = 4$ because $\eta \neq \tilde{y}$, although it is on the correct side of the decision boundary, and it gives large weight to observations with large negative margins. The log loss does neither. Figure 9 plots the log, hinge, squared-error and zero–one misclassification losses as functions of the margin m.

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds

Exercise 1.18 (Three names for the same optimization)**.** The fit of Exercise 1.3 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], $\arg\min_{\vec{\beta}} \left[-\ell(\vec{\beta})\right]$;

- empirical risk minimization, $\arg\min_{\vec{\beta}} \hat{R}\big(\vec{\beta}\big)$, with

$$\hat{R} = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \pi_i).$$

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik

Solution

Solution 1.18. A point that maximizes ℓ minimizes $-\ell$, and dividing an objective by the positive constant[a] n does not change its minimizer:

$$\begin{aligned}
\arg\max_{\vec{\beta}}\ \ell(\vec{\beta}) &= \arg\min_{\vec{\beta}}\ \Big[-\ell(\vec{\beta})\Big] && \text{(negate the objective)}\\
&= \arg\min_{\vec{\beta}}\ \left[-\sum_i (y_i \log \pi_i + (1-y_i)\log(1-\pi_i))\right] && \text{(log-likelihood of the dataset)}\\
&= \arg\min_{\vec{\beta}}\ \sum_i [-y_i \log \pi_i - (1-y_i)\log(1-\pi_i)] && \text{(move the minus sign into the sum)}\\
&= \arg\min_{\vec{\beta}}\ \frac{1}{n}\sum_i [-y_i \log \pi_i - (1-y_i)\log(1-\pi_i)] && \text{(divide by } n>0\text{)}\\
&= \arg\min_{\vec{\beta}}\ \frac{1}{n}\sum_i L(y_i, \pi_i) && \text{(name the per-observation loss).}
\end{aligned}$$

The first line is maximum likelihood[b], the second and third minimize the negative log-likelihood[c], and the fourth and fifth are empirical risk minimization.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function
[b] https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle
[c] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik

Exercise 1.19 (Three names for the same optimization (part 2))**.** Use the setup of Exercise 1.18. State the per-observation loss $L(y_i, \pi_i)$.

Solution

Solution 1.19. The per-observation loss is

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

the log loss, which equals the cross-entropy loss by Theorem 1.2. A later exercise in this series shows where the second name comes from.

Exercise 1.20 (Three names for the same optimization (part 3))**.** Use the setup of Exercise 1.18. Identify which step introduces the factor $\frac{1}{n}$.

Solution

Solution 1.20. In Solution 1.18, the factor $\frac{1}{n}$ enters at the fourth line, the step that divides by n, and nowhere else.

Exercise 1.21 (Three names for the same optimization (part 4))**.** Use the setup of Exercise 1.18. State what the factor $\frac{1}{n}$ changes and what it does not change.

Solution

Solution 1.21. Multiplying by $\frac{1}{n}$ does not change the minimizer, because multiplying an objective by a positive constant[a] does not change its argmin[b]. It does not change the solutions of the score equations[c] either, because those equations set a gradient equal to zero, and a gradient multiplied by $\frac{1}{n}$ is zero exactly where the original gradient is zero.
The factor does change the scale of the gradient and of the second derivatives:

- the size of the gradient, and so any step size tuned to it;
- the second derivatives, and so any standard error[d] computed from them. A standard error computed from the second derivatives of $\frac{1}{n}C$ instead of C is too large by a factor of $\sqrt{n}$.

[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/intro-MLEs.html#def-score-equation
[d] https://morrison-lab.github.io/sds/estimation.html#def-SE

Exercise 1.22 (Three names for the same optimization (part 5))**.** Use the setup of Exercise 1.18. For distributions[a] $\underset{\sim}{p}$ and $\underset{\sim}{q}$ over the same outcomes, the cross-entropy[b] is $H(\underset{\sim}{p}, \underset{\sim}{q}) = -\sum_j p_j \log q_j$. Let $p = (1-y,\ y)$ be the distribution[c] that puts all its probability on the observed label $y \in \{0, 1\}$, and let $q = (1-\pi,\ \pi)$.
Show that $H(\underset{\sim}{p}, \underset{\sim}{q})$ equals the per-observation loss of Exercise 1.18.

[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 1.22. The two distributions[a] are over the two outcomes $\{0, 1\}$, so the sum has two terms:

$$\begin{aligned} H(\underset{\sim}{p}, \underset{\sim}{q}) &= -p_0 \log q_0 - p_1 \log q_1 && \text{(definition of cross-entropy)} \\ &= -(1-y)\log(1-\pi) - y \log \pi && \text{(substitute } \underset{\sim}{p} \text{ and } \underset{\sim}{q}\text{)} \\ &= L(y, \pi) && \text{(the log loss).} \end{aligned}$$

The log loss is therefore the cross-entropy[b] between the distribution[c] that puts all its probability on the observed label and the distribution that the model predicts, which is why it is also called the cross-entropy loss.

[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 1.23 (Three names for the same optimization (part 6))**.** Use the setup of Exercise 1.18. Show that $H(\underset{\sim}{p}, \underset{\sim}{q}) \geq H(\underset{\sim}{p}, \underset{\sim}{p})$ whenever $q_j > 0$ for every j with $p_j > 0$, with equality only when $\underset{\sim}{q} = \underset{\sim}{p}$.

Solution

Solution 1.23. Terms with $p_j = 0$ contribute 0 to both $H(\underset{\sim}{p}, \underset{\sim}{q})$ and $H(\underset{\sim}{p}, \underset{\sim}{p})$, by the convention $0 \log 0 = 0$, so let $S = \{j : p_j > 0\}$ and sum over S only. The difference of the two cross-entropies is

$$
\begin{aligned}
H(\underset{\sim}{p}, \underset{\sim}{q}) - H(\underset{\sim}{p}, \underset{\sim}{p}) &= -\sum_{j \in S} p_j \log q_j + \sum_{j \in S} p_j \log p_j && \text{(definition of cross-entropy)} \\
&= \sum_{j \in S} p_j (\log p_j - \log q_j) && \text{(combine the sums)} \\
&= \sum_{j \in S} p_j \log \frac{p_j}{q_j} && \text{(log of a quotient)} \\
&\stackrel{\text{def}}{=} D_{\mathrm{KL}}(\underset{\sim}{p} \,\|\, \underset{\sim}{q}).
\end{aligned}
$$

This difference is the Kullback–Leibler divergence[a] of q from p.
To show that it is nonnegative, use $\log x \le x - 1$ for $x > 0$, with equality only at $x = 1$:

$$
\begin{aligned}
-D_{\mathrm{KL}}(\underset{\sim}{p} \,\|\, \underset{\sim}{q}) &= \sum_{j \in S} p_j \log \frac{q_j}{p_j} && (\log(a/b) = -\log(b/a)) \\
&\le \sum_{j \in S} p_j \left(\frac{q_j}{p_j} - 1 \right) && (\log x \le x - 1) \\
&= \sum_{j \in S} q_j - \sum_{j \in S} p_j && \text{(distribute } p_j) \\
&\le 1 - \sum_{j \in S} p_j && \text{(the } q_j \text{ in } S \text{ sum to at most 1)} \\
&= 1 - 1 && \text{(the } p_j \text{ in } S \text{ sum to 1)} \\
&= 0.
\end{aligned}
$$

So $D_{\mathrm{KL}}(p \,\|\, q) \ge 0$, and $H(p, q) \ge H(p, p)$. Equality needs equality in both inequalities. The first needs $q_j / p_j = 1$ for every $j \in S$, and the second needs the q_j in S to sum to 1, so that $q_j = 0$ outside S. Together these give $\underset{\sim}{q} = \underset{\sim}{p}$.

[a] https://morrison-lab.github.io/pds/expectation.html#def-kl-divergence

Exercise 1.24 (Three names for the same optimization (part 7))**.** Use the setup of Exercise 1.18. Evaluate $H(\underset{\sim}{p}, \underset{\sim}{p})$ for the $\underset{\sim}{p}$ of Exercise 1.22.

Solution

Solution 1.24. Here $p = (1 - y, y)$ with $y \in \{0, 1\}$, so one entry of $\underset{\sim}{p}$ is 1 and the other is 0. With the convention $0 \log 0 = 0$, which is the limit of $p \log p$ as $p \to 0$,

$$
\begin{aligned}
H(\underset{\sim}{p}, \underset{\sim}{p}) &= -1 \log 1 - 0 \log 0 && \text{(definition of cross-entropy)} \\
&= -1 \cdot 0 - 0 && (\log 1 = 0 \text{ and } 0 \log 0 = 0) \\
&= 0.
\end{aligned}
$$

Exercise 1.25 (Three names for the same optimization (part 8))**.** Use the setup of Exercise 1.18. State the smallest achievable per-observation loss and when it is attained.

Solution

Solution 1.25. By Solution 1.22, the per-observation loss is $H(p, q)$. By Solution 1.23 and Solution 1.24, $H(p, q) \ge H(p, p) = 0$, with equality only when $\underset{\sim}{q} = \underset{\sim}{p}$, so the smallest per-observation loss is 0, and it is attained only when $\pi = y$.
No fitted model attains it. By part 1b of the sigmoid properties exercise, $\pi = \operatorname{expit}(\eta)$ is strictly between 0 and 1 for every real number[a] η, so the loss is strictly positive on every observation.

[a] https://morrison-lab.github.io/mcs/number-systems.html#def-real-numbers

Exercise 1.26 (Three names for the same optimization (part 9))**.** Use the setup of Exercise 1.18. Show that a distribution[a] over M outcomes has $H(\underset{\sim}{p}, \underset{\sim}{p}) \in [0, \log M]$.

[a] https://morrison-lab.github.io/pds/random-variables.html#def-cdf

Solution

Solution 1.26. Each term $-p_j \log p_j$ is nonnegative for $p_j \in [0,1]$, so $H(\underset{\sim}{p}, \underset{\sim}{p}) \geq 0$.
For the upper end, compare $\underset{\sim}{p}$ with the uniform distribution[a] $\underset{\sim}{u}$, which has $u_j = 1/M$ for every j, and use the inequality of Solution 1.23:

$$\begin{aligned}
0 &\leq D_{\mathrm{KL}}(\underset{\sim}{p} \,\|\, \underset{\sim}{u}) && \text{(the divergence is nonnegative)} \\
&= \sum_j p_j \log \frac{p_j}{1/M} && \text{(definition of } D_{\mathrm{KL}} \text{ and } u_j = 1/M) \\
&= \sum_j p_j (\log p_j + \log M) && \text{(log of a product)} \\
&= \sum_j p_j \log p_j + \log M \sum_j p_j && \text{(split the sum)} \\
&= -H(\underset{\sim}{p}, \underset{\sim}{p}) + \log M && \text{(definition of } H \text{ and } \textstyle\sum_j p_j = 1).
\end{aligned}$$

So $H(\underset{\sim}{p}, \underset{\sim}{p}) \leq \log M$.

[a] https://morrison-lab.github.io/pds/random-variables.html#def-cdf

Exercise 1.27 (Three names for the same optimization (part 9b))**.** Use the setup of Exercise 1.18. State which distributions[a] attain the minimum and the maximum of $H(\underset{\sim}{p}, \underset{\sim}{p})$ in Exercise 1.26.

[a] https://morrison-lab.github.io/pds/random-variables.html#def-cdf

Solution

Solution 1.27. This continues Solution 1.26.
Equality in the lower bound of Solution 1.26 holds only if every term equals 0, that is, only if every p_j is 0 or 1. The minimum 0 is therefore attained exactly at the distributions[a] that put all their probability on one outcome.
Equality in the upper bound holds exactly when $D_{\mathrm{KL}}(\underset{\sim}{p} \,\|\, \underset{\sim}{u}) = 0$, that is, when $\underset{\sim}{p} = \underset{\sim}{u}$, the uniform distribution[b]. A distribution with one certain outcome has entropy[c] 0, and the uniform distribution has the largest entropy, $\log M$.

[a] https://morrison-lab.github.io/pds/random-variables.html#def-cdf
[b] https://morrison-lab.github.io/pds/random-variables.html#def-cdf
[c] https://morrison-lab.github.io/pds/expectation.html#def-entropy

Theorem 1.5 (Cross-entropy is smallest at the true distribution)**.** *For distributions $\underset{\sim}{p}$ and $\underset{\sim}{q}$ over the same outcomes with $q_j > 0$ wherever $p_j > 0$,*

$$H(\underset{\sim}{p}, \underset{\sim}{q}) \geq H(\underset{\sim}{p}, \underset{\sim}{p}),$$

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[a] $D_{\mathrm{KL}}(\underset{\sim}{p} \,\|\, \underset{\sim}{q}) \geq 0$.

[a] https://morrison-lab.github.io/pds/expectation.html#def-kl-divergence

Proof

Proof. By Solution 1.23. □

Example 1.13 (Cross-entropy against a wrong guess)**.** Let $\underset{\sim}{p} = (0.9, 0.1)$ be the true distribution and $\underset{\sim}{q} = (0.5, 0.5)$ a guess.

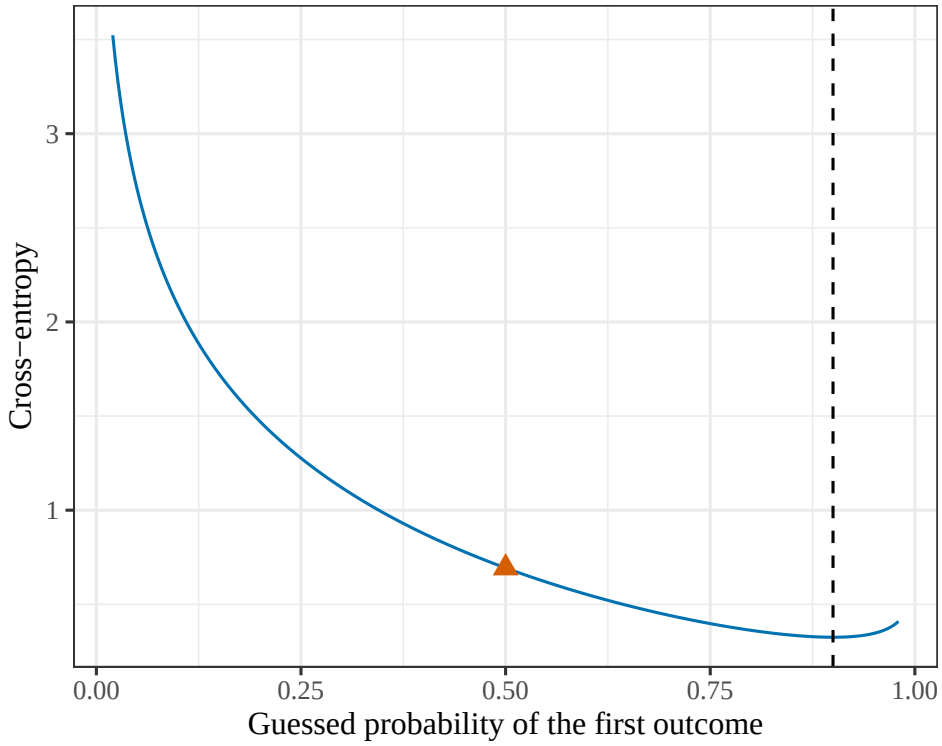
```
ce_p <- c(0.9, 0.1)
ce_q <- c(0.5, 0.5)
ce_pq <- -sum(ce_p * log(ce_q))
ce_pp <- -sum(ce_p * log(ce_p))
c(H_pq = ce_pq, H_pp = ce_pp, KL = ce_pq - ce_pp)
```

```
     H_pq      H_pp        KL
0.6931472 0.3250830 0.3680642
```

Here $H(\underset{\sim}{p}, \underset{\sim}{q}) = 0.693 \geq H(\underset{\sim}{p}, \underset{\sim}{p}) = 0.325$, as Theorem 1.5 says. The gap $D_{\mathrm{KL}}(\underset{\sim}{p} \,\|\, \underset{\sim}{q}) = 0.368$ is positive because $\underset{\sim}{q} \neq \underset{\sim}{p}$.
Figure 10 plots $H(\underset{\sim}{p}, \underset{\sim}{q})$ as q_1 varies.

1.0.33 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")
```

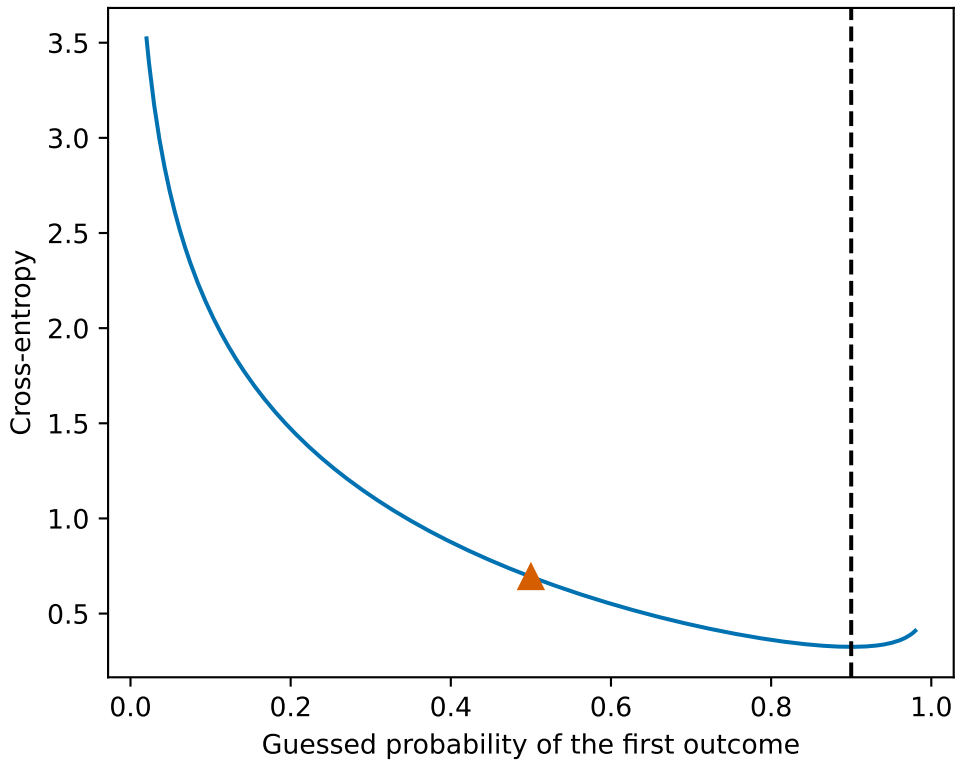


(a)

1.0.34 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
ax.set_xlabel("Guessed probability of the first outcome");
ax.set_ylabel("Cross-entropy");
plt.show()
```



(b)

Figure 10: The cross-entropy[a] $H(p, q)$ of Example 1.13 as the guess $q = (q_1, 1 - q_1)$ varies, when the truth is $p = (0.9, 0.1)$. It is lowest where the guess equals the truth (dashed), and the guess $(0.5, 0.5)$ of the example (triangle) lies above that minimum.

[a] https://morrison-lab.github.io/pds/expectation.html#def-cross-entropy

Theorem 1.6 (The range of the entropy). *A distribution $\underset{\sim}{p}$ over M outcomes has*

$$0 \leq H(\underset{\sim}{p}, \underset{\sim}{p}) \leq \log M.$$

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

Proof

Proof. By Solution 1.26 and Solution 1.27. □

Example 1.14 (Entropy of three distributions over four outcomes)**.**

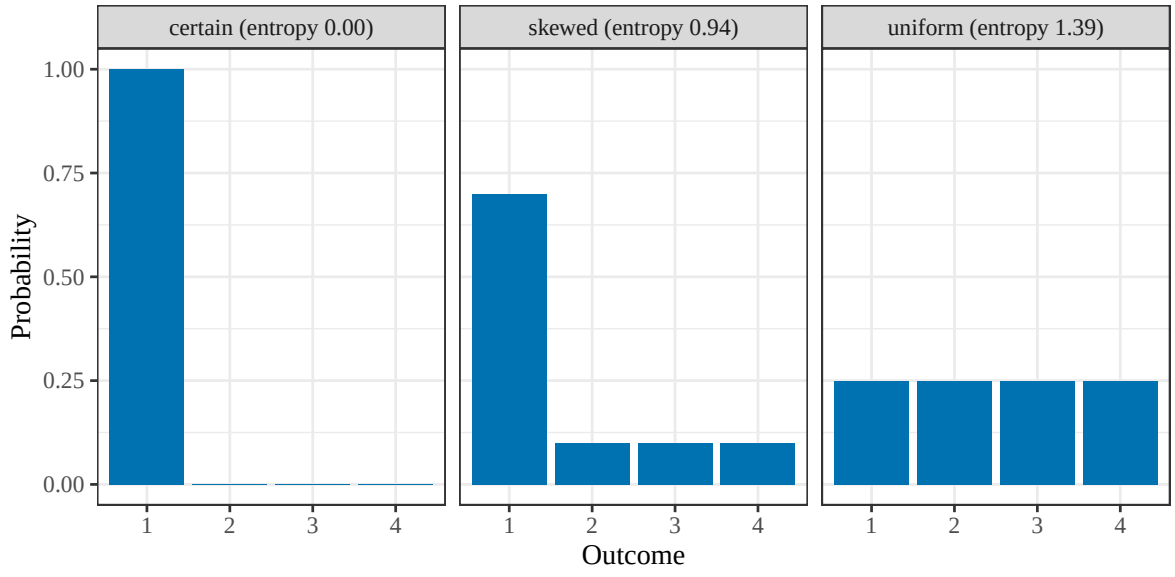
```
en_entropy <- function(p) -sum(ifelse(p > 0, p * log(p), 0))
en_dists <- list(
  certain = c(1, 0, 0, 0),
  skewed = c(0.7, 0.1, 0.1, 0.1),
  uniform = rep(0.25, 4)
)
vapply(en_dists, en_entropy, numeric(1))
```

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

With $M = 4$ outcomes, Theorem 1.6 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 0.94, between the other two.
Figure 11 plots the three cases.

1.0.35 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")
```



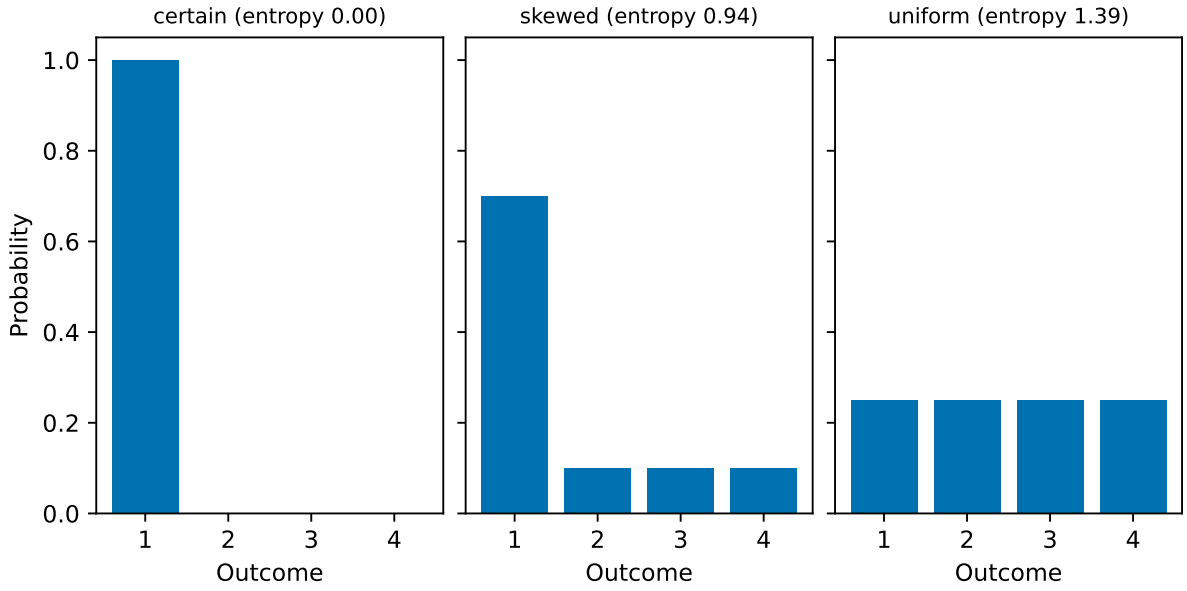
(a)

1.0.36 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()
```



(b)

Figure 11: The three distributions[b] of Example 1.14 put their probability on four outcomes. The entropy[c] rises from 0, when one outcome has all of the probability, to $\log 4$, when the distribution[d] spreads it evenly.

[a] https://morrison-lab.github.io/pds/random-variables.html#def-cdf
[b] https://morrison-lab.github.io/pds/random-variables.html#def-cdf
[c] https://morrison-lab.github.io/pds/expectation.html#def-entropy
[d] https://morrison-lab.github.io/pds/random-variables.html#def-cdf

2 Cross-entropy as a metric (lower is better)

Definition 2.1 (Cross-entropy of a classifier)**.** For labels $y_1, \dots, y_n \in \{1, \dots, K\}$, let $\underset{\sim}{q}_i$ be the probability mass function[a] over the K classes[b] that a classifier[c] gives for observation[d] i, and let $\underset{\sim}{p}_i$ be the one that puts probability[e] 1 on y_i. The **cross-entropy** of the classifier in base $b > 1$ is the sum of its cross-entropy losses, each computed with $\log_b$ in place of log,

$$\mathbb{H}_b \stackrel{\text{def}}{=} \sum_{i=1}^{n} \mathbb{H}(\underset{\sim}{p}_i, \underset{\sim}{q}_i). \tag{1}$$

Lower is better.

[a] https://morrison-lab.github.io/pds/random-variables.html#def-pmf
[b] https://morrison-lab.github.io/sds/classification.html#def-class
[c] https://morrison-lab.github.io/sds/classification.html#def-classification
[d] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-observation
[e] https://morrison-lab.github.io/pds/probability-basics.html#def-probability

Example 2.1 (The cross-entropy of a logistic regression on the spam emails)**.**
Fit the logistic regression of spam on the percent of words that are "george" and "you" in the spam data of the confusion-matrix exercise. Here $K = 2$, and the probability q_{iy_i} that the fit gives to the observed label of email i is its predicted probability[a] of spam π_i when the email is spam, and $1 - \pi_i$ otherwise. By Theorem 1.2, written in base 2, the cross-entropy loss of email i in bits[b] is $-\log_2 q_{iy_i}$, so the cross-entropy in bits is $\mathbb{H}_2 = -\sum_{i=1}^{n} \log_2 q_{iy_i}$.

```
spam_data <- readr::read_csv("data/spam-george-you.csv", show_col_types = FALSE)
spam_fit <- glm(spam ~ george + you, family = binomial, data = spam_data)
```

```
Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
spam_data |>
  dplyr::mutate(
    p_spam = fitted(spam_fit),
    p_true_class = dplyr::if_else(spam == 1, p_spam, 1 - p_spam)
  ) |>
  dplyr::summarize(
    n_emails = dplyr::n(),
    cross_entropy_bits = sum(-log2(p_true_class))
  )
```

```
# A tibble: 1 x 2
  n_emails cross_entropy_bits
     <int>              <dbl>
1     4601              3701.
```

[a] https://morrison-lab.github.io/sds/classification.html#def-predicted-probability
[b] https://morrison-lab.github.io/ids/entropy.html#def-bit

Remark (Units of cross-entropy)

Remark 2.1 (Units of cross-entropy). The unit of the cross-entropy depends on the base b of the logarithm:

- $b = 2$ gives bits[a].
- $b = e$ gives nats[b].
- $b = 10$ gives hartleys.

[a] https://morrison-lab.github.io/ids/entropy.html#def-bit
[b] https://morrison-lab.github.io/ids/entropy.html#def-nat

Exercise 2.1 (Reading the cross-entropy sum)**.** For labels $y_1, \ldots, y_n \in \{1, \ldots, K\}$, the cross-entropy of a classifier in base b is $\mathbb{H}_b = \sum_{i=1}^{n} \mathbb{H}(\underset{\sim}{p}_i, \underset{\sim}{q}_i)$, where $p_{ik} = 1$ when $k = y_i$ and $p_{ik} = 0$ otherwise. The theorem on the cross-entropy of two mass functions[a] writes each term as a sum over the K classes[b]:

$$\mathbb{H}_b = -\sum_{i=1}^{n} \sum_{k=1}^{K} p_{ik} \log_b q_{ik}. \tag{2}$$

The inner sum runs over all K classes, but only one of its terms can be nonzero.
Say which term.

[a] https://morrison-lab.github.io/pds/expectation.html#thm-cross-entropy-discrete
[b] https://morrison-lab.github.io/sds/classification.html#def-class

Solution

Solution 2.1. **Only the term for the observed label can be nonzero.** The probability mass function[a] $\underset{\sim}{p}_i$ puts probability[b] 1 on the observed label y_i, so $p_{ik} = 0$ for every $k \neq y_i$. Each of those terms is $0 \cdot \log_b q_{ik} = 0$, since a term with $p_{ik} = 0$ is taken to be 0 whatever q_{ik} is.

[a] https://morrison-lab.github.io/pds/random-variables.html#def-pmf
[b] https://morrison-lab.github.io/pds/probability-basics.html#def-probability

Exercise 2.2 (Reading the cross-entropy sum (part 2))**.** Keep $\underset{\sim}{p}_i$, $\underset{\sim}{q}_i$ and $\mathbb{H}_b$ from Exercise 2.1. Derive what Equation 2 reduces to once only the term for each observed label is kept.

Solution

Solution 2.2. Only the term for $k = y_i$ is left, and $p_{iy_i} = 1$, so

$$\mathbb{H}_b = -\sum_{i=1}^{n} \log_b q_{iy_i}.$$

The probabilities[a] that the classifier[b] assigns to the other classes[c] never appear. They matter only because the entries of $\underset{\sim}{q}_i$ sum to 1, so probability spent on a wrong class is probability taken from the observed one.

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-probability
[b] https://morrison-lab.github.io/sds/classification.html#def-classification
[c] https://morrison-lab.github.io/sds/classification.html#def-class

Exercise 2.3 (Reading the cross-entropy sum (part 3))**.** Keep $\underset{\sim}{p}_i$, $\underset{\sim}{q}_i$ and $\mathbb{H}_b$ from Exercise 2.1. What is the smallest value $\mathbb{H}_b$ can take, and what must the classifier do to reach it?

Solution

Solution 2.3. **The smallest value is 0.** A probability[a] is at most 1, so $\log_b q_{iy_i} \leq 0$, and each term of the sum in Solution 2.2 is at least 0. Equality needs $q_{iy_i} = 1$ for every i: the classifier must put all of its probability on the observed label, every time.

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-probability

Exercise 2.4 (Reading the cross-entropy sum (part 4))**.** Keep $\underset{\sim}{p}_i$, $\underset{\sim}{q}_i$ and $\mathbb{H}_b$ from Exercise 2.1. What is the largest value $\mathbb{H}_b$ can take?

Solution

Solution 2.4. **There is no largest value.** As $q_{iy_i} \to 0$, $-\log_b q_{iy_i} \to \infty$. One confident mistake is enough: a classifier[a] that assigns the observed label a probability[b] of 10^{-12} on a single observation[c] contributes 12 hartleys from that observation alone. This asymmetry is the metric working as intended. A score that only counts mistakes charges 1 for a mistake whether the classifier was hesitant or certain; cross-entropy[d] charges for the certainty.

[a] https://morrison-lab.github.io/sds/classification.html#def-classification
[b] https://morrison-lab.github.io/pds/probability-basics.html#def-probability
[c] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-observation
[d] https://morrison-lab.github.io/pds/expectation.html#def-cross-entropy

Exercise 2.5 (Reading the cross-entropy sum (part 5))**.** Keep p_i, q_i and $\mathbb{H}_b$ from Exercise 2.1. The base b names the units: 2 gives bits, e gives nats, 10 gives hartleys. Show that changing b multiplies $\mathbb{H}_b$ by a positive constant, and say what that implies for comparing two classifiers.

Solution

Solution 2.5. **The base is a scale factor.** By the change-of-base identity, $\log_b z = \ln z / \ln b$, so

$$\mathbb{H}_b = \frac{\mathbb{H}_e}{\ln b},$$

and $\ln b > 0$ for any $b > 1$. So the three unit choices give three numbers that are positive multiples of each other: 1.386 nats is 2 bits is 0.602 hartleys.
Multiplying every score by the same positive constant[a] leaves their order alone, so a classifier that beats another in bits beats it in nats too, and the base cannot change which model you pick. It changes only the units in which the number is expressed.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function

Theorem 2.1 (Cross-entropy of a classifier from the probabilities of the observed labels)**.** *The cross-entropy of a classifier in base b equals the sum over observations[a] of the negative base-b logarithm[b] of the probability[c] that the classifier[d] gives to the observed label,*

$$\mathbb{H}_b = -\sum_{i=1}^{n} \log_b q_{iy_i}. \tag{3}$$

[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-observation
[b] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-logarithm
[c] https://morrison-lab.github.io/pds/probability-basics.html#def-probability
[d] https://morrison-lab.github.io/sds/classification.html#def-classification

Proof

Proof. Sum Theorem 1.2, written in base b, over the observations[a]. Solution 2.2 works the same step from the expanded form in Exercise 2.1. □

[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-observation

Example 2.2.
A classifier[a] with $K = 3$ classes[b] gives $q_i = (\frac{1}{2}, \frac{1}{4}, \frac{1}{4})$ for one observation[c]. In bits[d] ($b = 2$), Equation 3 gives $-\log_2 \frac{1}{2} = 1$ if $y_i = 1$, and $-\log_2 \frac{1}{4} = 2$ if y_i is 2 or 3. The probability on the other two classes never enters.

[a] https://morrison-lab.github.io/sds/classification.html#def-classification

[b] https://morrison-lab.github.io/sds/classification.html#def-class
[c] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-observation
[d] https://morrison-lab.github.io/ids/entropy.html#def-bit

Theorem 2.2 (Cross-entropy of a classifier is nonnegative)**.** *The cross-entropy of a classifier in base b is nonnegative,*

$$\mathbb{H}_b \geq 0,$$

with equality exactly when $q_{iy_i} = 1$ for every observation[a] i.

[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-observation

Proof

Proof. By Theorem 2.1, $\mathbb{H}_b = -\sum_{i=1}^{n} \log_b q_{iy_i}$. Each q_{iy_i} is a probability[a], so $q_{iy_i} \leq 1$, and each term $-\log_b q_{iy_i}$ is at least 0, with equality only when $q_{iy_i} = 1$. A sum of nonnegative terms is 0 exactly when every term is 0. □

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-probability

Example 2.3 (Cross-entropy of two classifiers)**.**
On two observations[a], a classifier[b] assigns the observed labels the probabilities[c] 0.8 and 0.5. A second classifier assigns them 1 and 1.

```
cn_first <- c(0.8, 0.5)
cn_second <- c(1, 1)
cn_first_terms <- -log2(cn_first)
cn_second_terms <- -log2(cn_second)
cn_first_bits <- sum(cn_first_terms)
cn_second_bits <- sum(cn_second_terms)
cn_bits <- c(first = cn_first_bits, second = cn_second_bits)
cn_bits
```

```
   first   second
1.321928 0.000000
```

By Theorem 2.2 both values are at least 0. The first classifier scores 1.322 bits[d]. The second scores exactly 0, because it puts all its probability on the observed label every time.

[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-observation
[b] https://morrison-lab.github.io/sds/classification.html#def-classification
[c] https://morrison-lab.github.io/pds/probability-basics.html#def-probability
[d] https://morrison-lab.github.io/ids/entropy.html#def-bit

Theorem 2.3 (Changing the base of cross-entropy rescales it)**.** *For $b > 1$, the cross-entropy of a classifier in base b is its cross-entropy in base e divided by $\ln b$,*

$$\mathbb{H}_b = \frac{\mathbb{H}_e}{\ln b}.$$

Proof

Proof. The observed-label form is Theorem 2.1:

$$
\begin{aligned}
\mathbb{H}_b &= -\sum_{i=1}^{n} \log_b q_{iy_i} && \text{(observed-label form)} \\
&= -\sum_{i=1}^{n} \frac{\ln q_{iy_i}}{\ln b} && \text{(change of base, } \log_b z = \ln z / \ln b\text{)} \\
&= \frac{1}{\ln b}\left(-\sum_{i=1}^{n} \ln q_{iy_i}\right) && \text{(factor out the constant } 1/\ln b\text{)} \\
&= \frac{\mathbb{H}_e}{\ln b} && \text{(observed-label form in base } e\text{).}
\end{aligned}
$$

Since $\ln b > 0$, two classifiers[a] are ordered the same way in every base. □

[a] https://morrison-lab.github.io/sds/classification.html#def-classification

Example 2.4 (One classifier in three units)**.**
A classifier[a] assigns the observed labels the probabilities[b] 0.8 and 0.5 on two observations[c].

```
cb_p <- c(0.8, 0.5)
cb_bits_terms <- -log2(cb_p)
cb_nats_terms <- -log(cb_p)
cb_hartleys_terms <- -log10(cb_p)
cb_bits <- sum(cb_bits_terms)
cb_nats <- sum(cb_nats_terms)
cb_hartleys <- sum(cb_hartleys_terms)
cb_h <- c(bits = cb_bits, nats = cb_nats, hartleys = cb_hartleys)
cb_e <- exp(1)
cb_b <- c(2, cb_e, 10)
cb_ln_b <- log(cb_b)
cb_nats_over_ln_b <- cb_h[["nats"]] / cb_ln_b
names(cb_nats_over_ln_b) <- names(cb_h)
dplyr::bind_rows(
  value = cb_h,
  nats_over_ln_b = cb_nats_over_ln_b,
  .id = "row"
)
```

```
# A tibble: 2 x 4
  row             bits  nats hartleys
  <chr>          <dbl> <dbl>    <dbl>
1 value           1.32 0.916    0.398
2 nats_over_ln_b  1.32 0.916    0.398
```

The second row is $\mathbb{H}_e / \ln b$ for $b = 2, e, 10$, and by Theorem 2.3 it matches the first row. Every classifier's score is multiplied by the same $1/\ln b$, so the ordering of classifiers does not change with the base.

[a] https://morrison-lab.github.io/sds/classification.html#def-classification
[b] https://morrison-lab.github.io/pds/probability-basics.html#def-probability
[c] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-observation

Other names for cross-entropy, and two conventions to state

Software and statistics each renamed this quantity. **Log loss** and **logistic loss** are the same number, and so is the **negative log-likelihood** of the labels under the model, since Theorem 1.2 gives each loss as $-\log q_{iy_i}$, the log-likelihood of label y_i with its sign reversed. Generalized linear models add **deviance**, which for one label per observation[a] is twice the summed negative log-likelihood.

Settle two conventions before comparing one of these numbers with another source's. The notation choices page[b] explains both.

Which base. The definition of $\mathbb{H}_b$ leaves the base b free. Theorem 2.3 shows the base cannot change which of two classifiers you would pick, but it does change the number itself, so state the base whenever you quote one.

Summed or averaged. The cross-entropy of a classifier $\mathbb{H}_b$ sums over all n observations. Most software reports the mean, $\bar{\mathbb{H}}_b = \mathbb{H}_b/n$, under the same name, as Remark 1.1 says of the log loss. The perplexity is defined from the mean, not from the sum.

[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-observation
[b] https://morrison-lab.github.io/macros/notation-choices.html#sec-cross-entropy-conventions

Read

GeeksforGeeks explains the cross-entropy loss[a] [3].

[a] https://www.geeksforgeeks.org/machine-learning/what-is-cross-entropy-loss-function/

3 Tracing the origin of cross-entropy

Cross-entropy[8] comes from Claude Shannon's 1948 theory of communication [4], where it measures the average number of bits per symbol needed to encode a source with a code designed for another distribution[9].

A code designed for the source itself needs no more bits per symbol than a code designed for another distribution[10]. The difference is the Kullback–Leibler divergence[11]. The difference is zero only when the code's distribution equals the source's. The coding theorems behind these statements are in Source Coding[12] in *Information Theory for Data Science*.

Minimizing cross-entropy[13] over q moves q toward p, and fitting a logistic regression minimizes the sum[14] of these cross-entropies over the observations[15], with one q for each observation and all of them determined by the shared $\vec{\beta}$. The reference distribution[16] p puts probability 1 on the observed label and 0 on the other outcome, and the model's distribution q is $(1-\pi, \pi)$. A distribution that puts probability 1 on a single outcome has entropy[17] 0, so for a labeled training point the cross-entropy equals the Kullback–Leibler divergence[18].

The base of the logarithm sets the unit of information. Base 2 measures information in bits, and the natural logarithm measures it in *nats*. Because $\log_2 z = \ln z / \ln 2$, switching between them multiplies every term by a positive constant[19], so it does not change the minimizer. In this chapter, log is the natural logarithm, so the losses are measured in nats.

[8] https://morrison-lab.github.io/pds/expectation.html#def-cross-entropy
[9] https://morrison-lab.github.io/pds/random-variables.html#def-cdf
[10] https://morrison-lab.github.io/pds/random-variables.html#def-cdf
[11] https://morrison-lab.github.io/pds/expectation.html#def-kl-divergence
[12] https://morrison-lab.github.io/ids/source-coding.html#thm-wrong-code
[13] https://morrison-lab.github.io/pds/expectation.html#def-cross-entropy
[14] https://morrison-lab.github.io/mds/algebra-sums.html#def-summation
[15] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-observation
[16] https://morrison-lab.github.io/pds/random-variables.html#def-cdf
[17] https://morrison-lab.github.io/pds/expectation.html#def-entropy
[18] https://morrison-lab.github.io/pds/expectation.html#def-kl-divergence
[19] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function

Watch

- Weinberger fits logistic regression by maximum likelihood[a], from 30:50[b] [5], Lecture 11.
- Starmer also fits logistic regression by maximum likelihood[c] [6].
- Rocke fits logistic regression by maximum likelihood[d] (slides[e]) and compares models by deviance[f] and AIC[g] [7].
- Starmer explains cross-entropy[h] [8] 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://dmrocke.ucdavis.edu/Class/EPI204-Spring-2021/EPI204-2021-04-01.mp4
[e] https://dmrocke.ucdavis.edu/Class/EPI204-Spring-2021/EPI204-2%20Logistic%20Regression%20and%20Likelihood.pdf
[f] https://morrison-lab.github.io/sds/logistic-inference.html#def-deviance
[g] https://morrison-lab.github.io/sds/model-validation.html#def-aic
[h] https://www.youtube.com/watch?v=6ArSys5qHAU

Read

GeeksforGeeks explains binary cross-entropy[a] [9].

[a] https://www.geeksforgeeks.org/deep-learning/binary-cross-entropy-log-loss-for-binary-classification/

4 The gradient

In general there is no closed-form expression[20] for $\hat{\vec{\beta}}$, so it is computed iteratively by the methods of the gradient descent chapter. Those methods need the gradient of the total log loss. Exercise 4.1 shows that the derivative of the logistic function[21] cancels in this gradient, leaving an expression[22] of the same form as the least-squares gradient of linear regression. Later in this section, Figure 18 draws the total log loss on the WCGS data as a bowl over two slopes[23], with its gradient entries crossing zero at the bottom of the bowl and its Hessian[24] entries measuring how steep the bowl is.

Exercise 4.1 (The gradient, and the factor that cancels)**.** Let $z_i = \vec{\beta} \cdot \underset{\sim}{x}_i$, which is the log-odds[a] of $\pi_i = \text{expit}(z_i)$. Let $C_i = -\ell_i$ be observation i's contribution to the total log loss of Exercise 1.3, where ℓ_i is the log-likelihood contribution of one binary outcome in terms of the log-odds[b]: $\ell_i = y_i z_i + \log(1-\pi_i)$. Show that $\dfrac{\partial C_i}{\partial z_i} = \pi_i - y_i$, using $\text{expit}' = \text{expit}(1-\text{expit})$ from part 3 of the sigmoid properties exercise and without splitting into cases.

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-odds
[b] https://morrison-lab.github.io/sds/intro-MLEs.html#cor-bernoulli-loglik-contribution-logodds

Solution

Solution 4.1. Write $\text{expit}'(z_i)$ for the derivative of expit at z_i. Then

[20] https://morrison-lab.github.io/mcs/expressions-algorithms.html#def-closed-form-expression
[21] https://morrison-lab.github.io/mcs/functions.html#def-logistic-function
[22] https://morrison-lab.github.io/mcs/expressions-algorithms.html#def-expression
[23] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope
[24] https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian

$$
\begin{aligned}
\frac{\partial C_i}{\partial z_i} &= -\frac{\partial}{\partial z_i}(y_i z_i + \log(1 - \operatorname{expit}(z_i))) && (C_i = -\ell_i) \\
&= -y_i - \frac{\partial}{\partial z_i}\log(1 - \operatorname{expit}(z_i)) && \text{(derivative of a sum, and of } y_i z_i) \\
&= -y_i + \frac{\operatorname{expit}'(z_i)}{1 - \operatorname{expit}(z_i)} && \text{(chain rule)} \\
&= -y_i + \frac{\operatorname{expit}(z_i)(1 - \operatorname{expit}(z_i))}{1 - \operatorname{expit}(z_i)} && (\operatorname{expit}' = \operatorname{expit}(1 - \operatorname{expit})) \\
&= -y_i + \operatorname{expit}(z_i) && (\text{cancel } 1 - \operatorname{expit}(z_i)) \\
&= \pi_i - y_i && (\pi_i = \operatorname{expit}(z_i)).
\end{aligned}
$$

In terms of the log-odds[a] z_i, the label y_i appears only in the term $y_i z_i$, which is linear[b] in z_i. That term differentiates to y_i with no case split.
The derivative[c] of the cost with respect to the log-odds is the predicted probability[d] minus the observed label. It contains no exponential and no factor of expit′.

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-odds
[b] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-linear-function
[c] https://morrison-lab.github.io/mds/calculus-derivatives.html#def-derivative
[d] https://morrison-lab.github.io/sds/classification.html#def-predicted-probability

Exercise 4.2 (The gradient, and the factor that cancels (part 2))**.** Use the setup of Exercise 4.1. Hence give $\nabla_{\vec{\beta}} C$ for the whole dataset, first as a sum over observations and then in matrix form, where $\mathbf{D}(\mathbf{x})$ is the $n \times (p+1)$ design matrix and $\underset{\sim}{\pi}$ is the vector of fitted probabilities.

Solution

Solution 4.2. Because $z_i = \vec{\beta} \cdot \underset{\sim}{x}_i$, the gradient $\nabla_{\vec{\beta}} z_i$ is $\underset{\sim}{x}_i$. Then

$$
\begin{aligned}
\nabla_{\vec{\beta}} C &= \sum_{i=1}^{n} \nabla_{\vec{\beta}} C_i && \text{(gradient of a sum)} \\
&= \sum_{i=1}^{n} \frac{\partial C_i}{\partial z_i} \nabla_{\vec{\beta}} z_i && \text{(chain rule)} \\
&= \sum_{i=1}^{n} (\pi_i - y_i) \nabla_{\vec{\beta}} z_i && (\partial C_i / \partial z_i = \pi_i - y_i) \\
&= \sum_{i=1}^{n} (\pi_i - y_i) \underset{\sim}{x}_i && (\nabla_{\vec{\beta}} z_i = \underset{\sim}{x}_i) \\
&= \mathbf{D}(\mathbf{x})^\top (\underset{\sim}{\pi} - \underset{\sim}{y}) && (\text{row } i \text{ of } \mathbf{D}(\mathbf{x}) \text{ is } \underset{\sim}{x}_i{}^\top).
\end{aligned}
$$

Least squares has gradient $\mathbf{D}(\mathbf{x})^\top \left(\mathbf{D}(\mathbf{x})\vec{\beta} - \underset{\sim}{y}\right)$, up to a constant[a] factor, which is the same expression[b] with the fitted values $\mathbf{D}(\mathbf{x})\vec{\beta}$ of least squares in place of $\underset{\sim}{\pi}$. Both gradients are the transposed design matrix times a vector of fitted values minus labels. Only the map from $\vec{\beta}$ to the fitted values differs, and because $\underset{\sim}{\pi}$ is a nonlinear function of $\vec{\beta}$, setting the logistic gradient to zero gives equations with no closed-form solution[c] in general (the gradient descent chapter's exercise on why there is no closed form).
One observation at a time, the factor $\pi_i - y_i$ shows what each observation 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 $\pi_i \approx 1$, then $\pi_i - y_i$ is near zero, and the observation changes the gradient very little.
- If $y_i = 1$ and $\pi_i \approx 0$, the observation's term in the step adds about $\alpha \underset{\sim}{x}_i$ to $\vec{\beta}$. That term raises the log-odds[d] $\vec{\beta} \cdot \underset{\sim}{x}_i$ by about $\alpha \|\underset{\sim}{x}_i\|^2$, and so raises π_i toward 1.

- If $y_i = 0$ and $\pi_i \approx 1$, the observation's term subtracts about $\alpha \underset{\sim}{x}_i$ from $\vec{\beta}$, which lowers the log-odds and so lowers π_i toward 0.

The absolute value $|\pi_i - y_i|$ is the probability that the model gives to the label $1 - y_i$, which the observation does not have. Each observation therefore contributes to the gradient in the direction of its feature vector $\underset{\sim}{x}_i$, weighted by the probability that the model gives to the wrong label.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function
[b] https://morrison-lab.github.io/mcs/expressions-algorithms.html#def-expression
[c] https://morrison-lab.github.io/mcs/expressions-algorithms.html#def-closed-form-solution
[d] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds

Exercise 4.3 (The gradient, and the factor that cancels (part 3))**.** Use the setup of Exercise 4.1. In Exercise 4.1, the factor $\text{expit}'(z_i)$ cancels. Differentiate the squared-error cost $S_i = \frac{1}{2}(\pi_i - y_i)^2$ with respect to z_i instead.

Solution

Solution 4.3. By the chain rule,

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

The factor $\text{expit}'(z_i)$ stays in this derivative, because squared error has no logarithm whose derivative cancels it.

Exercise 4.4 (The gradient, and the factor that cancels (part 3b))**.** Use the setup of Exercise 4.1. Compare the derivative of Exercise 4.1 with the derivative of Exercise 4.3 at $z_i = 8$ with $y_i = 0$, an observation whose predicted probability is close to 1 while its observed label is 0.

Solution

Solution 4.4. This continues Solution 4.3.
At $z_i = 8$ with $y_i = 0$, the predicted probability $\pi_i = \text{expit}(8)$ is 0.99966, so

- the log loss has derivative $\pi_i - y_i$, which is 0.99966;
- squared error has derivative $\pi_i \cdot \pi_i(1 - \pi_i)$, which is 0.0003351.

Exercise 4.5 (The gradient, and the factor that cancels (part 3c))**.** Use the setup of Exercise 4.1. Report the ratio of the derivative of Exercise 4.1 to the derivative of Exercise 4.3 at $z_i = 8$ with $y_i = 0$.

Solution

Solution 4.5. This continues Solution 4.4.
At $z_i = 8$ with $y_i = 0$, the log-loss derivative of Solution 4.4 is about 2983 times the squared-error derivative of Solution 4.4, and the ratio $1/(\pi_i(1 - \pi_i))$ grows without bound as $|z_i|$ increases.
The model gives probability 0.9997 to the wrong class, yet squared error gives a gradient of only 0.000335, so gradient descent[a] on squared error changes $\vec{\beta}$ very little on this observation[b]. In Solution 4.1, the factor $1/(1 - \pi_i)$ from differentiating the logarithm cancels $\text{expit}'(z_i)$ (here $y_i = 0$; for $y_i = 1$ the factor is $1/\pi_i$), so the log loss has no such factor.
The solution to part 4c of the sigmoid properties exercise showed that expit is nearly constant[c] far from the decision boundary. Under the log loss, that flatness does not shrink the gradient on misclassified observations.

[a] https://morrison-lab.github.io/mds/optimization-gradient-descent.html#def-gradient-descent

[b] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-observation
[c] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function

Exercise 4.6 (The gradient, and the factor that cancels (part 4))**.** Use the setup of Exercise 4.1. Show that $\left\|\nabla_{\vec{\beta}} C_i\right\| < \|\underset{\sim}{x}_i\|$ for every $\vec{\beta}$.

Solution

Solution 4.6.
Because $\nabla_{\vec{\beta}} C_i = (\pi_i - y_i)\underset{\sim}{x}_i$,

$$\left\|\nabla_{\vec{\beta}} C_i\right\| = |\pi_i - y_i| \cdot \|\underset{\sim}{x}_i\|.$$

By part 1b of the sigmoid properties exercise, $\pi_i \in (0, 1)$, and $y_i \in \{0, 1\}$, so $|\pi_i - y_i| < 1$, and the bound follows.

Exercise 4.7 (The gradient, and the factor that cancels (part 4b))**.** Use the setup of Exercise 4.1. Explain what the bound of Exercise 4.6 guarantees.

Solution

Solution 4.7. This continues Solution 4.6.
Solution 4.6 guarantees that no single observation[a] can give a gradient larger in norm[b] than $\|\underset{\sim}{x}_i\|$, however wrong the predicted probability is. The contribution of an observation to a gradient step is bounded by the norm of its feature vector, which is one reason to standardize features that are on different scales before fitting.

[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-observation
[b] https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-euclidean-norm

Exercise 4.8 (The gradient, and the factor that cancels (part 5))**.** Use the setup of Exercise 4.1. At a minimizer of C, the gradient equals the zero vector, which gives the score equations[a] $\mathbf{D}(\mathbf{x})^\top(\underset{\sim}{\pi} - y) = \vec{0}$. Suppose the first column of $\mathbf{D}(\mathbf{x})$ is a column of ones, and the second is a binary indicator $\underset{\sim}{x}_i \in \{0, 1\}$ of membership in a group. State in words what each of those two equations implies about the fitted model.

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-score-equation

Solution

Solution 4.8. *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. For example, if 30 of 100 training observations have $y_i = 1$, the fitted probabilities sum to 30. A logistic regression with an intercept[a] therefore predicts the right number of positive outcomes on the training set as a whole, which is called being *calibrated in aggregate*. *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 group* where $x_i = 1$. Subtracting this equation from the intercept equation shows that it also holds in the complementary group, where $x_i = 0$.
Each indicator feature therefore makes the fit calibrated in aggregate within the group it defines. A categorical[b] predictor coded with indicator variables, together with an intercept, makes the fitted probabilities sum to the observed number of positive outcomes in every level of the predictor. The score equations[c] impose no condition on groups defined by variables that are not in the design matrix, and the fit can be miscalibrated on those groups.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept

[b] https://morrison-lab.github.io/sds/data.html#def-categorical-var
[c] https://morrison-lab.github.io/sds/intro-MLEs.html#def-score-equation

Exercise 4.9 (Leaving the intercept unpenalized)**.** Use the setup of Exercise 4.1.
ℓ_2 regularization replaces the empirical risk $\hat{R}$ by $\hat{R} + \lambda \sum_{k=1}^{p} \beta_k^2$, summing from $k = 1$ so that the intercept[a] β_0 is not penalized. Determine how penalizing β_0 would change the intercept score equation of Exercise 4.8.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept

Solution

Solution 4.9. The intercept[a] column of $\mathbf{D}(\mathbf{x})$ is a column of ones, so by Solution 4.2 the derivative of $\hat{R} = \frac{1}{n} C$ with respect to β_0 is

$$\frac{\partial \hat{R}}{\partial \beta_0} = \frac{1}{n} \sum_i (\pi_i - y_i).$$

Without a penalty on β_0, setting this derivative to zero gives the intercept score equation of Exercise 4.8, $\sum_i \pi_i = \sum_i y_i$: the fitted probabilities sum to the observed number of positive outcomes.
Now add $\lambda \beta_0^2$ to the objective. At the minimizer, the derivative with respect to β_0 is zero:

$$\begin{aligned}
\frac{1}{n} \sum_i (\pi_i - y_i) + 2\lambda \hat{\beta}_0 = 0 \qquad & \text{(derivative of the penalized objective)} \\
\frac{1}{n} \sum_i (\pi_i - y_i) = -2\lambda \hat{\beta}_0 \qquad & \text{(subtract } 2\lambda \hat{\beta}_0\text{)} \\
\sum_i (\pi_i - y_i) = -2n\lambda \hat{\beta}_0 \qquad & \text{(multiply by } n\text{)} \\
\sum_i \pi_i - \sum_i y_i = -2n\lambda \hat{\beta}_0 \qquad & \text{(split the sum)} \\
\sum_i \pi_i = \sum_i y_i - 2n\lambda \hat{\beta}_0 \qquad & \text{(add } \textstyle\sum_i y_i\text{)}.
\end{aligned}$$

The fitted probabilities no longer sum to the observed count: their sum is too small when $\hat{\beta}_0 > 0$ and too large when $\hat{\beta}_0 < 0$.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept

Exercise 4.10 (Leaving the intercept unpenalized (part 10b))**.** Use the setup of Exercise 4.1.
Use the result of Exercise 4.9 to explain why β_0 is left unpenalized.

Solution

Solution 4.10. This continues Solution 4.9.
Penalizing β_0 moves the fitted probabilities away from the observed count (Solution 4.9). When the features are centered, β_0 is the log-odds[a] that the model gives an observation with average features, which is usually negative for an outcome with prevalence[b] below $\frac{1}{2}$. Shrinking β_0 toward 0 then moves the average predicted probability toward $\frac{1}{2}$, so a rare outcome is predicted too often and a common outcome too rarely.
A second reason is that a penalty on β_0 makes the fit depend on where the features' origin is. A penalty on a slope[c] β_k penalizes a large rate of change of the log-odds with respect to x_k, but the intercept is not a rate of change. Without a penalty on β_0, centering a feature, $x_k \mapsto x_k - \bar{x}_k$, changes $\hat{\beta}_0$ and leaves every slope and every fitted probability unchanged. With a penalty on β_0, the penalty depends on the size of β_0, so the fitted probabilities depend on the choice of origin for the features.

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds
[b] https://morrison-lab.github.io/sds/classification.html#def-prevalence
[c] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope

Definition 4.1 (Residual of a logistic regression)**.** The **residual** of observation[a] i in logistic regression is its predicted probability[b] minus its label,

$$\pi_i - y_i, \qquad \pi_i = \operatorname{expit}\left(\vec{\beta} \cdot \underset{\sim}{x}_i\right).$$

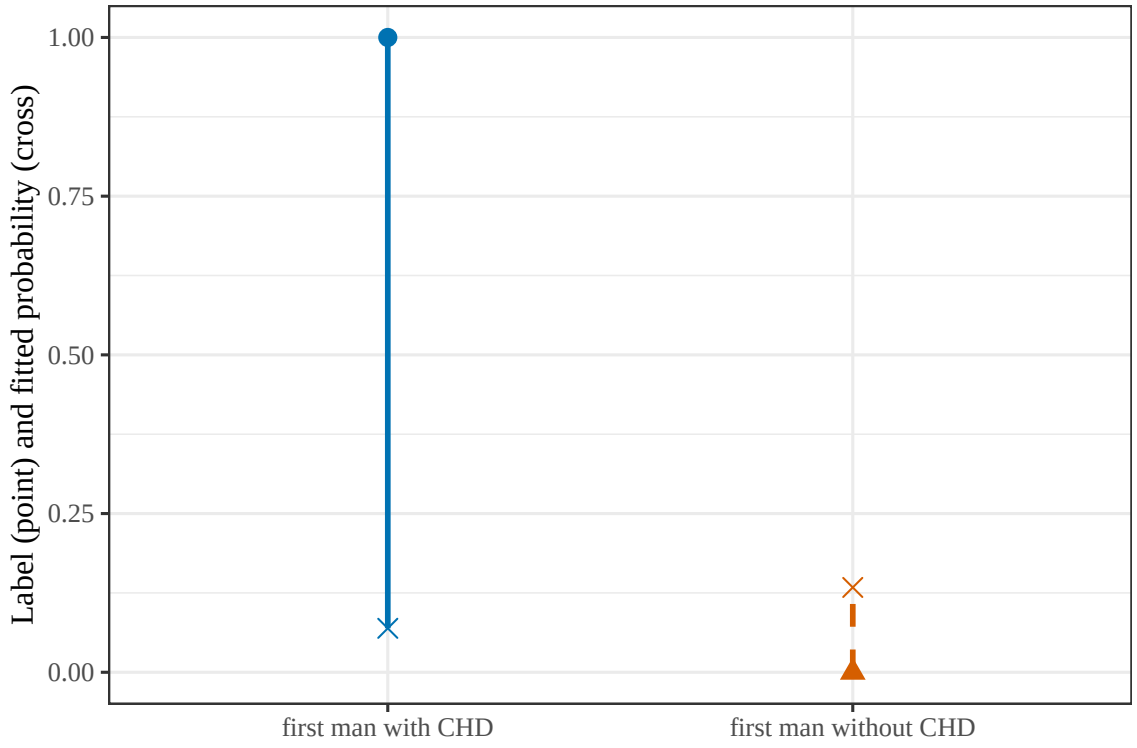
It is the negative of the response residual $y_i - \pi_i$, and it is in $(-1, 1)$.

[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-observation
[b] https://morrison-lab.github.io/sds/classification.html#def-predicted-probability

Example 4.1 (Residuals of two men in the WCGS data)**.** In the model of the logistic function example, 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 $\pi_i - y_i$ is -0.931. The first man without it has $y_i = 0$ and $\pi_i = 0.134$, so his residual is 0.134.
Figure 12 draws one vertical segment for each man.

4.0.1 R

```
library(ggplot2)
ggplot(lrv_pts) +
  aes(x = man, colour = man) +
  geom_segment(
    aes(xend = man, y = label, yend = prob, linetype = man),
    linewidth = 1
  ) +
  geom_point(aes(y = label, shape = man), size = 3) +
  geom_point(aes(y = prob), shape = 4, size = 3) +
  scale_colour_manual(values = c("#0072B2", "#D55E00")) +
  scale_shape_manual(values = c(16, 17)) +
  scale_linetype_manual(values = c("solid", "dashed")) +
  scale_y_continuous(limits = c(0, 1)) +
  labs(x = NULL, y = "Label (point) and fitted probability (cross)") +
  theme(legend.position = "none")
```

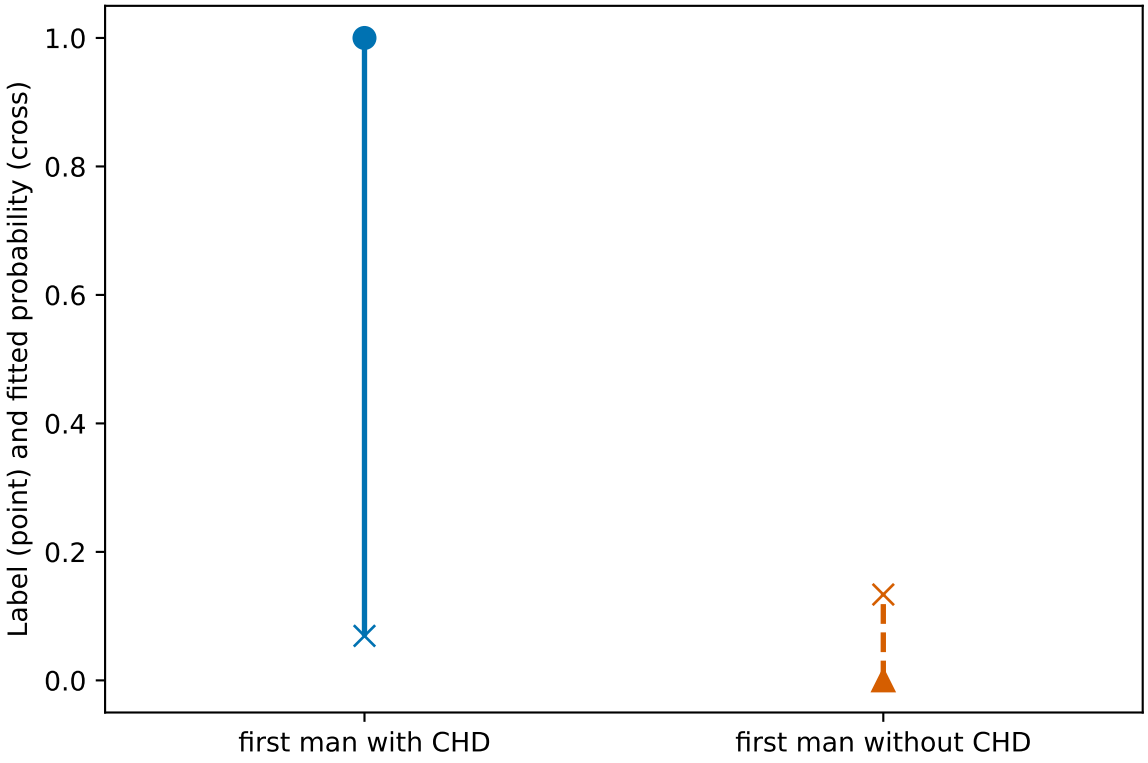


(a)

4.0.2 Python

```
import matplotlib.pyplot as plt

pts_lrv = r.lrv_pts
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
for i, (colour, marker, style) in enumerate([("#0072B2", "o", "-"), ("#D55E00", "^", "--")]):
    ax.vlines(i, pts_lrv["label"][i], pts_lrv["prob"][i], color=colour, linestyle=style, linewidth=2);
    ax.plot(i, pts_lrv["label"][i], color=colour, marker=marker, markersize=8, linestyle="");
    ax.plot(i, pts_lrv["prob"][i], color=colour, marker="x", markersize=8, linestyle="");
ax.set_xticks([0, 1], ["first man with CHD", "first man without CHD"]);
ax.set_xlim(-0.5, 1.5);
ax.set_ylim(-0.05, 1.05);
ax.set_ylabel("Label (point) and fitted probability (cross)");
plt.show()
```



(b)

Figure 12: The two men of Example 4.1, with each man's label and the probability that the model gives to a label of 1. The vertical distance between them is the size of his residual.

[a] https://morrison-lab.github.io/lds/chapters/data/wcgs.csv

Theorem 4.1 (Derivative of the log loss with respect to the log-odds)**.** *Let* $z_i = \vec{\beta} \cdot \underset{\sim}{x}_i$ *and* $\pi_i = \text{expit}(z_i)$*. The log loss* C_i *of observation* i *has derivative*

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

the residual of Definition 4.1.

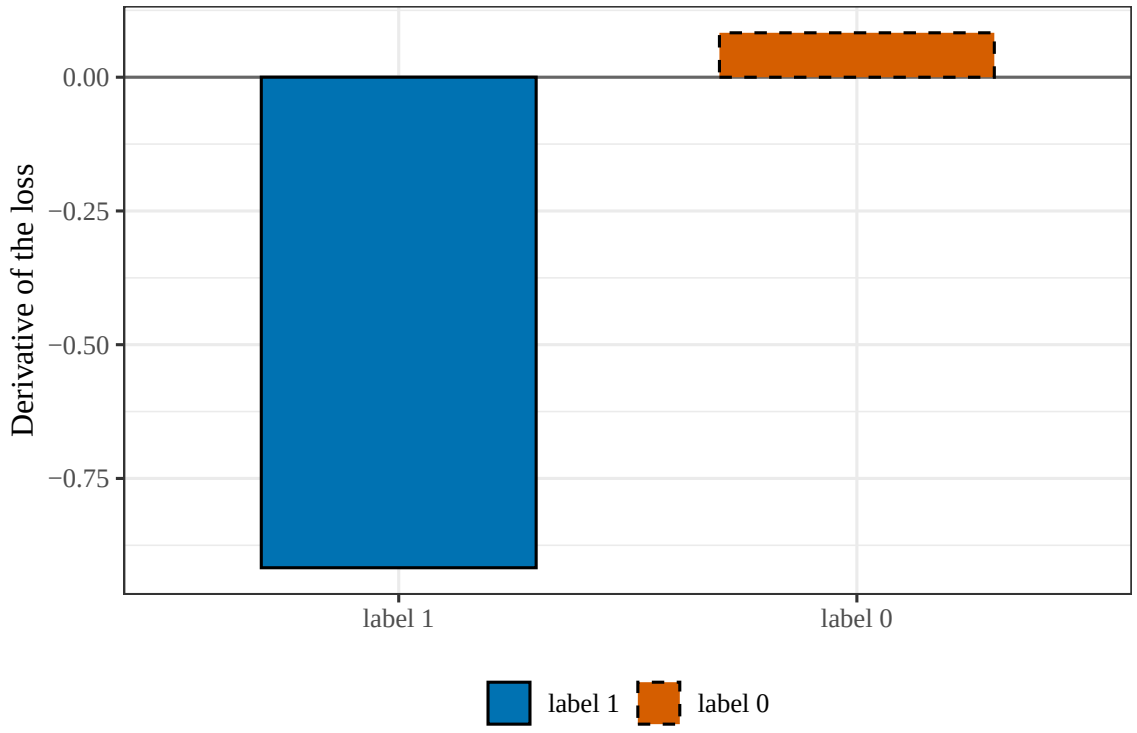
Proof

Proof. By Solution 4.1. □

Example 4.2 (The residual of a man with a body mass index of 25)**.** In the model of the logistic function example, the man with bmi = 25 of the logistic symmetry example has $\pi_i = 0.083$. If he has coronary heart disease, then $y_i = 1$, and $\frac{\partial C_i}{\partial z_i} = \pi_i - 1$, which is -0.917. If he does not, then $y_i = 0$, and the derivative is π_i, which is 0.083. The derivative is about 11 times as large in magnitude when the label is 1, because the model gives his disease a low probability.
Figure 13 plots the derivative under each of the two cases.

4.0.3 R

```
library(ggplot2)
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)
```

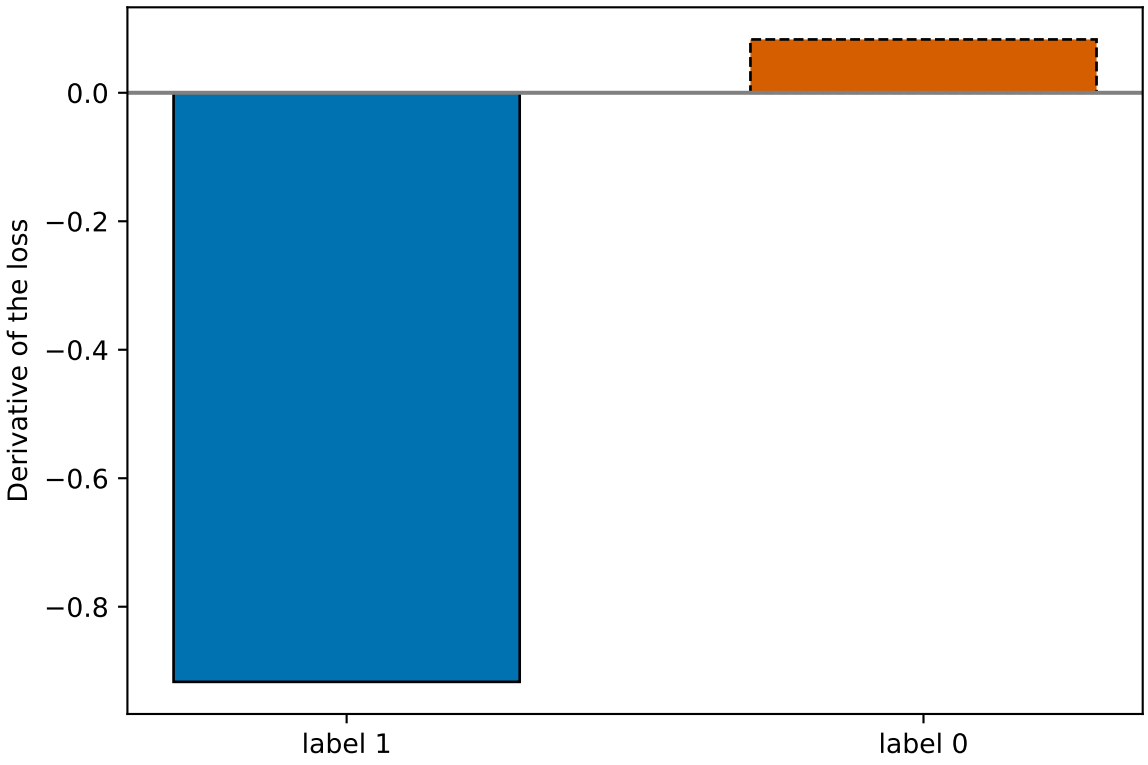


(a)

4.0.4 Python

```
import matplotlib.pyplot as plt

deriv_lrf = r.lr25_deriv
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.axhline(0, color="grey");
ax.bar(["label 1"], [deriv_lrf[0]], color="#0072B2", edgecolor="black", width=0.6);
ax.bar(["label 0"], [deriv_lrf[1]], color="#D55E00", edgecolor="black", linestyle="--", width=0.6);
ax.set_ylabel("Derivative of the loss");
plt.show()
```



(b)

Figure 13: The derivative[a] of the loss with respect to the log-odds[b] for the man of Example 4.2, under each of the two labels. The label 1 is much farther from the probability that the model gives him, so its derivative is much larger in magnitude.

[a] https://morrison-lab.github.io/mds/calculus-derivatives.html#def-derivative
[b] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds

Theorem 4.2 (Gradient of the total log loss)**.** *The gradient of the total log loss with respect to* $\vec{\beta}$ *is*

$$\begin{aligned}\nabla C(\vec{\beta}) &= \sum_{i=1}^{n} (\pi_i - y_i)\underset{\sim}{x}_i \\ &= \mathbf{D}^\top(\underset{\sim}{\pi} - \underset{\sim}{y}),\end{aligned}$$

where $\mathbf{D}$ *is the design matrix, with the column of ones that holds the intercept*[a]*, and* $\underset{\sim}{\pi}$ *is the vector of fitted probabilities.*

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept

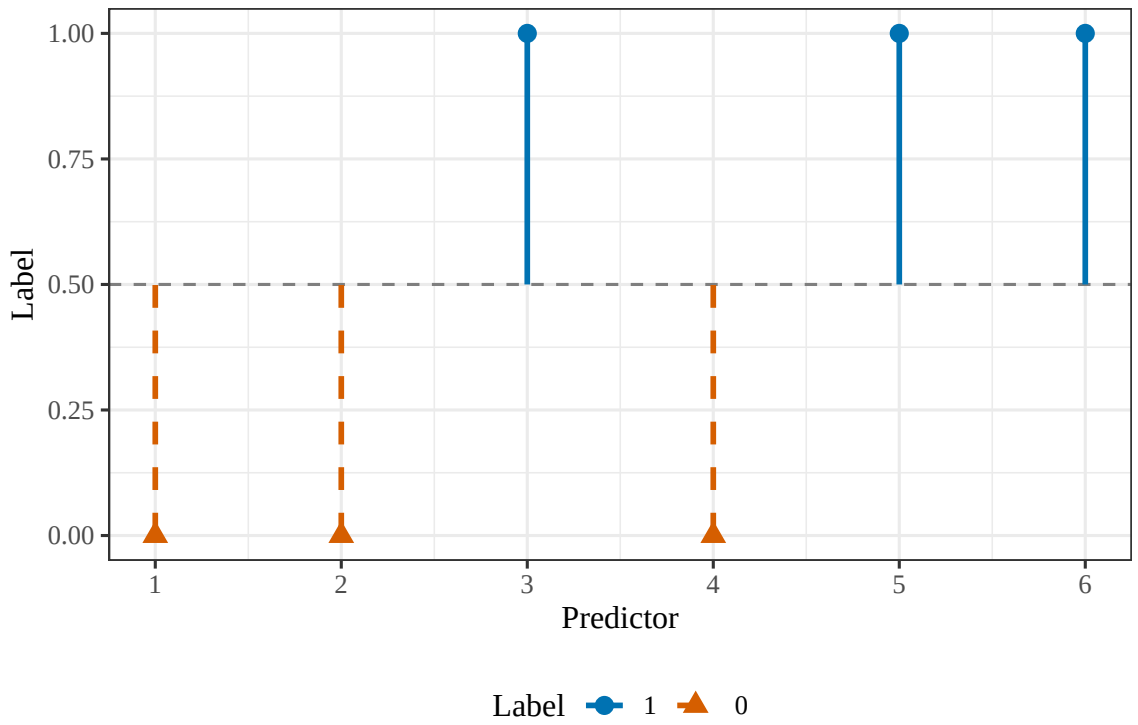
Proof

Proof. By Theorem 4.1 and the chain rule, as in Solution 4.2. □

Example 4.3 (The gradient at zero coefficients)**.** Consider six points with predictor values $1, \dots, 6$ and labels $0, 0, 1, 0, 1, 1$, at $\vec{\beta} = \vec{0}$. Every fitted probability is then $\pi_i = \text{expit}(0)$, which is $1/2$. The intercept[a] entry of $\mathbf{D}^\top(\underset{\sim}{\pi} - y)$ is $\sum_{i=1}^{n}(1/2 - y_i)$, which is 0. The slope[b] entry is $\sum_{i=1}^{n}(1/2 - y_i)x_i$, which is -3.5. Because the slope entry is negative, the loss decreases as the slope increases from 0, so a first step of gradient descent[c] increases the slope.
Figure 14 draws one vertical segment for each of the six points.

4.0.5 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")
```

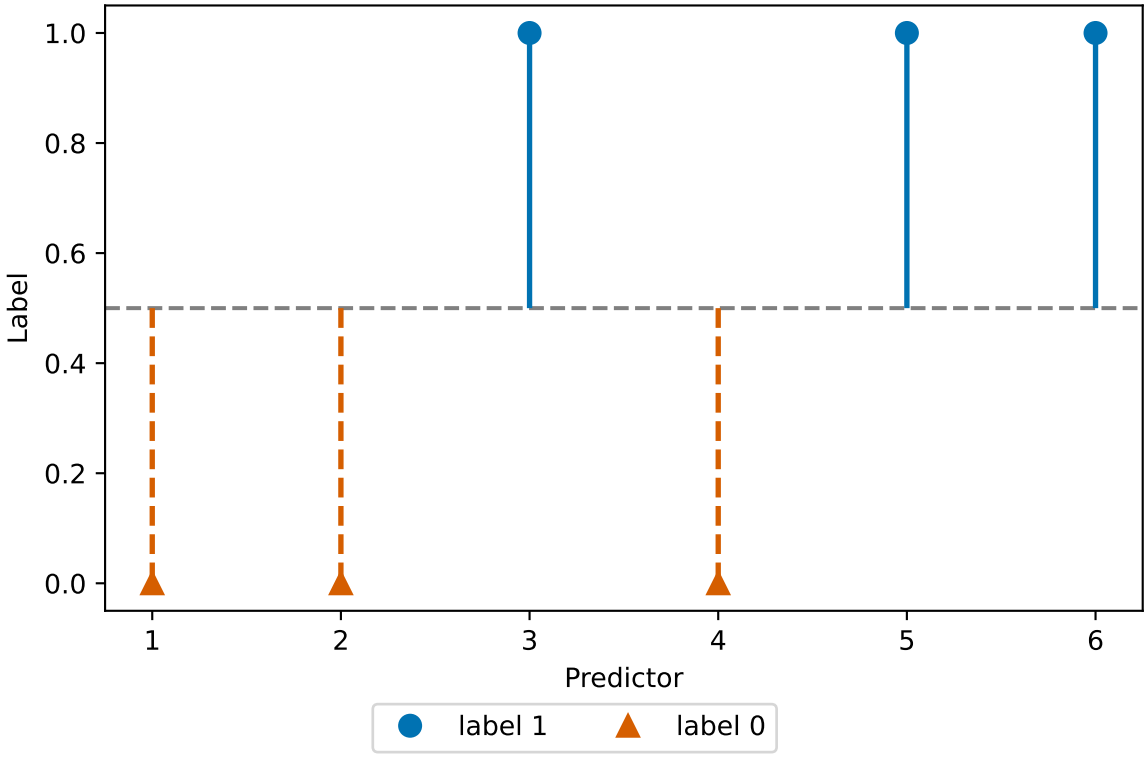


(a)

4.0.6 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");
fig.legend(loc="outside lower center", ncol=2);
plt.show()
```



(b)

Figure 14: The six points of Example 4.3, with the fitted probability 1/2 that they all have at zero coefficients (dashed line). Each vertical segment is the distance between a point's label and that probability, which is the size of its residual.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[b] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope
[c] https://morrison-lab.github.io/mds/optimization-gradient-descent.html#def-gradient-descent

Theorem 4.3 (Hessian of the total log loss)**.** *The Hessian*[a] *of the total log loss with respect to* $\vec{\beta}$ *is*

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

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

[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-semidefinite
[c] https://morrison-lab.github.io/mds/algebra-sums.html#def-convex-function

Proof

Proof. By Theorem 4.2, $\nabla C(\vec{\beta}) = \sum_{i=1}^{n}(\pi_i - y_i)\underset{\sim}{x}_i$. Only π_i depends on $\vec{\beta}$, and by the logistic derivative theorem and the chain rule the gradient of π_i is $\nabla \pi_i = \pi_i(1-\pi_i)\underset{\sim}{x}_i$. Differentiating each term of the sum gives $\pi_i(1-\pi_i)\underset{\sim}{x}_i\underset{\sim}{x}_i^\top$, and the sum of these is $\mathbf{D}^\top W \mathbf{D}$. □

Example 4.4 (The Hessian at zero coefficients)**.** For the six points of Example 4.3 at $\vec{\beta} = \vec{0}$, every $\pi_i = 1/2$, so $W = \frac{1}{4}I$.

4.0.7 R

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

```
          x_six
     1.50  5.25
x_six 5.25 22.75
```

4.0.8 Python

```
import numpy as np

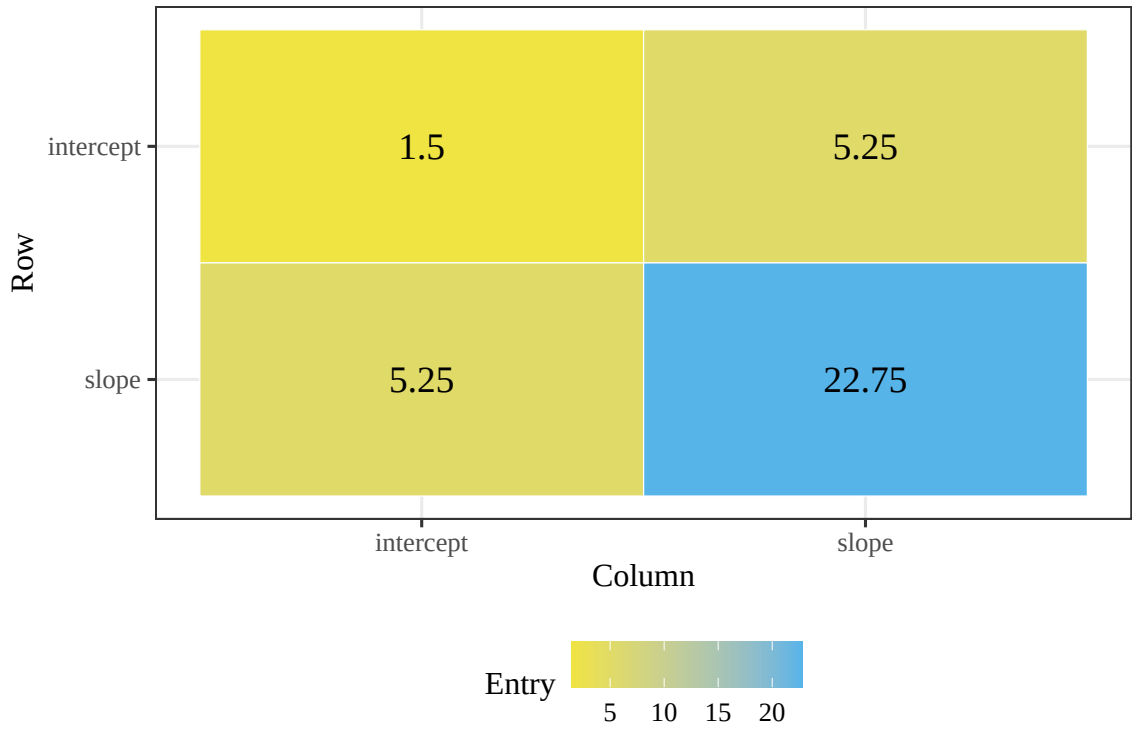
# the six points, with the column of ones for the intercept
x_six = np.arange(1, 7)
design_six = np.column_stack([np.ones(6), x_six])
# fitted probabilities at zero coefficients, and their weights p (1 - p)
p_zero = np.full(6, 0.5)
w_zero = p_zero * (1 - p_zero)
# the Hessian: design' W design
print(design_six.T @ (w_zero[:, None] * design_six))
```

```
[[ 1.5   5.25]
 [ 5.25 22.75]]
```

The intercept[a] entry is $\frac{1}{4} \cdot 6$, which is 1.5. The off-diagonal entry is $\frac{1}{4}\sum_i x_i$, which is 5.25. The slope[b] entry is $\frac{1}{4}\sum_i x_i^2$, which is 22.75.
Figure 15 displays the entries.

4.0.9 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")
```

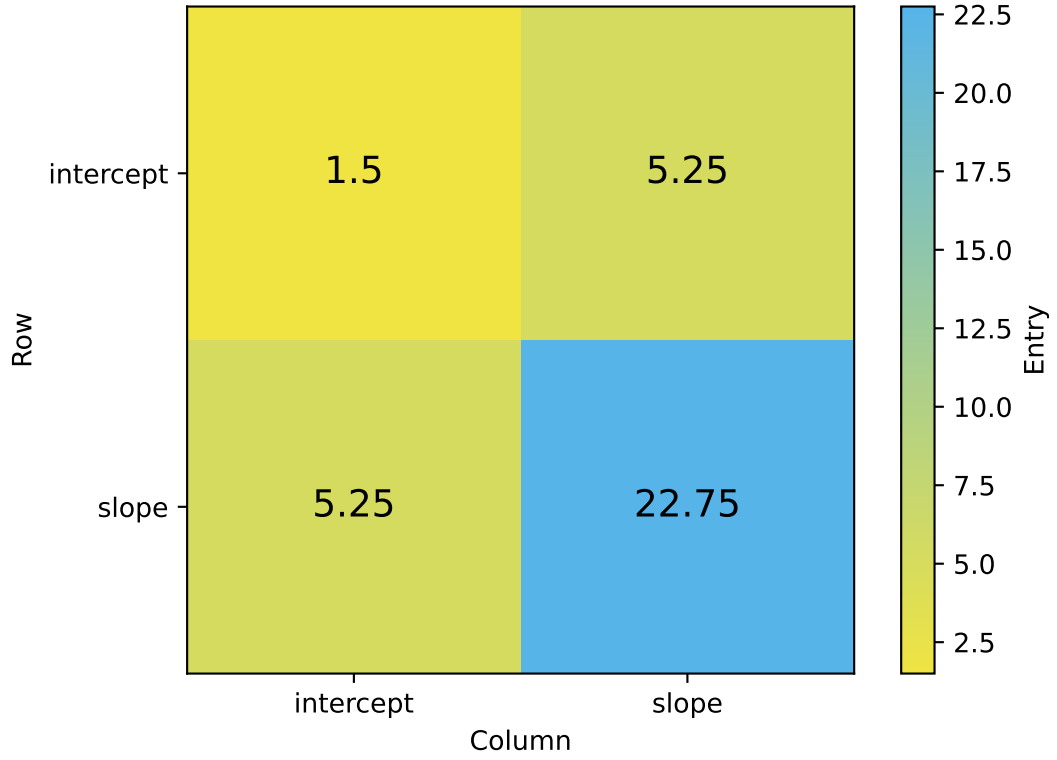


(a)

4.0.10 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()
```



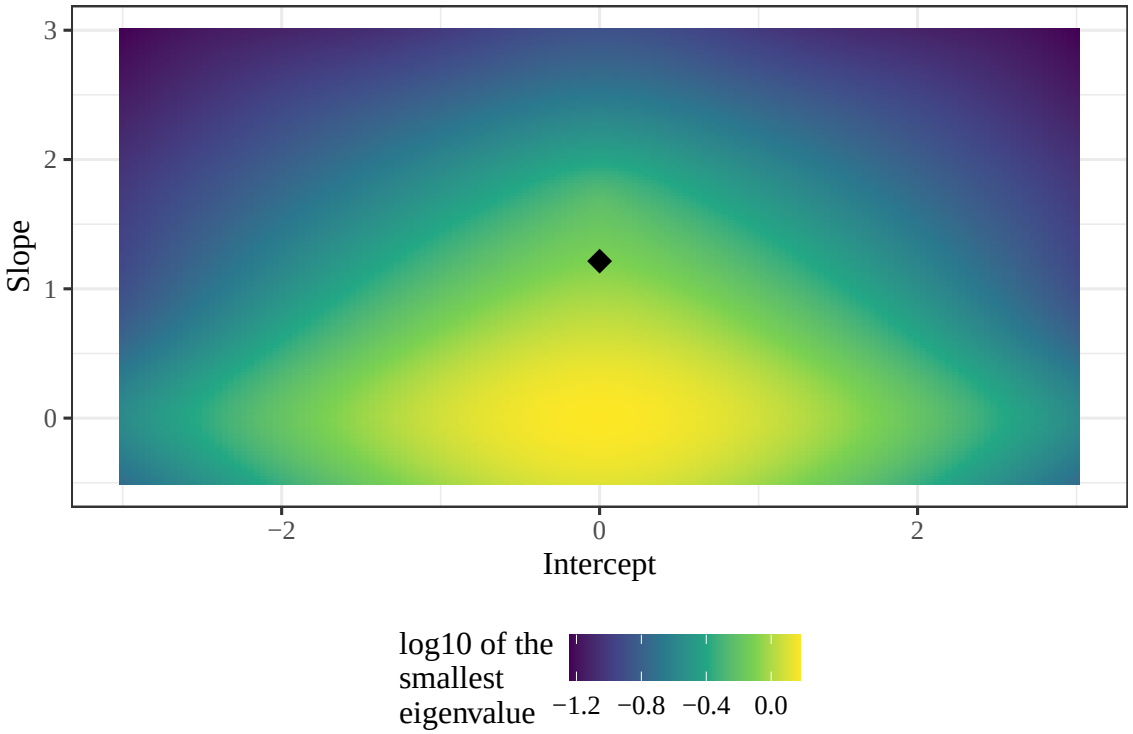
(b)

Figure 15: The Hessian[c] of Example 4.4 at zero coefficients. The slope[d] entry is far larger than the intercept[e] entry because it sums the squared values of the feature.

The Hessian[f] changes with the coefficients. Figure 16 shows its smallest eigenvalue[g], after subtracting 3.5 from every predictor value, over the intercept[h] and the slope[i].

4.0.11 R

```
x_six <- 1:6 - 3.5
y_six <- c(0, 0, 1, 0, 1, 1)
# nolint next: undesirable_function_linter.
design_six <- cbind(1, x_six)
intercepts_six <- seq(-3, 3, length.out = 150)
slopes_six <- seq(-0.5, 3, length.out = 150)
grid_six <- tidyr::expand_grid(intercept = intercepts_six, slope = slopes_six)
# nolint next: undesirable_function_linter.
coef_mat <- rbind(grid_six$intercept, grid_six$slope)
prob_mat <- stats::plogis(design_six %*% coef_mat)
weight_mat <- prob_mat * (1 - prob_mat)
entry_top_left <- colSums(weight_mat)
entry_off_diag <- colSums(weight_mat * x_six)
entry_bottom_right <- colSums(weight_mat * x_six^2)
determinant <- entry_top_left * entry_bottom_right - entry_off_diag^2
half_gap <- (entry_top_left - entry_bottom_right) / 2
eigen_mean <- (entry_top_left + entry_bottom_right) / 2
eigen_max <- eigen_mean + sqrt(half_gap^2 + entry_off_diag^2)
hessian_six <- dplyr::mutate(
  grid_six,
  log_eigen_min = log10(determinant / eigen_max)
)
fit_six <- stats::glm(y_six ~ x_six, family = stats::binomial)
fit_coef <- stats::coef(fit_six)
mle_six <- tibble::tibble(
  intercept = fit_coef[1],
  slope = fit_coef[2]
)
hessian_six |>
  ggplot2::ggplot() +
  ggplot2::aes(intercept, slope, fill = log_eigen_min) +
  ggplot2::geom_raster() +
  ggplot2::geom_point(
    data = mle_six,
    mapping = ggplot2::aes(intercept, slope),
    inherit.aes = FALSE,
    shape = 18,
    size = 4
  ) +
  ggplot2::scale_fill_viridis_c() +
  ggplot2::labs(
    x = "Intercept",
    y = "Slope",
    fill = "log10 of the\nsmallest\neigenvalue"
  )
```

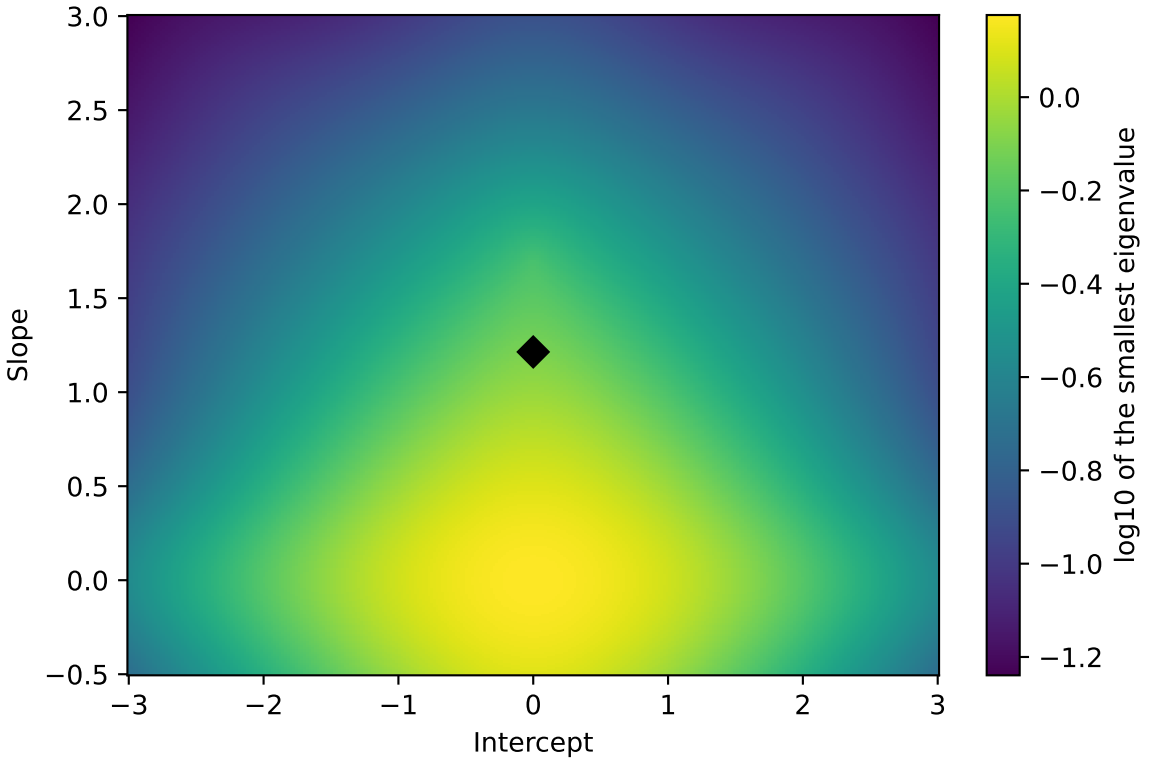

Slope
3
2
1
0
−2
0
2
Intercept
log10 of the smallest eigenvalue
−1.2
−0.8
−0.4
0.0

(a)

4.0.12 Python

```
import matplotlib.pyplot as plt
import numpy as np
import statsmodels.api as sm

x_six = np.arange(1, 7) - 3.5
y_six = np.array([0, 0, 1, 0, 1, 1])
design_six = sm.add_constant(x_six.astype(float))
intercepts = np.linspace(-3, 3, 300)
slopes = np.linspace(-0.5, 3, 300)
grid_intercept, grid_slope = np.meshgrid(intercepts, slopes)
coef_grid = np.stack([grid_intercept.ravel(), grid_slope.ravel()])
prob_grid = 1 / (1 + np.exp(-(design_six @ coef_grid)))
weight_grid = prob_grid * (1 - prob_grid)
entry_top_left = weight_grid.sum(axis=0)
entry_off_diag = (weight_grid * x_six[:, None]).sum(axis=0)
entry_bottom_right = (weight_grid * x_six[:, None] ** 2).sum(axis=0)
determinant = entry_top_left * entry_bottom_right - entry_off_diag**2
half_gap = (entry_top_left - entry_bottom_right) / 2
eigen_max = (entry_top_left + entry_bottom_right) / 2 + np.sqrt(half_gap**2 + entry_off_diag**2)
log_eigen_min = np.log10(determinant / eigen_max).reshape(grid_intercept.shape)
fit_six = sm.GLM(y_six, design_six, family=sm.families.Binomial()).fit()
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
image = ax.pcolormesh(
    grid_intercept, grid_slope, log_eigen_min, cmap="viridis", shading="auto"
)
ax.plot(fit_six.params[0], fit_six.params[1], "kD", markersize=8);
ax.set_xlabel("Intercept");
ax.set_ylabel("Slope");
fig.colorbar(image, ax=ax, label="log10 of the smallest eigenvalue");
plt.show()
```



(b)

Figure 16: The smallest eigenvalue[j] of the Hessian[k] of the total log loss for the six points of Example 4.3, over the intercept[l] and the slope[m]. The maximum likelihood estimate[n] is in black. The smallest eigenvalue is above 0 at every vector of coefficients shown. It is largest at the zero coefficients, where every fitted probability[o] is 1/2, and shrinks as any of them approaches 0 or 1.

Figure 17 draws the same smallest eigenvalue as a surface, so its height shows the curvature.

4.0.13 R

(This 3D surface is interactive; see the HTML or slide version of this page to view it.)

4.0.14 Python

(This 3D surface is interactive; see the HTML or slide version of this page to view it.)

Figure 17: The smallest eigenvalue[p] of the Hessian[q] of the total log loss for the six points of Example 4.3, as a surface over the intercept[r] and the slope[s]. The maximum likelihood estimate[t] is in vermilion.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[b] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope
[c] https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian
[d] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope
[e] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[f] https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian
[g] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-eigenvalue
[h] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[i] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope
[j] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-eigenvalue
[k] https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian
[l] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[m] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope
[n] https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle

[o] https://morrison-lab.github.io/pds/probability-basics.html#def-probability
[p] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-eigenvalue
[q] https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian
[r] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[s] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope
[t] https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle

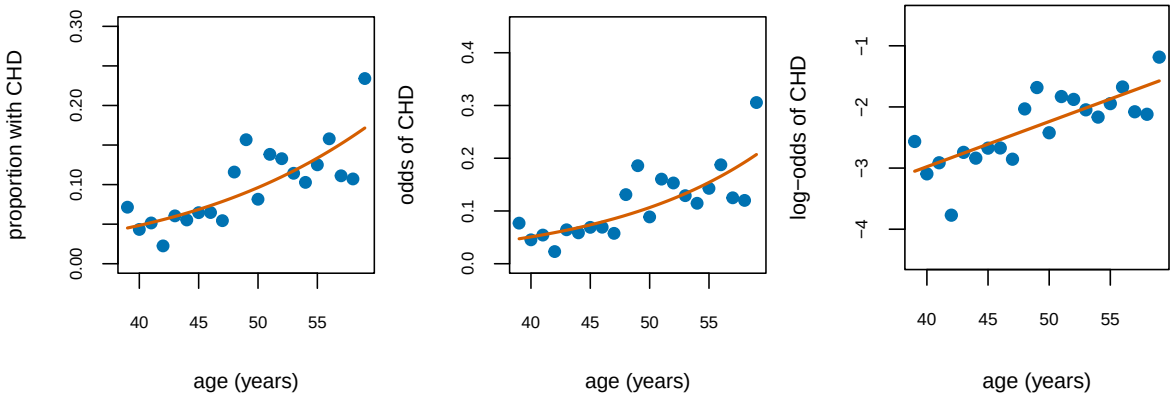
Example 4.5 (The total log loss, its gradient and its Hessian on the WCGS data)**.** Consider a logistic regression model of coronary heart disease (`chd`) in the WCGS data with two features, age and body mass index (BMI). The earlier examples on this page use age alone or BMI alone, and this one uses both. Center each feature at its sample mean[a], 46.28 years and 24.52 kg/m^2, so that $\pi_i = \text{expit}(\beta_0 + \beta_{\text{age}} x_{i,\text{age}} + \beta_{\text{BMI}} x_{i,\text{BMI}})$, where $x_{i,\text{age}}$ and $x_{i,\text{BMI}}$ are man i's centered age and BMI. Centering makes β_0 the log-odds[b] of a man whose age and BMI are both at their sample means. It also keeps the surfaces below readable. They hold the intercept[c] fixed and vary the two slopes[d]. With uncentered ages, a change of 0.1 in β_{age} would move every man's log-odds by about 4.6, and the total log loss C would rise steeply on either side of one narrow valley. The maximum likelihood fit[e] is $\hat{\beta}_0 = -2.514$, $\hat{\beta}_{\text{age}} = 0.0739$ and $\hat{\beta}_{\text{BMI}} = 0.0815$.

Figure 18 starts at that fit. Its top three panels compare the fitted curve at the sample mean BMI with the proportion[f] of men with the disease at each age. The curve rises with age on the probability[g] scale (Figure 18a) and curves upward on the odds[h] scale (Figure 18b), and on the log-odds scale (Figure 18c) it is a straight line with slope β_{age}. Its bottom three panels draw C over the two slopes, the two slope entries of its gradient from Theorem 4.2, and the three distinct slope entries of its Hessian[i] from Theorem 4.3.

- **The total log loss is a bowl** (Figure 18d). The sliders start at the maximum likelihood fit, so the black marker starts at the bottom of the bowl. There both gradient surfaces cross the gray plane at height 0 (Figure 18e): both gradient entries round to 0 at four decimal places.
- **The bowl is steeper along β_{age}.** The slope of each gradient surface along its own coefficient is the matching diagonal Hessian entry, 8672 for age and 1739 for BMI at the fit (Figure 18f). Each entry is a weighted sum[j] of a squared centered feature, and the sample standard deviation[k] of age, in years, is 2.2 times that of BMI, in kg/m^2.
- **The cross entry is small.** At the fit $\frac{\partial^2 C}{\partial \beta_{\text{age}} \partial \beta_{\text{BMI}}} = 241$, because the sample correlation[l] of age and BMI in the WCGS men is only 0.027.

Each surface depends on all three coefficients, so moving the intercept slider changes every surface, while moving a slope slider moves only the black markers.

[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean
[b] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds
[c] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[d] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope
[e] https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle
[f] https://morrison-lab.github.io/sds/classification.html#def-prevalence
[g] https://morrison-lab.github.io/pds/probability-basics.html#def-probability
[h] https://morrison-lab.github.io/pds/probability-basics.html#def-odds
[i] https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian
[j] https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-weighted-sum
[k] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-sd
[l] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-pearson-r



(a) The probability[a] scale

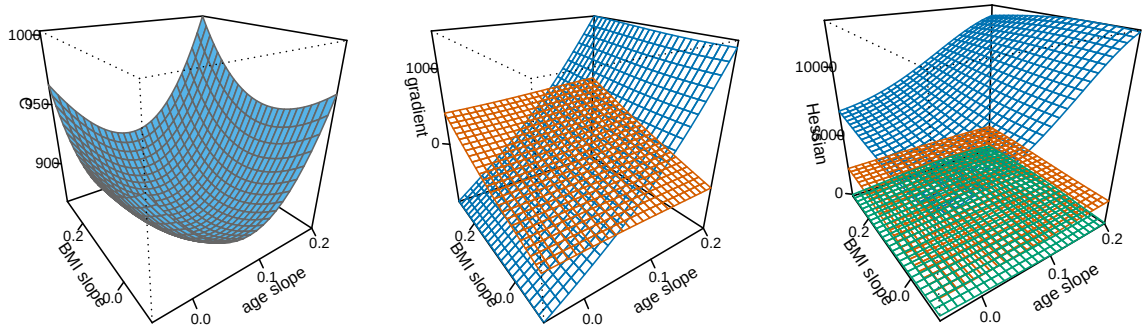
(b) The odds[a] scale

(c) The log-odds[a] scale

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-probability

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-odds

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds



(d) The total log loss

(e) The gradient

(f) The Hessian[a]

[a] https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian

Figure 18: The logistic regression model of Example 4.5, for coronary heart disease with centered age and BMI as features. The top three panels show, at each age, the proportion[b] of men with the disease, as dots, and the fitted curve at the sample mean[c] BMI, in vermilion, on the probability[d], odds[e] and log-odds[f] scales. The bottom three panels show, over the two slopes[g] at a fixed intercept[h], the total log loss C, the two slope entries of its gradient (age in blue, BMI in vermilion), and the three distinct slope entries of its Hessian[i] (age twice in blue, BMI twice in vermilion, one of each in green). All panels show the maximum likelihood fit[j].

[b] https://morrison-lab.github.io/sds/classification.html#def-prevalence
[c] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean
[d] https://morrison-lab.github.io/pds/probability-basics.html#def-probability

[e] https://morrison-lab.github.io/pds/probability-basics.html#def-odds
[f] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds
[g] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope
[h] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[i] https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian
[j] https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle

Theorem 4.4 (The Hessian of the total log loss has bounded eigenvalues)**.** *For every $\vec{\beta}$, every eigenvalue[a] of the Hessian[b] of the total log loss is at most*

$$\tfrac{1}{4}\lambda_{\max}(\mathbf{D}^\top\mathbf{D}).$$

[a] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-eigenvalue
[b] https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian

Proof

Proof. By Theorem 4.3, the Hessian[a] is $\mathbf{D}^\top W\mathbf{D}$, with $W = \operatorname{diag}(\pi_i(1-\pi_i))$. Part 4d of the sigmoid properties exercise showed that every $\pi_i(1-\pi_i) \le 1/4$.
Let $\underset{\sim}{v}$ be a unit eigenvector[b] of $\mathbf{D}^\top W\mathbf{D}$ for the eigenvalue[c] λ. Then $\lambda = \underset{\sim}{v}^\top\mathbf{D}^\top W\mathbf{D}\underset{\sim}{v}$. Write z_i for the ith entry of $\mathbf{D}\underset{\sim}{v}$. Then

$$\lambda = \sum_{i=1}^{n}\pi_i(1-\pi_i)z_i^2 \le \tfrac{1}{4}\sum_{i=1}^{n}z_i^2 = \tfrac{1}{4}\underset{\sim}{v}^\top\mathbf{D}^\top\mathbf{D}\underset{\sim}{v} \le \tfrac{1}{4}\lambda_{\max}(\mathbf{D}^\top\mathbf{D}).$$

The last step holds because $\underset{\sim}{v}^\top\mathbf{D}^\top\mathbf{D}\underset{\sim}{v}$ is at most the largest eigenvalue of $\mathbf{D}^\top\mathbf{D}$. □

[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-eigenvector
[c] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-eigenvalue

Example 4.6 (The eigenvalue bound for the six points)**.**
Take the six points of Example 4.3, and the coefficients $\vec{\beta} = (-3.5, 1)$, which put π_1 and π_6 away from 1/2. The code compares the largest eigenvalue[a] of the Hessian[b] with $\frac{1}{4}\lambda_{\max}(\mathbf{D}^\top\mathbf{D})$.

4.0.15 R

```
# the six points, with the column of ones for the intercept
x_six <- 1:6
design_six <- cbind(1, x_six) # nolint
# the bound: a quarter of the largest eigenvalue of design' design
gram_six <- crossprod(design_six)
gram_eigen <- eigen(gram_six, symmetric = TRUE, only.values = TRUE)
gram_max <- max(gram_eigen$values)
# the Hessian at the coefficients (-3.5, 1)
coef_far <- c(-3.5, 1)
linear_mat <- design_six %*% coef_far
linear_far <- drop(linear_mat)
p_far <- plogis(linear_far)
w_far <- p_far * (1 - p_far)
hessian_far <- crossprod(design_six, w_far * design_six)
hessian_eigen <- eigen(hessian_far, symmetric = TRUE, only.values = TRUE)
hessian_max <- max(hessian_eigen$values)
c(hessian_max = hessian_max, bound = gram_max / 4)
```

```
hessian_max       bound
```

```
   13.59139    23.97629
```

4.0.16 Python

```python
import numpy as np

# the six points, with the column of ones for the intercept
x_six = np.arange(1, 7)
design_six = np.column_stack([np.ones(6), x_six])
# the bound: a quarter of the largest eigenvalue of design' design
gram_six = design_six.T @ design_six
gram_max = np.linalg.eigvalsh(gram_six).max()
# the Hessian at the coefficients (-3.5, 1)
coef_far = np.array([-3.5, 1.0])
linear_far = design_six @ coef_far
p_far = 1 / (1 + np.exp(-linear_far))
w_far = p_far * (1 - p_far)
hessian_far = design_six.T @ (w_far[:, None] * design_six)
hessian_max = np.linalg.eigvalsh(hessian_far).max()
print(hessian_max, gram_max / 4)
```

```
13.591392199129405 23.976292123646264
```

The largest eigenvalue[c] of the Hessian[d] is 13.591, which is below 23.976. At $\vec{\beta} = \vec{0}$ every $\pi_i = 1/2$, so the Hessian is $\frac{1}{4}\mathbf{D}^\top\mathbf{D}$ and the two are equal.

[a] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-eigenvalue
[b] https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian
[c] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-eigenvalue
[d] https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian

Corollary 4.1 (A step size that is safe for every coefficient vector)**.** *Gradient descent*[a] *on the total log loss with the step size*

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

lowers the loss at every step that starts from a $\vec{\beta}$ *with a nonzero gradient.*

[a] https://morrison-lab.github.io/mds/optimization-gradient-descent.html#def-gradient-descent

Proof

Proof. Write C for the total log loss and $L = \frac{1}{4}\lambda_{\max}(\mathbf{D}^\top\mathbf{D})$, so that $\alpha < 2/L$ is the condition $\alpha < 8/\lambda_{\max}(\mathbf{D}^\top\mathbf{D})$. Let $\underset{\sim}{g} = \nabla C(\vec{\beta})$ be nonzero, and take the step $\underset{\sim}{d} = -\alpha\underset{\sim}{g}$.
The Taylor expansion[a] of C to second order, with a remainder, says that for some $\vec{\beta}'$ on the segment from $\vec{\beta}$ to $\vec{\beta} + \underset{\sim}{d}$,

$$C(\vec{\beta} + \underset{\sim}{d}) = C(\vec{\beta}) + \underset{\sim}{g}^\top\underset{\sim}{d} + \tfrac{1}{2}\underset{\sim}{d}^\top\big(\nabla^2 C(\vec{\beta}')\big)\underset{\sim}{d}.$$

By Theorem 4.3 the Hessian[b] is positive semidefinite[c], and by Theorem 4.4 its eigenvalues[d] are at most L, so $\frac{1}{2}\underset{\sim}{d}^\top\big(\nabla^2 C(\vec{\beta}')\big)\underset{\sim}{d} \le \frac{L}{2}\underset{\sim}{d}^\top\underset{\sim}{d}$. Substituting $\underset{\sim}{d} = -\alpha\underset{\sim}{g}$ gives

$$C(\vec{\beta} + \underset{\sim}{d}) \le C(\vec{\beta}) - \alpha\big(1 - \tfrac{L\alpha}{2}\big)\underset{\sim}{g}^\top\underset{\sim}{g}.$$

Because $\alpha < 2/L$, the factor $1 - L\alpha/2$ is greater than zero, and $\underset{\sim}{g}^\top\underset{\sim}{g} > 0$, so C falls. □

[a] https://morrison-lab.github.io/mds/calculus-taylor-series.html#def-taylor-polynomial
[b] https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian

[c] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-positive-semidefinite
[d] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-eigenvalue

Example 4.7 (The safe step size for the six points)**.**
For the six points of Example 4.3, the step-size condition of Corollary 4.1 is $\alpha < 8/\lambda_{\max}(\mathbf{D}^\top \mathbf{D})$.

4.0.17 R

```
# the six points, with the column of ones for the intercept
x_six <- 1:6
design_six <- cbind(1, x_six) # nolint
# the largest eigenvalue of design' design
gram_six <- crossprod(design_six)
gram_eigen <- eigen(gram_six, symmetric = TRUE, only.values = TRUE)
gram_max <- max(gram_eigen$values)
# the safe step-size limit
8 / gram_max
```

```
[1] 0.08341573
```

4.0.18 Python

```
import numpy as np

# the six points, with the column of ones for the intercept
x_six = np.arange(1, 7)
design_six = np.column_stack([np.ones(6), x_six])
# the largest eigenvalue of design' design
gram_max = np.linalg.eigvalsh(design_six.T @ design_six).max()
# the safe step-size limit
print(8 / gram_max)
```

```
0.08341573374590017
```

The largest eigenvalue[a] of $\mathbf{D}^\top \mathbf{D}$ is 95.91, so every α below 0.0834 is safe at every $\vec{\beta}$.

[a] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-eigenvalue

Remark (The safe step size is often too small)

Remark 4.1 (The safe step size is often too small). The condition of Corollary 4.1 holds for every $\vec{\beta}$, so it is often smaller than needed. In practice, a line search chooses step sizes adaptively without computing $\lambda_{\max}(\mathbf{D}^\top \mathbf{D})$.

Theorem 4.5 (The residuals are orthogonal to every column of the design matrix at a minimizer)**.** *At any finite minimizer $\hat{\vec{\beta}}$ of the total log loss, the gradient equals the zero vector:*

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

Each column of $\mathbf{D}$ is orthogonal[a] to the vector of residuals, and the column of ones makes the residuals sum to zero.

[a] https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-orthogonal-vectors

Proof

Proof. The log loss is differentiable[a] and $\hat{\vec{\beta}}$ minimizes it over all coefficient vectors, so its gradient vanishes there. By Theorem 4.2, $\nabla C(\hat{\vec{\beta}}) = \mathbf{D}^\top(\pi - y)$. The jth entry of this vector is the dot product of the jth column of $\mathbf{D}$ with the residuals, and the entry for the column of ones is $\sum_{i=1}^{n}(\pi_i - y_i)$. □

[a] https://morrison-lab.github.io/mds/calculus-derivatives.html#def-differentiable

Example 4.8 (Residuals at the fitted coefficients)**.** Fit a logistic regression with an intercept to the six points of Example 4.3, and compute $\mathbf{D}^\top(\pi - y)$.

4.0.19 R

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

```
(Intercept)       x_six
          0           0
```

4.0.20 Python

```python
import numpy as np
import statsmodels.api as sm

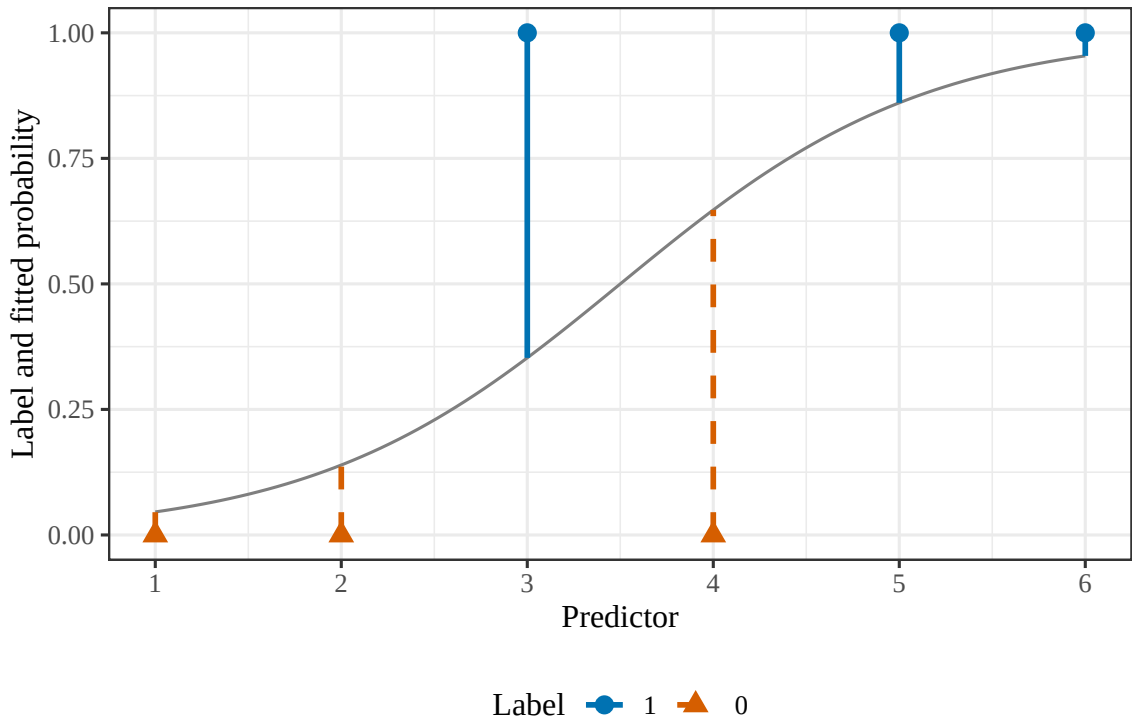
# the six points
x_six = np.arange(1, 7)
y_six = np.array([0, 0, 1, 0, 1, 1])
# the fit, and its design matrix with the column of ones
design_six = sm.add_constant(x_six.astype(float))
fit_six = sm.GLM(y_six, design_six, family=sm.families.Binomial()).fit()
# the residuals: fitted probabilities minus labels
residual_six = fit_six.fittedvalues - y_six
# the gradient, rounded to six decimals
# adding 0.0 turns a negative zero into 0, so the output prints 0. rather than -0.
print((design_six.T @ residual_six).round(6) + 0.0)
```

```
[0. 0.]
```

Both entries are zero to six decimals, as Theorem 4.5 requires.
Figure 19 draws one vertical segment for each of the six points.

4.0.21 R

```
library(ggplot2)
lst_pts <- data.frame(
  x = x_six,
  label = factor(y_six, levels = c(1, 0)),
  y = y_six,
  fitted = fitted(fit_six)
)
lst_curve <- data.frame(x = seq(1, 6, length.out = 100))
lst_curve$prob <- predict(
  fit_six,
  newdata = data.frame(x_six = lst_curve$x),
  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")
```

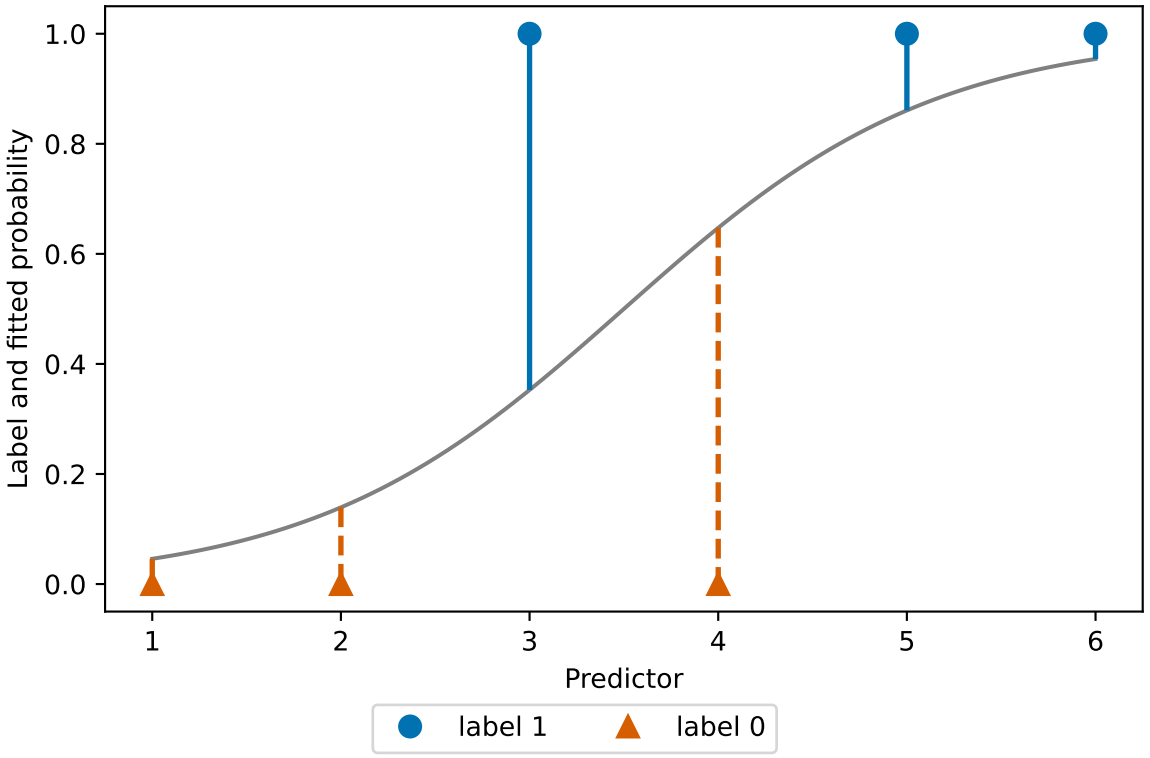


(a)

4.0.22 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,
            linestyle="", label=f"label {label_lst}");
ax.set_xticks(range(1, 7));
ax.set_xlabel("Predictor");
ax.set_ylabel("Label and fitted probability");
fig.legend(loc="outside lower center", ncol=2);
plt.show()
```



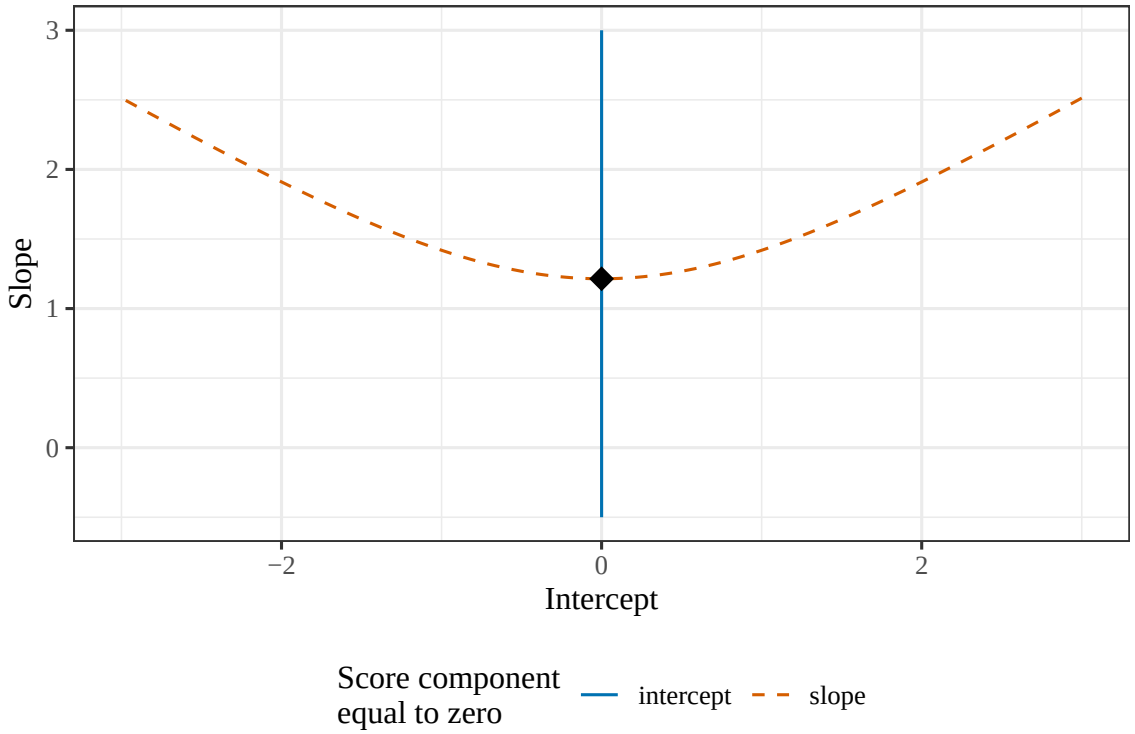
(b)

Figure 19: The six points of Example 4.8 and their fitted probability (curve). Each vertical segment is the residual of one point. The solid segments and the dashed segments have the same total length, because the intercept[a] entry of the gradient is zero.

The score[b] is this gradient with its sign reversed. Figure 20 draws the coefficients where each entry of the score is zero. It first subtracts 3.5 from every predictor value. This leaves the slope[c] unchanged and moves the intercept[d] to 0, the fitted log-odds[e] at the predictor value 3.5. Without it, the two curves nearly coincide and are hard to tell apart. The two curves cross only at the fitted coefficients.

4.0.23 R

```
x_six <- 1:6 - 3.5
y_six <- c(0, 0, 1, 0, 1, 1)
# nolint next: undesirable_function_linter.
design_six <- cbind(1, x_six)
intercepts_six <- seq(-3, 3, length.out = 150)
slopes_six <- seq(-0.5, 3, length.out = 150)
grid_six <- tidyr::expand_grid(intercept = intercepts_six, slope = slopes_six)
# nolint next: undesirable_function_linter.
coef_mat <- rbind(grid_six$intercept, grid_six$slope)
prob_mat <- stats::plogis(design_six %*% coef_mat)
score_mat <- crossprod(design_six, y_six - prob_mat)
score_six <- dplyr::mutate(
  grid_six,
  score_intercept = score_mat[1, ],
  score_slope = score_mat[2, ]
)
fit_six <- stats::glm(y_six ~ x_six, family = stats::binomial)
fit_coef <- stats::coef(fit_six)
mle_six <- tibble::tibble(
  intercept = fit_coef[1],
  slope = fit_coef[2]
)
colours_six <- c("intercept" = "#0072B2", "slope" = "#D55E00")
linetypes_six <- c("intercept" = "solid", "slope" = "dashed")
score_six |>
  ggplot2::ggplot() +
  ggplot2::aes(intercept, slope) +
  ggplot2::geom_contour(
    ggplot2::aes(
      z = score_intercept,
      colour = "intercept",
      linetype = "intercept"
    ),
    breaks = 0 # check-unexplained-choices: allow
  ) +
  ggplot2::geom_contour(
    ggplot2::aes(z = score_slope, colour = "slope", linetype = "slope"),
    breaks = 0 # check-unexplained-choices: allow
  ) +
  ggplot2::geom_point(
    data = mle_six,
    shape = 18,
    size = 4
  ) +
  ggplot2::scale_colour_manual(values = colours_six) +
  ggplot2::scale_linetype_manual(values = linetypes_six) +
  ggplot2::labs(
    x = "Intercept",
    y = "Slope",
    colour = "Score component\nequal to zero",
    linetype = "Score component\nequal to zero"
  )
```

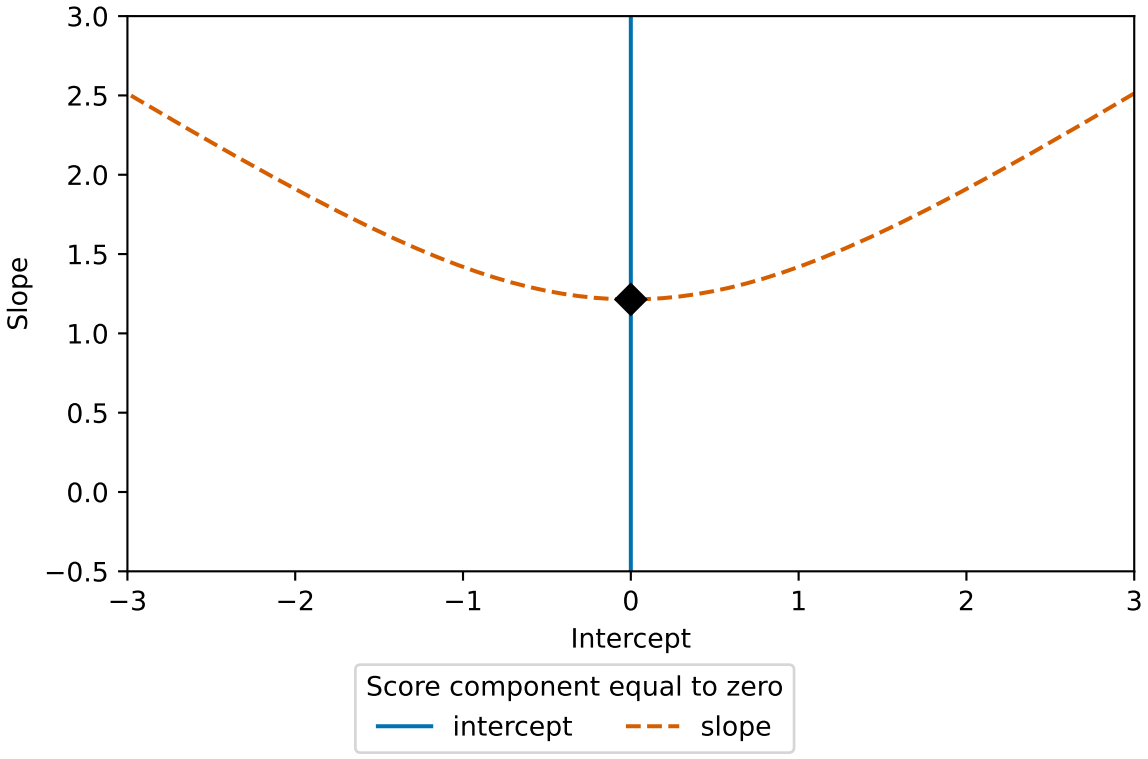

3
2
1
0
Slope
−2
0
2
Intercept
Score component
equal to zero
intercept
slope

(a)

4.0.24 Python

```
import matplotlib.pyplot as plt
import numpy as np
import statsmodels.api as sm
from matplotlib.lines import Line2D

x_six = np.arange(1, 7) - 3.5
y_six = np.array([0, 0, 1, 0, 1, 1])
design_six = sm.add_constant(x_six.astype(float))
intercepts = np.linspace(-3, 3, 300)
slopes = np.linspace(-0.5, 3, 300)
grid_intercept, grid_slope = np.meshgrid(intercepts, slopes)
coef_grid = np.stack([grid_intercept.ravel(), grid_slope.ravel()])
prob_grid = 1 / (1 + np.exp(-(design_six @ coef_grid)))
score_grid = design_six.T @ (y_six[:, None] - prob_grid)
fit_six = sm.GLM(y_six, design_six, family=sm.families.Binomial()).fit()
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
for row, colour, style in [(0, "#0072B2", "-"), (1, "#D55E00", "--")]:
    ax.contour(
        grid_intercept,
        grid_slope,
        score_grid[row].reshape(grid_intercept.shape),
        levels=[0],
        colors=colour,
        linestyles=style,
    );
ax.plot(fit_six.params[0], fit_six.params[1], "kD", markersize=8);
ax.set_xlabel("Intercept");
ax.set_ylabel("Slope");
handles = [
    Line2D([], [], color="#0072B2", linestyle="-", label="intercept"),
    Line2D([], [], color="#D55E00", linestyle="--", label="slope"),
]
fig.legend(
    handles=handles,
    title="Score component equal to zero",
    loc="outside lower center",
    ncol=2,
);
plt.show()
```



(b)

Figure 20: The score[f] of the log-likelihood[g] for the six points of Example 4.3, over the intercept[h] and the slope[i]. Each curve marks the coefficients where one entry of the score is zero. The maximum likelihood estimate[j], in black, is the one point where both entries are zero.

Figure 21 draws both entries of the score as surfaces, so the curves of Figure 20 are where the surfaces cross height 0.

4.0.25 R

(This 3D surface is interactive; see the HTML or slide version of this page to view it.)

4.0.26 Python

(This 3D surface is interactive; see the HTML or slide version of this page to view it.)

Figure 21: The score[k] of the log-likelihood[l] for the six points of Example 4.3, as surfaces over the intercept[m] and the slope[n]. The maximum likelihood estimate[o], in black, is the one point where both surfaces are at height 0.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[b] https://morrison-lab.github.io/sds/intro-MLEs.html#def-score
[c] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope
[d] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[e] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds
[f] https://morrison-lab.github.io/sds/intro-MLEs.html#def-score
[g] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik
[h] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[i] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope
[j] https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle
[k] https://morrison-lab.github.io/sds/intro-MLEs.html#def-score
[l] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik
[m] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[n] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle

Exercise 4.11 (Fitting by gradient descent, in code)**.** Simulate $n = 400$ observations with two standard normal features x_1, x_2 and labels drawn from $\Pr[Y = 1 \mid \underset{\sim}{x}] = \text{expit}(-0.5 + 2x_1 - 1.5x_2)$, so the coefficients that generated the data are $\vec{\beta} = (-0.5, 2, -1.5)$.
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 4.2. Explain how the sign and scale of the gradient of J relate to the gradient $\mathbf{D}(\mathbf{x})^\top(\underset{\sim}{y} - \underset{\sim}{\pi})$ of the log-likelihood[c] of Exercise 1.3.

[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean
[b] https://morrison-lab.github.io/mds/optimization-gradient-descent.html#def-gradient-descent
[c] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik

Solution

Solution 4.11. The gradient of J is $\mathbf{D}(\mathbf{x})^\top(\underset{\sim}{\pi} - y)/n$. It is the log-likelihood[a]'s gradient $\mathbf{D}(\mathbf{x})^\top(y - \underset{\sim}{\pi})$ with its sign reversed, because J is the negative log-likelihood, and divided by n, because J is a mean[b]. Dividing by n does not change the minimizer (Exercise 1.21), and it makes a good step size depend less on how many observations there are.

4.0.27 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
```

4.0.28 R

```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
```

```r
round(J(beta), 4)
```

```
[1] 0.4201
```

4.0.29 Julia

```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]
```

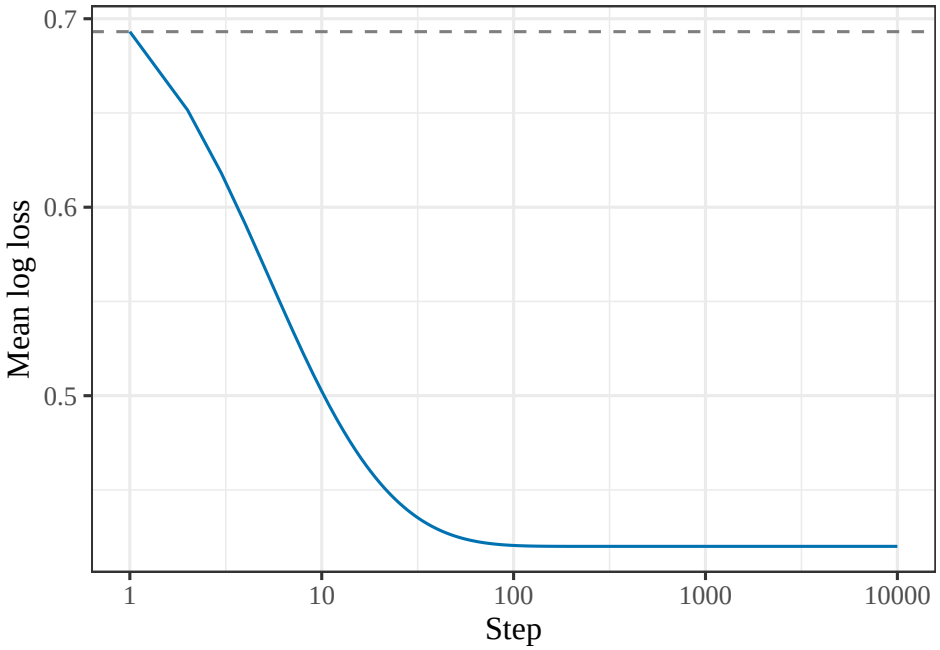
```julia
println(round(J(beta), digits = 4))
```

```
0.4069
```

Figure 22 plots J at each step, for the simulated data of this solution.

4.0.30 R

```
set.seed(0)
gdl_n <- 400
gdl_x1 <- rnorm(gdl_n)
gdl_x2 <- rnorm(gdl_n)
gdl_ones <- rep(1, gdl_n)
gdl_cols <- c(gdl_ones, gdl_x1, gdl_x2)
gdl_x <- matrix(gdl_cols, ncol = 3)
gdl_score <- -0.5 + 2 * gdl_x1 - 1.5 * gdl_x2
gdl_u <- runif(gdl_n)
gdl_y <- as.numeric(gdl_u < plogis(gdl_score))
gdl_beta <- c(0, 0, 0)
gdl_loss <- numeric(10000)
for (gdl_s in seq_along(gdl_loss)) {
  gdl_eta <- drop(gdl_x %*% gdl_beta)
  gdl_p <- plogis(gdl_eta)
  gdl_terms <- gdl_y * log(gdl_p) + (1 - gdl_y) * log(1 - gdl_p)
  gdl_loss[gdl_s] <- -mean(gdl_terms)
  gdl_resid <- gdl_p - gdl_y
  gdl_cross <- crossprod(gdl_x, gdl_resid)
  gdl_grad <- drop(gdl_cross) / gdl_n
  gdl_beta <- gdl_beta - 0.5 * gdl_grad
}
gdl_step <- seq_along(gdl_loss)
gdl_curve <- tibble::tibble(step = gdl_step, loss = gdl_loss)
gdl_start <- log(2)
gdl_curve |>
  ggplot2::ggplot(ggplot2::aes(step, loss)) +
  ggplot2::geom_hline(
    yintercept = gdl_start,
    linetype = "dashed",
    colour = "grey50"
  ) +
  ggplot2::geom_line(colour = "#0072B2") +
  ggplot2::scale_x_log10() +
  ggplot2::labs(x = "Step", y = "Mean log loss")
```



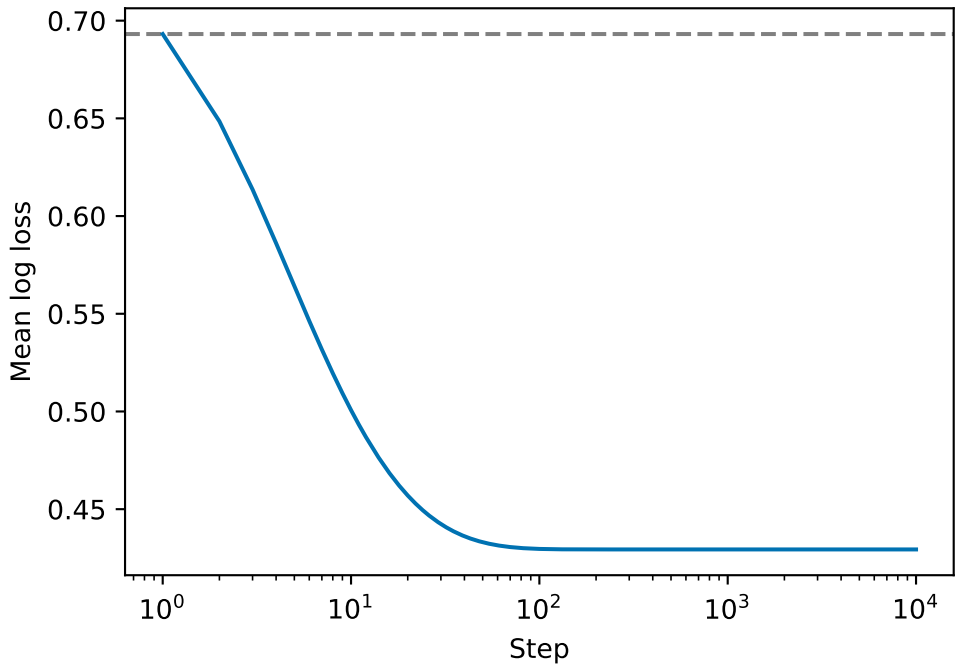
(a)

4.0.31 Python

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

rng_gdl = np.random.default_rng(0)
n_gdl = 400
x1_gdl = rng_gdl.normal(size=n_gdl)
x2_gdl = rng_gdl.normal(size=n_gdl)
x_gdl = np.column_stack([np.ones(n_gdl), x1_gdl, x2_gdl])
score_gdl = -0.5 + 2 * x1_gdl - 1.5 * x2_gdl
y_gdl = (rng_gdl.random(n_gdl) < expit(score_gdl)).astype(float)
beta_gdl = np.zeros(3)
loss_gdl = np.zeros(10000)
for s in range(10000):
    p_gdl = expit(x_gdl @ beta_gdl)
    terms_gdl = y_gdl * np.log(p_gdl) + (1 - y_gdl) * np.log(1 - p_gdl)
    loss_gdl[s] = -np.mean(terms_gdl)
    beta_gdl = beta_gdl - 0.5 * x_gdl.T @ (p_gdl - y_gdl) / n_gdl

fig, ax = plt.subplots(figsize=(5, 3.5), layout="constrained")
ax.axhline(np.log(2), color="grey", linestyle="--");
ax.plot(np.arange(1, 10001), loss_gdl, color="#0072B2");
ax.set_xscale("log");
ax.set_xlabel("Step");
ax.set_ylabel("Mean log loss");
plt.show()
```



(b)

Figure 22: The mean[c] log loss falls quickly at first and then stays at its minimum[d], as gradient descent[e] converges[f].

[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/sds/exploratory-descriptive.html#def-sample-mean
[d] https://morrison-lab.github.io/mds/algebra-basics.html#def-minimum
[e] https://morrison-lab.github.io/mds/optimization-gradient-descent.html#def-gradient-descent
[f] https://morrison-lab.github.io/mds/optimization-newton.html#def-vector-sequence-limit

Exercise 4.12 (Fitting by gradient descent, in code (part 2))**.** Use the setup of Exercise 4.11. Compare the fitted coefficients with $(-0.5, 2, -1.5)$.

Solution

Solution 4.12.

```
# the Python estimate, the size of its gradient and 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
fmt <- function(v, d = 2) formatC(v, format = "e", digits = d)
```

The Python version gives $(-0.54, 1.95, -1.18)$, against the $(-0.5, 2, -1.5)$ from which the data were generated.

Exercise 4.13 (Fitting by gradient descent, in code (part 2c))**.** Use the setup of Exercise 4.11. Explain how to check whether the optimizer stopped before converging.

Solution

Solution 4.13. This continues Solution 4.12.
To check, evaluate the gradient at the final iterate. At a converged iterate it is close to $\vec{0}$.

Exercise 4.14 (Fitting by gradient descent, in code (part 2b))**.** Use the setup of Exercise 4.11. If one of the coefficients is farther from the true value than you anticipated, determine whether the optimizer stopped before converging or whether the difference has another source.

Solution

Solution 4.14. This continues Solution 4.12.
By the check of Solution 4.13, the largest entry of the gradient at the final iterate of Solution 4.12 is 2.22e-16, so the iteration of Exercise 4.11 has converged. The third coefficient, -1.18, *is* the maximum likelihood estimate[a] for this sample[b] of 400 points.
The standard error[c] of that coefficient is 0.164, so -1.18 is 1.96 standard errors[d] from -1.5. If the estimate were approximately normal about -1.5 with that standard error, an estimate at least that far from -1.5 would occur with probability about 0.05, so the difference is large but can arise from sampling alone. The R tab draws an independent[e] sample of 400 observations from R's random number generator, and its estimate[f] of the same coefficient is -1.46. The difference between the Python estimate and -1.5 comes from the random sample, not from the optimizer.

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle
[b] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample
[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 4.15 (Fitting by gradient descent, in code (part 3))**.** Use the setup of Exercise 4.11. The code computes the logistic function[a] as $\frac{1}{2}(1+\tanh(\eta/2))$ rather than $1/(1+\mathrm{e}^{-\eta})$. Part 5 of the sigmoid properties exercise shows that these two expressions[b] are the same mathematical function[c]. State whether they are the same floating-point computation.

[a] https://morrison-lab.github.io/mcs/functions.html#def-logistic-function
[b] https://morrison-lab.github.io/mcs/expressions-algorithms.html#def-expression
[c] https://morrison-lab.github.io/mcs/functions.html#def-function

Solution

Solution 4.15. They are not the same floating-point computation. One form applies `tanh` and then an addition, and the other applies the exponential function and then a division, and each operation rounds its result to the nearest floating-point number, so the two forms can return different values for the same η.

Exercise 4.16 (Fitting by gradient descent, in code (part 4))**.** Use the setup of Exercise 4.11. Compare the numerical behavior of the two forms of Exercise 4.15 for large negative η.

Solution

Solution 4.16.

```
# the two forms of the logistic function 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)
# the largest absolute log-odds in the R simulation, at the fitted coefficients
max_score <- max(abs(drop(X %*% beta)))
```

For negative η, the relative error of the `tanh` form grows much sooner than that of the exponential form.
The exponential form evaluates $\mathrm{e}^{-\eta}$, which overflows to $+\infty$ below about $\eta=-709$, and dividing 1 by $+\infty$ returns 0. The `tanh` form never overflows, but its relative error grows for negative η. For very negative η, $\tanh(\eta/2)$ is close to -1, where neighboring floating-point numbers are about 10^{-16} apart, so $1+\tanh(\eta/2)$ is a multiple of that spacing. The smallest positive value that the `tanh` form returns is 5.55e-17, and for $\eta \le -38.13$ it returns 0. The exponential form evaluates to 9.9e-305 at $\eta=-700$. The log loss evaluates $\log\pi$, so a probability rounded to 0 gives the loss $-\log 0=\infty$, although the exact value of $-\log\pi$ at $\eta=-700$ is about 700. That rounding does not occur in the R simulation, where the largest $|\eta|$ at the fitted coefficients is 6.97, but the two forms still differ. In *absolute* terms the two forms differ by at most 2.22e-16 over $|\eta|\le 10$. In *relative* terms, which set how many correct

digits $\log \pi$ has, the difference grows as π approaches 0. The largest relative difference over $|\eta| \leq 10$ is 5.81e-13, at $\eta = -9.987$, where π is 4.60e-05, and the two floating-point values differ by 3941 units in the last place. The `tanh` form loses these significant digits when it adds 1 to $\tanh(\eta/2)$, which is close to -1, but it keeps more than enough digits over $|\eta| \leq 10$.
For very negative log-odds[a], neither form of the probability should be computed at all. Evaluating the log-likelihood[b] directly from η, for example $\log \pi = \eta - \log(1 + e^{\eta})$ for $\eta < 0$, avoids computing π.

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds
[b] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik

Exercise 4.17 (Fitting by gradient descent, in code (part 5))**.** Use the setup of Exercise 4.11. Name the library function that you would use instead in each language.

Solution

Solution 4.17. In practice you would call a library function:

- in R, `glm(y ~ x1 + x2, family = binomial)`;
- in Python, `LogisticRegression` from scikit-learn, or `sm.GLM(y, X, family=sm.families.Binomial())` from statsmodels;
- in Julia, `glm(@formula(y ~ x1 + x2), data, Binomial(), LogitLink())` from GLM.jl.

Exercise 4.18 (Fitting by gradient descent, in code (part 5b))**.** Use the setup of Exercise 4.11. Check whether each function named in Exercise 4.17 fits the same objective by default.

Solution

Solution 4.18. This continues Solution 4.17.
Of the functions of Solution 4.17, R's `glm`, statsmodels' `GLM` and GLM.jl maximize the unpenalized likelihood[a], so they give the estimates[b] that gradient descent[c] converged to. scikit-learn's `LogisticRegression` [10] adds an ℓ_2 penalty by default, with `C=1.0`, so it fits a regularized problem like that of Exercise 4.9, and it gives different estimates until the penalty is turned off. In scikit-learn 1.9 (checked 2026-10-28), `C=np.inf` turns the penalty off; the older argument `penalty=None` is deprecated.

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik
[b] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[c] https://morrison-lab.github.io/mds/optimization-gradient-descent.html#def-gradient-descent

5 A fit small enough to do by hand

For a dataset of six patients in two groups, the maximum likelihood[25] estimate[26] can be found by hand. Its value found by hand is a benchmark for the step sizes, Newton's method[27], standard errors[28], and likelihood ratio test[29] in the exercises that follow.

Exercise 5.1 (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.

[25] https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle
[26] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[27] https://morrison-lab.github.io/mds/optimization-newton.html#def-newton-method
[28] https://morrison-lab.github.io/sds/estimation.html#def-SE
[29] https://morrison-lab.github.io/sds/intro-MLEs.html#def-lrt-stat

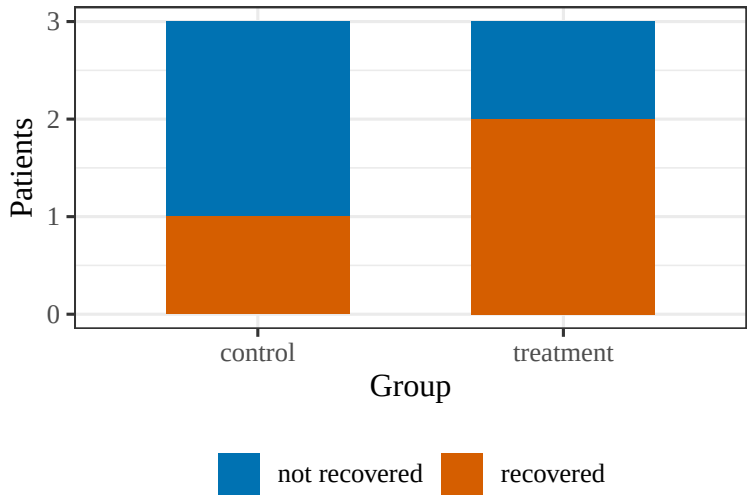
Table 1: The number of patients and recoveries in each group.

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

Model the probability of recovery as $\Pr[Y = 1 \mid x] = \text{expit}(\beta_0 + \beta_1 x)$.
Give the design matrix $\mathbf{D}(\mathbf{x})$.
Figure 23 plots the counts.

5.0.1 R

```
wf_groups <- c("control", "treatment")
wf_statuses <- c("not recovered", "recovered")
wf_group <- rep(wf_groups, each = 2)
wf_status <- rep(wf_statuses, times = 2)
wf_count <- c(2, 1, 1, 2)
wf_fig <- tibble::tibble(group = wf_group, status = wf_status, n = wf_count)
wf_fills <- c("not recovered" = "#0072B2", recovered = "#D55E00")
wf_fig |>
  ggplot2::ggplot(ggplot2::aes(group, n, fill = status)) +
  ggplot2::geom_col(width = 0.6) +
  ggplot2::scale_fill_manual(values = wf_fills) +
  ggplot2::labs(x = "Group", y = "Patients", fill = NULL)
```

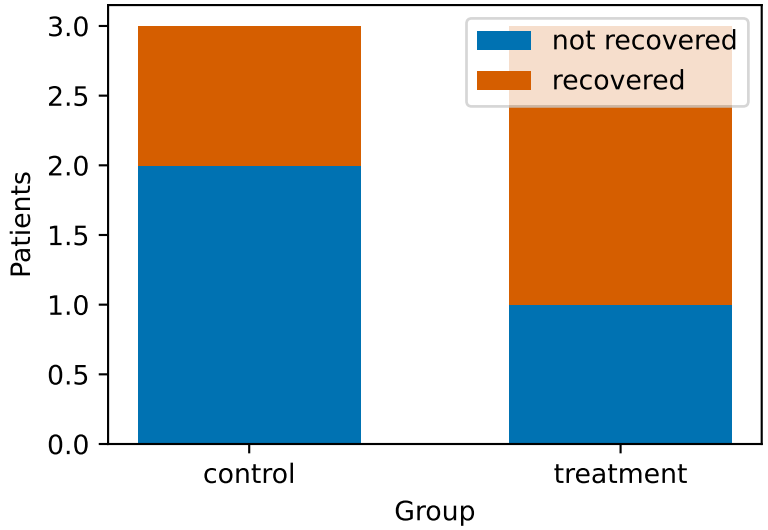


(a)

5.0.2 Python

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(4, 2.8), layout="constrained")
groups_wf = ["control", "treatment"]
ax.bar(groups_wf, [2, 1], color="#0072B2", width=0.6, label="not recovered");
ax.bar(groups_wf, [1, 2], bottom=[2, 1], color="#D55E00", width=0.6,
       label="recovered");
ax.set_xlabel("Group");
ax.set_ylabel("Patients");
ax.legend();
plt.show()
```



(b)

Figure 23: The six patients by group and recovery. Each group has three patients. One recovers in the control group, and two recover in the treatment group.

Solution

Solution 5.1. Order the patients control-first, and let the three control outcomes be $y = 0, 0, 1$ and the three treatment outcomes $y = 0, 1, 1$:

$$\mathbf{D}(\mathbf{x}) = \begin{bmatrix} 1 & -1 \\ 1 & -1 \\ 1 & -1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}, \qquad \underset{\sim}{y} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Exercise 5.2 (A logistic regression you can fit by hand (part 1b))**.** Use the setup of Exercise 5.1. Evaluate $\nabla_{\vec{\beta}} \ell = \mathbf{D}(\mathbf{x})^\top (\underset{\sim}{y} - \underset{\sim}{\pi})$ at $\vec{\beta} = \vec{0}$.

Solution

Solution 5.2. This continues Solution 5.1.
At $\vec{\beta} = \vec{0}$, every $\pi_i = \text{expit}(0)$, which is $1/2$, so $\underset{\sim}{y} - \underset{\sim}{\pi} = \left(-\frac{1}{2}, -\frac{1}{2}, \frac{1}{2}, -\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right)^\top$.
The intercept[a] component of $\mathbf{D}(\mathbf{x})^\top(\underset{\sim}{y} - \underset{\sim}{\pi})$ is the sum of the entries of $\underset{\sim}{y} - \underset{\sim}{\pi}$:

$$\begin{aligned}\sum_{i=1}^{n}(y_i - \pi_i) &= \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} \pi_i && \text{(split the sum)}\\ &= 3 - 6 \times \tfrac{1}{2} && \text{(three recoveries, six probabilities of } \tfrac{1}{2})\\ &= 0.\end{aligned}$$

The slope[b] component multiplies each entry by x_i before summing:

$$\begin{aligned}\sum_{i=1}^{n} x_i(y_i - \pi_i) &= -\left(-\tfrac{1}{2} - \tfrac{1}{2} + \tfrac{1}{2}\right) + \left(-\tfrac{1}{2} + \tfrac{1}{2} + \tfrac{1}{2}\right) && \text{(control rows, then treatment rows)}\\ &= -\left(-\tfrac{1}{2}\right) + \tfrac{1}{2} && \text{(add within each group)}\\ &= 1.\end{aligned}$$

[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 5.3 (A logistic regression you can fit by hand (part 1c))**.** Use the setup of Exercise 5.1. Interpret each of the two components of the gradient of Exercise 5.2.

Solution

Solution 5.3. This continues Solution 5.1 and Solution 5.2.
Three of the six patients recovered, and the probabilities at $\vec{\beta} = \vec{0}$ also sum to three, so the starting point is already calibrated in aggregate, in the sense of Solution 4.8.
The slope component is positive: the group coded $+1$ has more recoveries than the probabilities at $\vec{\beta} = \vec{0}$ predict, and the group coded -1 has fewer. Increasing β_1 from 0 therefore increases the log-likelihood[a] ℓ.

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik

Exercise 5.4 (A logistic regression you can fit by hand (part 2))**.** Use the setup of Exercise 5.1. Give the total 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 5.4. Every $\pi_i = 1/2$, so every observation contributes $-\log\frac{1}{2}$, which equals $\log 2$, whatever its label. The total log loss at $\vec{\beta} = \vec{0}$ is therefore

$$-\ell(\vec{0}) = 6 \log 2,$$

which is 4.159 to four significant figures.
The value does not depend on the data. At $\vec{\beta} = \vec{0}$ the model predicts $1/2$ for every observation, so it scores $\log 2$ per observation whatever the outcomes are, and the total for n observations is $n \log 2$. That makes $n \log 2$ a reference point: the coefficients $\vec{\beta} = \vec{0}$ attain it without using the data, and the fitted model's improvement can be measured against it.

Exercise 5.5 (A logistic regression you can fit by hand (part 3))**.** Use the setup of Exercise 5.1. Use the symmetry of the design and the intercept[a] score equation to show that $\hat{\beta}_0 = 0$.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept

Solution

Solution 5.5. In the trial of Exercise 5.1, the control and treatment groups have the same size, and their codes $x = \pm 1$ are negatives of each other. Let $\pi_+ = \text{expit}(\beta_0 + \beta_1)$ be the probability for the treatment group and $\pi_- = \text{expit}(\beta_0 - \beta_1)$ the probability for the control group.
The intercept[a] score equation $\sum_i \pi_i = \sum_i y_i$ is

$$\begin{aligned} 3\pi_- + 3\pi_+ &= 1 + 2 && \text{(three patients per group; one and two recoveries)} \\ 3\pi_- + 3\pi_+ &= 3 && \text{(add)} \\ \pi_- + \pi_+ &= 1 && \text{(divide by 3).} \end{aligned}$$

By part 2 of the sigmoid properties exercise, $1 - \pi_+ = \text{expit}(-\beta_0 - \beta_1)$, so the last equation is equivalent to $\text{expit}(\beta_0 - \beta_1) = \text{expit}(-\beta_0 - \beta_1)$. The logistic function[b] is strictly increasing, so its arguments are equal:

$$\begin{aligned} \beta_0 - \beta_1 &= -\beta_0 - \beta_1 && \text{(equal arguments)} \\ 2\beta_0 - \beta_1 &= -\beta_1 && \text{(add } \beta_0\text{)} \\ 2\beta_0 &= 0 && \text{(add } \beta_1\text{)} \\ \beta_0 &= 0 && \text{(divide by 2).} \end{aligned}$$

So $\hat{\beta}_0 = 0$, and $\pi_- = 1 - \pi_+$.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[b] https://morrison-lab.github.io/mcs/functions.html#def-logistic-function

Theorem 5.1 (The intercept of a balanced two-group fit)**.** *Let two groups of n patients each be coded $x = -1$ and $x = +1$, with n recoveries altogether. Fit $\Pr(Y = 1 \mid x) = \text{expit}(\beta_0 + \beta_1 x)$ by maximum likelihood[a], and write $\pi_- = \text{expit}(\beta_0 - \beta_1)$ and $\pi_+ = \text{expit}(\beta_0 + \beta_1)$ for the fitted probabilities[b] of the two groups. Every solution of the score equations[c] has $\hat{\beta}_0 = 0$, and then $\pi_- = 1 - \pi_+$.*

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle
[b] https://morrison-lab.github.io/sds/classification.html#def-predicted-probability
[c] https://morrison-lab.github.io/sds/intro-MLEs.html#def-score-equation

Proof

Proof. The intercept[a] score equation[b] $\sum_i \pi_i = \sum_i y_i$ reads $n\pi_- + n\pi_+ = n$, so

$$\text{expit}(\beta_0 - \beta_1) = \pi_- = 1 - \pi_+ = \text{expit}(-\beta_0 - \beta_1),$$

by the symmetry of the logistic function. The logistic function[c] is strictly increasing, so $\beta_0 - \beta_1 = -\beta_0 - \beta_1$, which gives $\beta_0 = 0$. □

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[b] https://morrison-lab.github.io/sds/intro-MLEs.html#def-score-equation
[c] https://morrison-lab.github.io/mcs/functions.html#def-logistic-function

Exercise 5.6 (A logistic regression you can fit by hand (part 3b))**.** Use the setup of Exercise 5.1. Use the score equation for β_1 and the result of Exercise 5.5 to find $\hat{\beta}_1$ exactly.

Solution

Solution 5.6. This continues Solution 5.5.
With $\widehat{\beta}_0 = 0$ and $\pi_- = 1 - \pi_+$ from Solution 5.5, the slope[a] score equation $\sum_{i=1}^{n} x_i(y_i - \pi_i) = 0$ is

$$\begin{aligned}
-(1 - 3\pi_-) + (2 - 3\pi_+) &= 0 && \text{(control rows have } x = -1\text{, treatment rows } x = 1\text{)} \\
-1 + 3\pi_- + 2 - 3\pi_+ &= 0 && \text{(remove the parentheses)} \\
1 + 3\pi_- - 3\pi_+ &= 0 && (-1 + 2 = 1) \\
1 + 3(1 - \pi_+) - 3\pi_+ &= 0 && (\pi_- = 1 - \pi_+) \\
1 + 3 - 3\pi_+ - 3\pi_+ &= 0 && \text{(distribute 3)} \\
4 - 6\pi_+ &= 0 && \text{(collect terms)} \\
\pi_+ &= \tfrac{2}{3} && \text{(solve for } \pi_+\text{)}.
\end{aligned}$$

Then $\pi_- = \frac{1}{3}$. Because $\widehat{\beta}_0 = 0$, $\pi_+ = \operatorname{expit}(\widehat{\beta}_1)$, and

$$\begin{aligned}
\widehat{\beta}_1 &= \operatorname{logit}\left(\tfrac{2}{3}\right) && \text{(invert expit)} \\
&= \log \frac{2/3}{1/3} && \text{(definition of logit)} \\
&= \log 2 && \text{(simplify the fraction)},
\end{aligned}$$

which is 0.6931.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope

Exercise 5.7 (A logistic regression you can fit by hand (part 4))**.** Use the setup of Exercise 5.1. Give the fitted probabilities.

Solution

Solution 5.7. The fitted probability vector is $\widehat{\underset{\sim}{\pi}} = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{2}{3}, \frac{2}{3}, \frac{2}{3}\right)$.

Exercise 5.8 (A logistic regression you can fit by hand (part 4b))**.** Use the setup of Exercise 5.1. Verify both score equations[a] at the fitted probabilities of Exercise 5.7.

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-score-equation

Solution

Solution 5.8. This continues Solution 5.7.
At the fitted probabilities of Solution 5.7, for the intercept[a] equation,

$$\begin{aligned}
\sum_{i=1}^{n} \pi_i &= 3 \times \tfrac{1}{3} + 3 \times \tfrac{2}{3} \\
&= 1 + 2 \\
&= 3,
\end{aligned}$$

which equals $\sum_{i=1}^{n} y_i$.
For the slope[b] equation,

$$\begin{aligned}\sum_{i=1}^{n} x_i \pi_i &= -3 \times \tfrac{1}{3} + 3 \times \tfrac{2}{3} \\ &= -1 + 2 \\ &= 1,\end{aligned}$$

and $\sum_{i=1}^{n} x_i y_i = -1 + 2$, which is also 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 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[c] make the fitted probabilities sum to the observed count in each group. The model has two parameters and the data have two groups, so the model can match both group proportions exactly, and it does.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[b] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope
[c] https://morrison-lab.github.io/sds/intro-MLEs.html#def-score-equation

Exercise 5.9 (A logistic regression you can fit by hand (part 5))**.** Use the setup of Exercise 5.1. 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.

[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 5.9. The slope[a] β_1 is the change in log-odds[b] per *one-unit* increase in x, and under the coding $x \in \{-1, +1\}$ the two groups are two units apart. The treatment-versus-control odds ratio[c] is therefore

$$\begin{aligned}e^{2\hat{\beta}_1} &= e^{2 \log 2} && (\hat{\beta}_1 = \log 2) \\ &= (e^{\log 2})^2 && (\text{power of a power}) \\ &= 2^2 && (e^{\log a} = a) \\ &= 4,\end{aligned}$$

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 $x = +1$.

[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

Exercise 5.10 (A logistic regression you can fit by hand (part 5b))**.** Use the setup of Exercise 5.1. State what $\hat{\beta}_1$ would have been under the coding $x \in \{0, 1\}$.

Solution

Solution 5.10. This continues Solution 5.9.
Under the indicator coding $x \in \{0, 1\}$, the two groups are one unit apart, so $\hat{\beta}_1 = \log 4$, which is 1.386, and $e^{\hat{\beta}_1} = 4$ is the between-group odds ratio itself. The intercept[a] changes to $\text{logit}(1/3) = -\log 2$, so that the fitted probability for the control group ($x = 0$) is still $1/3$.
The fitted probabilities are the same under this coding and under the ± 1 coding of Solution 5.9; only the parameterization differs. Coding a two-level predictor as a $\{0, 1\}$ indicator makes $e^{\hat{\beta}_1}$ equal to the between-group odds ratio.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept

Exercise 5.11 (A logistic regression you can fit by hand (part 6))**.** Use the setup of Exercise 5.1. Give the total log loss at $\hat{\vec{\beta}}$.

Solution

Solution 5.11. Four observations have outcomes equal to the majority outcome in their group, the two treated patients who recovered and the two control patients who did not, and each contributes $-\log\frac{2}{3}$, which equals $\log\frac{3}{2}$. The other two observations each contribute $-\log\frac{1}{3}$, which equals $\log 3$:

$$\begin{aligned} -\ell\left(\hat{\vec{\beta}}\right) &= 4\log\tfrac{3}{2} + 2\log 3 && \text{(sum the contributions)} \\ &= 4\log 3 - 4\log 2 + 2\log 3 && \text{(log of a quotient)} \\ &= 6\log 3 - 4\log 2 && \text{(collect the } \log 3 \text{ terms)}, \end{aligned}$$

which is 3.819.

Exercise 5.12 (A logistic regression you can fit by hand (part 6b))**.** Use the setup of Exercise 5.1. Use the drop in total log loss from Exercise 5.4 (at $\vec{\beta} = \vec{0}$) to Exercise 5.11 (at $\hat{\vec{\beta}}$) to test $H_0 : \beta_1 = 0$ with a likelihood ratio test[a].

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-lrt-stat

Solution

Solution 5.12. This continues Solution 5.11.
The decrease in negative log-likelihood[a] from Solution 5.4 is

$$\begin{aligned} 6\log 2 - (6\log 3 - 4\log 2) &= 6\log 2 - 6\log 3 + 4\log 2 && \text{(remove the parentheses)} \\ &= 10\log 2 - 6\log 3 && \text{(collect the } \log 2 \text{ terms)}, \end{aligned}$$

which is 0.3398, so the likelihood[b] ratio between the fitted model and the model with $\beta_1 = 0$ is $e^{0.3398}$, which is 1.405. The fitted model with $\beta_1 = 0$ has $\hat{\beta}_0 = \text{logit}(3/6)$, which is 0, so its negative log-likelihood is the $6\log 2$ of Solution 5.4.
The likelihood ratio test statistic[c] is twice the change in log-likelihood, $G^2 = 2 \times 0.3398$, which is 0.6796. Against the chi-square distribution[d] χ^2_1, it gives $p = 0.41$.

[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/sds/intro-MLEs.html#def-lrt-stat
[d] https://morrison-lab.github.io/sds/inference.html#def-chi-square-dist

Exercise 5.13 (A logistic regression you can fit by hand (part 6c))**.** Use the setup of Exercise 5.1. Explain what the result of the test of Exercise 5.12 implies for the trial.

Solution

Solution 5.13. This continues Solution 5.11.
The test of Solution 5.12 shows that the six-patient trial gives very little evidence against the null hypothesis[a] $\beta_1 = 0$. A fourfold odds ratio[b] ($e^{2\hat{\beta}_1} = 4$) is a large point estimate[c], but six observations are too few to distinguish it from chance variation. The estimate $\hat{\beta}_1 = \log 2$ has a standard error[d] of $\sqrt{3}/2$, which is 0.866, larger than the estimate itself. That standard error comes from the observed information[e] matrix. Every observation has $W_{ii} = \frac{1}{3} \times \frac{2}{3}$, which is $\frac{2}{9}$, and $\mathbf{D}(\mathbf{x})^\top \mathbf{D}(\mathbf{x}) = 6I$, so

$$\begin{aligned}\mathbf{D}(\mathbf{x})^\top W \mathbf{D}(\mathbf{x}) &= \tfrac{2}{9}\mathbf{D}(\mathbf{x})^\top \mathbf{D}(\mathbf{x}) && (W = \tfrac{2}{9} I) \\ &= \tfrac{2}{9} \cdot 6I && (\mathbf{D}(\mathbf{x})^\top \mathbf{D}(\mathbf{x}) = 6I) \\ &= \tfrac{4}{3} I,\end{aligned}$$

and the standard error of $\hat{\beta}_1$ is $\sqrt{\left(\frac{4}{3}\right)^{-1}}$, which equals $\sqrt{3}/2$.

[a] https://morrison-lab.github.io/sds/inference.html#def-null-hypothesis
[b] https://morrison-lab.github.io/sds/categorical-tests.html#def-odds-ratio
[c] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[d] https://morrison-lab.github.io/sds/estimation.html#def-SE
[e] https://morrison-lab.github.io/sds/intro-MLEs.html#def-oinf

Exercise 5.14 (Choosing the step size for the fit you just did by hand)**.** The fit of Exercise 5.1 has an exact solution, $\hat{\vec{\beta}} = (0, \log 2)$ (Exercise 5.5 and Exercise 5.6), so the exact error[a] of gradient ascent[b] can be computed at every step. Run gradient ascent on the log-likelihood,

$$\vec{\beta}^{(t+1)} = \vec{\beta}^{(t)} + \alpha\, \mathbf{D}(\mathbf{x})^\top \big(\underset{\sim}{y} - \underset{\sim}{\pi}^{(t)}\big),$$

starting from $\vec{\beta}^{(0)} = \vec{0}$.
Show that β_0 stays at 0 for every step, whatever α is, so that the iteration changes only β_1.

[a] https://morrison-lab.github.io/mds/optimization-newton.html#def-iterate-error
[b] https://morrison-lab.github.io/mds/optimization-gradient-descent.html#def-gradient-ascent

Solution

Solution 5.14.
The intercept[a] column of $\mathbf{D}(\mathbf{x})$ is all ones, so $\sum_i (y_i - \pi_i)$ is the first component of the gradient. Suppose $\beta_0 = 0$ at some step. Then $z_i = \beta_1 x_i$, so the three controls all have $\pi = \operatorname{expit}(-\beta_1)$, and the three treated patients all have $\pi = \operatorname{expit}(\beta_1)$. By part 2 of the sigmoid properties exercise, $\operatorname{expit}(-\beta_1) + \operatorname{expit}(\beta_1) = 1$, so

$$\begin{aligned}\sum_i \pi_i &= 3\operatorname{expit}(-\beta_1) + 3\operatorname{expit}(\beta_1) && \text{(three patients per group)} \\ &= 3[\operatorname{expit}(-\beta_1) + \operatorname{expit}(\beta_1)] && \text{(factor out 3)} \\ &= 3 && (\operatorname{expit}(-a) + \operatorname{expit}(a) = 1),\end{aligned}$$

which equals $\sum_i y_i$, the three recoveries. The first component of the gradient is therefore zero, and the step does not change the intercept. The intercept starts at 0, so it stays at 0 for every step and every α.
Exercise 5.5 used the same symmetry to show that $\hat{\beta}_0 = 0$; here it applies at every step, not only at the maximizer.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept

Exercise 5.15 (Choosing the step size for the fit you just did by hand (part 2))**.** Use the setup of Exercise 5.14.
Show that the remaining component of the gradient satisfies

$$g(\beta_1) = 4 - 6\ \operatorname{expit}(\beta_1).$$

Solution

Solution 5.15.
In the iteration of Exercise 5.14, the slope[a] component of the gradient has the form $g(\beta_1) = \sum_i x_i(y_i - \pi_i)$. With $\beta_0 = 0$, the controls have $x_i = -1$ and $\pi_i = \operatorname{expit}(-\beta_1)$, and the treated patients have $x_i = 1$ and $\pi_i = \operatorname{expit}(\beta_1)$, so

$$\begin{aligned} g(\beta_1) &= -[1 - 3\operatorname{expit}(-\beta_1)] + [2 - 3\operatorname{expit}(\beta_1)] && \text{(sum each group)} \\ &= -1 + 3\operatorname{expit}(-\beta_1) + 2 - 3\operatorname{expit}(\beta_1) && \text{(remove the brackets)} \\ &= 1 + 3\operatorname{expit}(-\beta_1) - 3\operatorname{expit}(\beta_1) && (-1 + 2 = 1) \\ &= 1 + 3[1 - \operatorname{expit}(\beta_1)] - 3\operatorname{expit}(\beta_1) && (\operatorname{expit}(-a) = 1 - \operatorname{expit}(a)) \\ &= 1 + 3 - 3\operatorname{expit}(\beta_1) - 3\operatorname{expit}(\beta_1) && \text{(distribute 3)} \\ &= 4 - 6\ \operatorname{expit}(\beta_1) && \text{(collect terms).} \end{aligned}$$

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope

Exercise 5.16 (Choosing the step size for the fit you just did by hand (part 3))**.** Use the setup of Exercise 5.14.
Check $g(0)$ against the value that Exercise 5.2 found at $\vec{\beta} = \vec{0}$.

Solution

Solution 5.16. At $\beta_1 = 0$,

$$\begin{aligned} g(0) &= 4 - 6 \cdot \tfrac{1}{2} \quad (\operatorname{expit}(0) = \tfrac{1}{2}) \\ &= 1, \end{aligned}$$

which is the slope[a] component that Solution 5.2 found.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope

Exercise 5.17 (Choosing the step size for the fit you just did by hand (part 3b))**.** Use the setup of Exercise 5.14.
Check $g(\log 2)$ against the score equation of Exercise 5.6.

Solution

Solution 5.17. This continues Solution 5.16.
At $\beta_1 = \log 2$,

$$\begin{aligned} g(\log 2) &= 4 - 6 \cdot \tfrac{2}{3} \quad (\operatorname{expit}(\log 2) = \tfrac{2}{3}) \\ &= 0, \end{aligned}$$

which agrees with the slope[a] score equation of Solution 5.6. The whole run is the one-variable iteration $\beta_1 \leftarrow \beta_1 + \alpha[4 - 6\operatorname{expit}(\beta_1)]$.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope

Exercise 5.18 (Choosing the step size for the fit you just did by hand (part 4))**.** Use the setup of Exercise 5.14.
Let $\alpha = \frac{1}{2}$, 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 each error to the one before it. What value does that ratio approach?

Solution

Solution 5.18. With $\alpha = \frac{1}{2}$:

5.0.3 R

```
# the one-variable iteration beta_1 <- beta_1 + rate * (4 - 6 * sigmoid(beta_1))
ascent_steps <- function(rate, steps, start = 0) {
  b <- numeric(steps)
  current <- start
  for (t in seq_len(steps)) {
    current <- current + rate * (4 - 6 * plogis(current))
    b[t] <- current
  }
  b
}
half_b <- ascent_steps(rate = 1 / 2, steps = 5)
half_err <- half_b - log(2)
knitr::kable(
  data.frame(
    step = 1:5,
    b = sprintf("%.6f", half_b),
    err = sprintf("%.6f", half_err),
    ratio = c("---", sprintf("%.4f", half_err[-1] / half_err[-5]))
  ),
  col.names = c(
    "step", "$\\coef{1}$", "$\\coef{1} - \\log 2$", "ratio to previous"
  )
)
```

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

5.0.4 Python

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

def ascent_steps(rate, steps, start=0.0):
    b, current = [], start
    for _ in range(steps):
        current = current + rate * (4 - 6 * expit(current))
        b.append(current)
    return np.array(b)

half_b = ascent_steps(0.5, 5)
half_err = half_b - np.log(2)
for t in range(5):
    ratio = "---" if t == 0 else f"{half_err[t] / half_err[t - 1]:.4f}"
    print(f"{t + 1}  {half_b[t]:.6f}  {half_err[t]:.6f}  {ratio}")
```

```
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
```

Every error is negative, so every iterate is below $\log 2$, which is 0.693147, and the error shrinks by a factor that approaches $1/3$.

Exercise 5.19 (Choosing the step size for the fit you just did by hand (part 5))**.** Use the setup of Exercise 5.14.
Now let $\alpha = 2$, and give β_1 after each of five steps.

Solution

Solution 5.19.

```
two_b <- ascent_steps(rate = 2, steps = 5)
sprintf("%.6f", two_b)
```

```
[1] "2.000000"  "-0.569565" "3.094388"  "-0.385540" "2.756962"
```

With $\alpha = 2$, the five iterates are 2.000000, -0.569565, 3.094388, -0.385540, 2.756962. They alternate between values below and above $\log 2$, and they do not approach it.

Exercise 5.20 (Choosing the step size for the fit you just did by hand (part 6))**.** Use the setup of Exercise 5.14.
Do the iterates grow without bound? Prove your answer, using the bound on $\|\nabla C_i\|$ from Exercise 4.6.

Solution

Solution 5.20. With $\alpha = 2$, the iterates overshoot the maximizer at each step, but they stay bounded. Exercise 4.6 bounds each observation's gradient by $\|\underset{\sim}{x}_i\|$, and here that bound makes the one-variable gradient bounded. Because $\operatorname{expit}(\beta_1)$ is strictly between 0 and 1,

$$g(\beta_1) = 4 - 6\operatorname{expit}(\beta_1) \in (-2, 4)$$

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 keeps the iterates in a closed, bounded interval. Let $K = [\log 2 - 2\alpha,\ \log 2 + 4\alpha]$, and consider any $\beta_1 \in K$.

- If $\beta_1 \geq \log 2$, then $\operatorname{expit}(\beta_1) \geq \frac{2}{3}$, so $g(\beta_1) \leq 0$, and the step moves β_1 down or not at all. The new value is at most β_1, which is at most $\log 2 + 4\alpha$, and more than $\beta_1 - 2\alpha$, which is at least $\log 2 - 2\alpha$.
- If $\beta_1 \leq \log 2$, then $g(\beta_1) \geq 0$, and the step moves β_1 up or not at all. The new value is at least β_1, which is at least $\log 2 - 2\alpha$, and less than $\beta_1 + 4\alpha$, which is at most $\log 2 + 4\alpha$.

In both cases the next iterate is in K.

The iteration starts at $\beta_1 = 0$, which is in K whenever $2\alpha \geq \log 2$. For $\alpha = 2$, the sequence therefore stays in $[-3.307, 8.693]$ for every t, and all five iterates of Solution 5.19 are in this interval.

Exercise 5.21 (Choosing the step size for the fit you just did by hand (part 7))**.** Use the setup of Exercise 5.14.

Linearize the update map of Exercise 5.14 about $\hat{\beta}_1 = \log 2$ and find its derivative there.

Solution

Solution 5.21. Let the update map be $T(\beta_1) = \beta_1 + \alpha[4 - 6\operatorname{expit}(\beta_1)]$. Its derivative is

$$\begin{aligned} T'(\beta_1) &= 1 - 6\alpha \operatorname{expit}'(\beta_1) && \text{(differentiate term by term)} \\ &= 1 - 6\alpha \operatorname{expit}(\beta_1)[1 - \operatorname{expit}(\beta_1)] && (\operatorname{expit}' = \operatorname{expit}(1 - \operatorname{expit})). \end{aligned}$$

The second line uses the derivative of expit from part 3 of the sigmoid properties exercise.

At the fixed point $\beta_1 = \log 2$, $\operatorname{expit}(\beta_1) = \frac{2}{3}$, so

$$\begin{aligned} \operatorname{expit}'(\log 2) &= \tfrac{2}{3} \cdot \tfrac{1}{3} \\ &= \tfrac{2}{9} \end{aligned}$$

and

$$\begin{aligned} T'(\log 2) &= 1 - 6\alpha \cdot \tfrac{2}{9} \\ &= 1 - \tfrac{4}{3}\alpha. \end{aligned}$$

Near $\log 2$, the linearized map is $T(\beta_1) \approx \log 2 + T'(\log 2)(\beta_1 - \log 2)$, so each step multiplies the error $\beta_1 - \log 2$ by about $1 - \frac{4}{3}\alpha$.

Exercise 5.22 (Choosing the step size for the fit you just did by hand (part 8))**.** Use the setup of Exercise 5.14.

For which α does the iteration converge?

Solution

Solution 5.22. Near $\log 2$, each step multiplies the error by about $1 - \frac{4}{3}\alpha$, so the iteration converges to $\log 2$ from starting points near it when $\left|1 - \frac{4}{3}\alpha\right| < 1$. That inequality holds when

$$0 < \alpha < \tfrac{3}{2}.$$

Exercise 5.23 (Choosing the step size for the fit you just did by hand (part 9))**.** Use the setup of Exercise 5.14.
Check that your answer accounts for both Exercise 5.18 and Exercise 5.19.

Solution

Solution 5.23. Both earlier parts follow from the multiplier $1 - \frac{4}{3}\alpha$.

- At $\alpha = \frac{1}{2}$, the multiplier is $1 - \frac{2}{3}$, which is $\frac{1}{3}$. It is positive, so the error keeps its sign, and it equals the ratio $1/3$ that the errors approach in Solution 5.0.4.
- At $\alpha = 2$, the multiplier is $1 - \frac{8}{3}$, which is $-\frac{5}{3}$. It is negative, so the error changes sign at every step, and its absolute value is greater than 1, so iterates near $\log 2$ move away from it.

With Solution 5.20, this result determines the behavior at $\alpha = 2$: the sequence does not converge to $\log 2$, and it does not leave the interval K.

Exercise 5.24 (Choosing the step size for the fit you just did by hand (part 10))**.** Use the setup of Exercise 5.14.
The step-size condition of Corollary 4.1 is $\alpha < 8/\lambda_{\max}(\mathbf{D}(\mathbf{x})^\top \mathbf{D}(\mathbf{x}))$. Evaluate it here.

Solution

Solution 5.24. The columns of $\mathbf{D}(\mathbf{x})$ are the intercept[a] column of ones and the x column $(-1, -1, -1, +1, +1, +1)$. These two columns are orthogonal[b], and each has squared length 6, so

$$\mathbf{D}(\mathbf{x})^\top \mathbf{D}(\mathbf{x}) = \begin{pmatrix} 6 & 0 \\ 0 & 6 \end{pmatrix}, \qquad \lambda_{\max}\big(\mathbf{D}(\mathbf{x})^\top \mathbf{D}(\mathbf{x})\big) = 6,$$

and the bound gives $\alpha < 8/6$, which is $4/3$.

[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

Exercise 5.25 (Choosing the step size for the fit you just did by hand (part 10b))**.** Use the setup of Exercise 5.14.
Compare the bound of Exercise 5.24 with the exact threshold from Exercise 5.22.

Solution

Solution 5.25. This continues Solution 5.24.
The bound of Solution 5.24 is $4/3$, and the exact threshold of Solution 5.22 is $3/2$, so the bound is smaller than the threshold by a factor of $\frac{3}{2} \div \frac{4}{3}$, which is $\frac{9}{8}$.

Exercise 5.26 (Choosing the step size for the fit you just did by hand (part 10c))**.** Use the setup of Exercise 5.14.
Explain the difference between the bound of Exercise 5.24 and the exact threshold from Exercise 5.22.

Solution

Solution 5.26. This continues Solution 5.24.
The difference between the bound and the exact threshold (Solution 5.25) comes from the inequality $W_{ii} \leq 1/4$. Equality holds only where $\pi_i = 1/2$, but at the maximum likelihood[a] estimate[b] the fitted probabilities are 1/3 and 2/3, so every W_{ii} is $\frac{1}{3} \cdot \frac{2}{3}$, which is $\frac{2}{9}$. The Hessian[c] at the maximum is therefore $\frac{2}{9}\mathbf{D}(\mathbf{x})^\top \mathbf{D}(\mathbf{x})$, which is $\frac{4}{3}I$, rather than the bound $\frac{1}{4}\mathbf{D}(\mathbf{x})^\top \mathbf{D}(\mathbf{x})$, which is $\frac{3}{2}I$. For the Hessian at the maximum, $2/\lambda_{\max}$ is 3/2, the exact threshold. The ratio $\frac{9}{8}$ equals $\frac{1}{4} \div \frac{2}{9}$.
The bound holds for *every* $\vec{\beta}$, and it is smaller than the exact threshold by the factor by which $\pi_i(1-\pi_i)$ at the maximum is below 1/4. A fit that separates the classes[d] well has fitted probabilities near 0 and 1, so it has small curvature at the maximum and a larger stable step size.

[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/mds/vector-calculus.html#def-hessian
[d] https://morrison-lab.github.io/sds/classification.html#def-class

Exercise 5.27 (Choosing the step size for the fit you just did by hand (part 11))**.** Use the setup of Exercise 5.14.
Run the iteration with $\alpha = 2$ for a few thousand steps.

Solution

Solution 5.27.

```
long_two <- ascent_steps(rate = 2, steps = 4000)
last_two <- tail(long_two, 2)
cycle_two <- sort(last_two)
cycle_two
```

```
[1] -0.4689554  2.9126825
```

The last two values of β_1 after 4000 steps are -0.468955 and 2.912682.

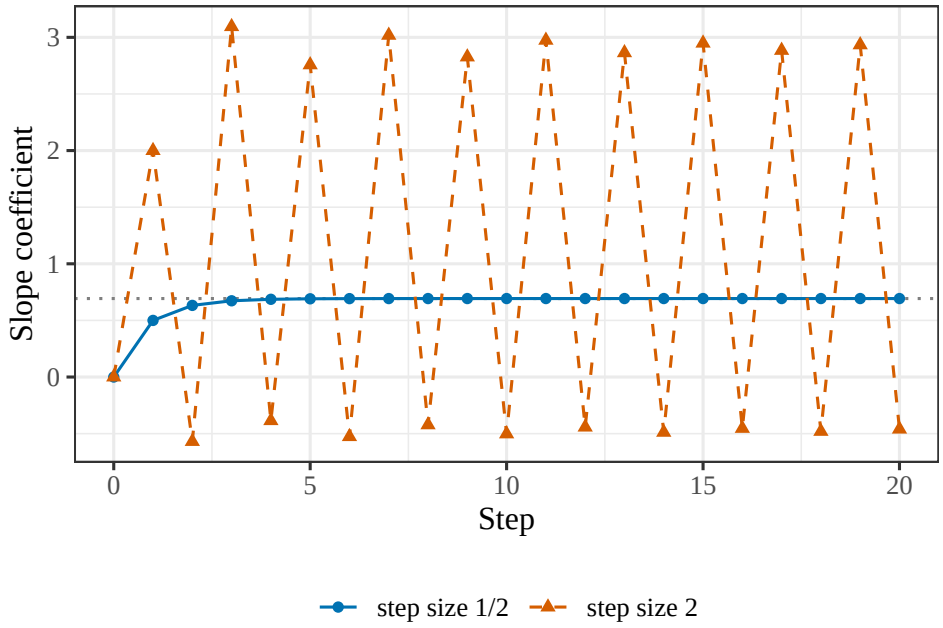
Exercise 5.28 (Choosing the step size for the fit you just did by hand (part 11b))**.** Use the setup of Exercise 5.14.
Describe the behavior of β_1 after many steps of the iteration of Exercise 5.27.

Solution

Solution 5.28. This continues Solution 5.27.
For $\alpha = 2$, the iterates approach a cycle of period 2, a pair of points that the update map T sends to each other. The sequence does not converge to a single point, and it does not grow without bound.
The two points of the cycle are on either side of $\log 2$, where the update map T has its fixed point. The multiplier of Solution 5.21 at that fixed point is $1 - \frac{4}{3}\alpha$, which equals $-\frac{5}{3}$ for $\alpha = 2$. That is below -1, so the iterates leave the fixed point.
Figure 24 plots the iterates for $\alpha = 2$ and, for comparison, for $\alpha = \frac{1}{2}$.

5.0.5 R

```r
asp_n <- 21
asp_steps <- seq_len(asp_n) - 1
asp_half <- numeric(asp_n)
asp_two <- numeric(asp_n)
for (asp_i in 2:asp_n) {
  asp_prev <- asp_half[asp_i - 1]
  asp_grad <- 4 - 6 * plogis(asp_prev)
  asp_half[asp_i] <- asp_prev + 0.5 * asp_grad
  asp_prev2 <- asp_two[asp_i - 1]
  asp_grad2 <- 4 - 6 * plogis(asp_prev2)
  asp_two[asp_i] <- asp_prev2 + 2 * asp_grad2
}
asp_names <- c("step size 1/2", "step size 2")
asp_step <- rep(asp_steps, 2)
asp_value <- c(asp_half, asp_two)
asp_rate <- rep(asp_names, each = asp_n)
asp_curves <- tibble::tibble(
  step = asp_step,
  value = asp_value,
  rate = asp_rate
)
asp_colours <- c("step size 1/2" = "#0072B2", "step size 2" = "#D55E00")
asp_types <- c("step size 1/2" = "solid", "step size 2" = "dashed")
asp_shapes <- c("step size 1/2" = 16, "step size 2" = 17)
asp_target <- log(2)
asp_curves |>
  ggplot2::ggplot(ggplot2::aes(step, value, colour = rate)) +
  ggplot2::geom_hline(
    yintercept = asp_target,
    linetype = "dotted",
    colour = "grey50"
  ) +
  ggplot2::geom_line(ggplot2::aes(linetype = rate)) +
  ggplot2::geom_point(ggplot2::aes(shape = rate)) +
  ggplot2::scale_colour_manual(values = asp_colours) +
  ggplot2::scale_linetype_manual(values = asp_types) +
  ggplot2::scale_shape_manual(values = asp_shapes) +
  ggplot2::labs(
    x = "Step",
    y = "Slope coefficient",
    colour = NULL,
    linetype = NULL,
    shape = NULL
  )
```


Slope coefficient
3
2
1
0
0
5
10
15
20
Step
step size 1/2
step size 2

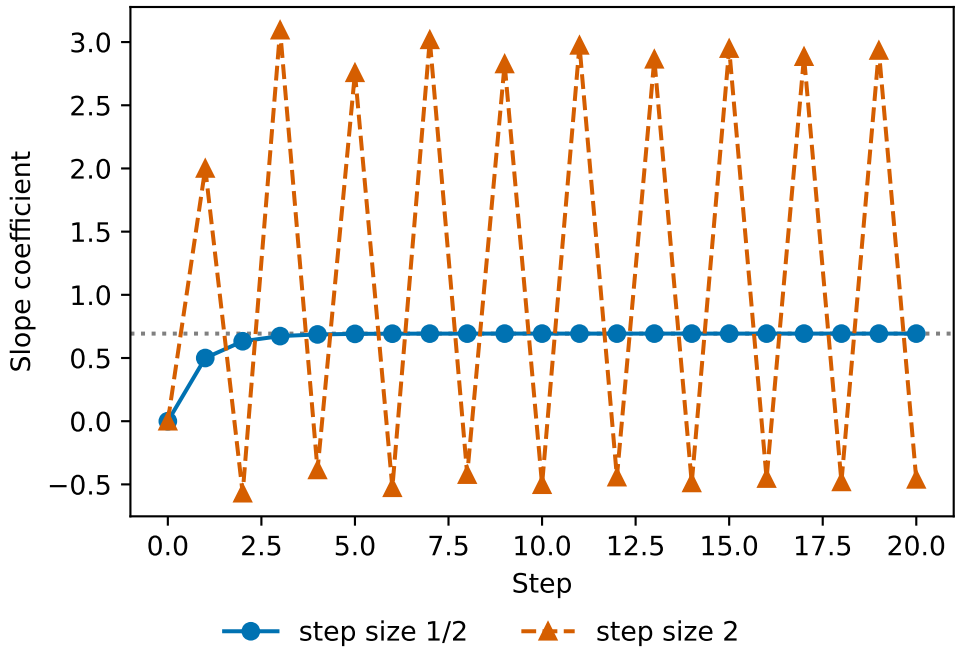
(a)

5.0.6 Python

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

steps_asp = np.arange(21)
half_asp = np.zeros(21)
two_asp = np.zeros(21)
for i in range(1, 21):
    half_asp[i] = half_asp[i - 1] + 0.5 * (4 - 6 * expit(half_asp[i - 1]))
    two_asp[i] = two_asp[i - 1] + 2 * (4 - 6 * expit(two_asp[i - 1]))

fig, ax = plt.subplots(figsize=(5, 3.5), layout="constrained")
ax.axhline(np.log(2), color="grey", linestyle=":");
ax.plot(steps_asp, half_asp, color="#0072B2", marker="o", label="step size 1/2");
ax.plot(steps_asp, two_asp, color="#D55E00", linestyle="--", marker="^",
        label="step size 2");
ax.set_xlabel("Step");
ax.set_ylabel("Slope coefficient");
fig.legend(loc="outside lower center", ncol=2, frameon=False);
plt.show()
```



(b)

Figure 24: The iterates of β_1 for a step size of 1/2 and of 2, with the dotted line at $\log 2$.

Exercise 5.29 (Choosing the step size for the fit you just did by hand (part 12))**.** Use the setup of Exercise 5.14.
State what happens instead when the step size is too large for the least-squares cost, whose step-size bound the chapter on gradient descent[a] derives.

[a] https://morrison-lab.github.io/mds/optimization-gradient-descent.html#def-gradient-descent

Solution

Solution 5.29. Least squares behaves differently. Its objective is quadratic, so the update is linear, and each step multiplies the error component along an eigenvector by $1 - 2\alpha\lambda$, with no higher-order terms. When $|1 - 2\alpha\lambda| > 1$ for some eigenvalue[a] λ, that component of the error is multiplied by the same factor at every step, so it grows geometrically and the iterates grow without bound.
In this one-variable logistic regression, the total log loss does not lead to this growth, for the reason shown in Solution 5.20. The fitted probabilities are in $(0, 1)$, so the gradient is bounded on $\mathbb{R}$, and a step size that is too large gives bounded iterates that do not converge, such as the cycle of period 2 at $\alpha = 2$. A step size that is too large therefore shows up differently in the two problems: the least-squares iterates grow until they overflow, and these logistic regression iterates stay bounded without converging. The bounded gradient bounds each step, not the iterates, so this argument does not carry over to every logistic regression: when the classes[b] can be separated, the iterates grow without bound at any step size.

[a] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-eigenvalue
[b] https://morrison-lab.github.io/sds/classification.html#def-class

6 Newton's method

Gradient ascent[30] uses the first derivative[31] of the log-likelihood[32], and needs a step size chosen by the user. Newton's method[33] also uses the second derivative[34], which sets both the direction and the length of each step. When the maximum likelihood estimate[35] exists and the model has a modest number of coefficients, Newton's method started at $\vec{0}$ usually reaches the maximizer, up to rounding error, in a few iterations, and R's `glm()` fits logistic regression with it.

Exercise 6.1 (Newton's method, and why it is a weighted least squares fit)**.** In Exercise 5.18, the error[a] of gradient ascent[b] shrank by a factor of about 3 per step, at a step size that had to be chosen. Newton's method[c] uses the second derivative[d] of ℓ as well as the first, and has no step size to choose. Write the second-order Taylor expansion[e] of $\ell(\vec{\beta}^{(t)} + \underset{\sim}{d})$ about the current iterate[f] $\vec{\beta}^{(t)}$, as a function of the step $\underset{\sim}{d}$.

[a] https://morrison-lab.github.io/mds/optimization-newton.html#def-iterate-error
[b] https://morrison-lab.github.io/mds/optimization-gradient-descent.html#def-gradient-ascent
[c] https://morrison-lab.github.io/mds/optimization-newton.html#def-newton-method
[d] https://morrison-lab.github.io/mds/calculus-derivatives.html#def-second-derivative
[e] https://morrison-lab.github.io/mds/calculus-taylor-series.html#def-taylor-polynomial
[f] https://morrison-lab.github.io/mds/optimization-newton.html#def-iterate

Solution

Solution 6.1. By Proposition 1.1, $\ell = -C$, so Theorem 4.2 and Theorem 4.3 give the gradient and Hessian[a] of ℓ:

$$\nabla \ell(\vec{\beta}) = \mathbf{D}^\top (\underset{\sim}{y} - \underset{\sim}{\pi}), \qquad \nabla^2 \ell(\vec{\beta}) = -\mathbf{D}^\top W \mathbf{D}.$$

Write $\underset{\sim}{\pi}^{(t)}$ and $W^{(t)}$ for $\underset{\sim}{\pi}$ and W evaluated at $\vec{\beta}^{(t)}$. Write $q(\underset{\sim}{d})$ for the second-order Taylor expansion[b] of ℓ about $\vec{\beta}^{(t)}$:

$$\ell(\vec{\beta}^{(t)} + \underset{\sim}{d}) \approx q(\underset{\sim}{d}) = \ell(\vec{\beta}^{(t)}) + (\mathbf{D}\underset{\sim}{d})^\top (\underset{\sim}{y} - \underset{\sim}{\pi}^{(t)}) - \tfrac{1}{2} \underset{\sim}{d}^\top \mathbf{D}^\top W^{(t)} \mathbf{D} \underset{\sim}{d}.$$

[30] https://morrison-lab.github.io/mds/optimization-gradient-descent.html#def-gradient-ascent
[31] https://morrison-lab.github.io/mds/calculus-derivatives.html#def-derivative
[32] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik
[33] https://morrison-lab.github.io/mds/optimization-newton.html#def-newton-method
[34] https://morrison-lab.github.io/mds/calculus-derivatives.html#def-second-derivative
[35] https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle

[a] https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian
[b] https://morrison-lab.github.io/mds/calculus-taylor-series.html#def-taylor-polynomial

Exercise 6.2 (Newton's method, and why it is a weighted least squares fit (part 1b))**.** Use the setup of Exercise 6.1.
Find the equation that the step $\underset{\sim}{d}$ maximizing the expansion of Exercise 6.1 satisfies.

Solution

Solution 6.2. This continues Solution 6.1.
Let q be the expansion of Solution 6.1. By Theorem 4.3, the matrix[a] $\mathbf{D}^\top W^{(t)}\mathbf{D}$ is positive semidefinite[b], so q is a concave quadratic in $\underset{\sim}{d}$, and a step $\underset{\sim}{d}$ maximizes q exactly when $\nabla q(\underset{\sim}{d}) = \vec{0}$:

$$\begin{aligned} \nabla q(\underset{\sim}{d}) &= \mathbf{D}^\top\big(\underset{\sim}{y} - \underset{\sim}{\pi}^{(t)}\big) - \mathbf{D}^\top W^{(t)}\mathbf{D}\underset{\sim}{d} && \text{(differentiate each term of } q\text{)} \\ &= \vec{0} && \text{(stationarity)} \\ \implies \quad \mathbf{D}^\top W^{(t)}\mathbf{D}\underset{\sim}{d} &= \mathbf{D}^\top\big(\underset{\sim}{y} - \underset{\sim}{\pi}^{(t)}\big). && \text{(move the second term to the right)} \end{aligned}$$

[a] https://morrison-lab.github.io/mds/linear-algebra-matrices.html#def-matrix
[b] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-positive-semidefinite

Exercise 6.3 (Newton's method, and why it is a weighted least squares fit (part 2))**.** Use the setup of Exercise 6.1.
Assume that $\mathbf{D}$ has full column rank[a]. Show that the Newton update $\vec{\beta}^{(t+1)} = \vec{\beta}^{(t)} + \underset{\sim}{d}$ is

$$\vec{\beta}^{(t+1)} = \vec{\beta}^{(t)} + \big(\mathbf{D}^\top W^{(t)}\mathbf{D}\big)^{-1}\mathbf{D}^\top\big(\underset{\sim}{y} - \underset{\sim}{\pi}^{(t)}\big),$$

where $W^{(t)}$ is the weight[b] matrix[c] from Theorem 4.3 evaluated at $\vec{\beta}^{(t)}$.

[a] https://morrison-lab.github.io/mds/linear-algebra-subspaces.html#def-rank
[b] https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-weight
[c] https://morrison-lab.github.io/mds/linear-algebra-matrices.html#def-matrix

Solution

Solution 6.3. Every diagonal entry[a] $\pi_i(1-\pi_i)$ of $W^{(t)}$ is positive, so $\underset{\sim}{d}^\top\mathbf{D}^\top W^{(t)}\mathbf{D}\underset{\sim}{d} = \sum_{i=1}^n \pi_i(1-\pi_i)(\underset{\sim}{x}_i \cdot \underset{\sim}{d})^2$ is 0 only when $\mathbf{D}\underset{\sim}{d} = \vec{0}$, which for a full-column-rank $\mathbf{D}$ requires $\underset{\sim}{d} = \vec{0}$. So $\mathbf{D}^\top W^{(t)}\mathbf{D}$ is positive definite[b] and invertible[c]. Multiplying the equation of Solution 6.2 on the left by its inverse[d] gives

$$\underset{\sim}{d} = \big(\mathbf{D}^\top W^{(t)}\mathbf{D}\big)^{-1}\mathbf{D}^\top\big(\underset{\sim}{y} - \underset{\sim}{\pi}^{(t)}\big),$$

and adding $\underset{\sim}{d}$ to $\vec{\beta}^{(t)}$ gives the stated update.

[a] https://morrison-lab.github.io/mds/linear-algebra-matrices.html#def-diagonal-entry
[b] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-positive-definite
[c] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-matrix-inverse
[d] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-matrix-inverse

Theorem 6.1 (The logistic log-likelihood is strictly concave)**.** *Let $\ell = -C$ be the log-likelihood[a], the negative of the total log loss C, and let W be the diagonal[b] matrix of Theorem 4.3. If the design matrix $\mathbf{D}$ has full column rank[c], then $\nabla^2\ell(\vec{\beta}) = -\mathbf{D}^\top W\mathbf{D}$ is negative definite for every $\vec{\beta}$. So ℓ is strictly concave, and has at most one maximizer.*

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik

[b] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-diagonal-matrix
[c] https://morrison-lab.github.io/mds/linear-algebra-subspaces.html#def-rank

Proof

Proof. For any $\underset{\sim}{d} \neq \vec{0}$, the diagonal[a] entries $\pi_i(1-\pi_i)$ of W are positive, so

$$\begin{aligned} \underset{\sim}{d}^\top \mathbf{D}^\top W \mathbf{D} \underset{\sim}{d} &= \sum_{i=1}^{n} \pi_i(1-\pi_i)(\underset{\sim}{x}_i \cdot \underset{\sim}{d})^2 && \text{(expand the quadratic form)} \\ &> 0. && (\mathbf{D}\underset{\sim}{d} \neq \vec{0} \text{ for a full-column-rank } \mathbf{D}) \end{aligned}$$

So $\mathbf{D}^\top W \mathbf{D}$ is positive definite[b], and the Hessian[c] $-\mathbf{D}^\top W \mathbf{D}$ of ℓ (Theorem 4.3) is negative definite. Then $-\ell$ is strictly convex[d] by the second-derivative test[e], and so has at most one global minimizer[f]. □

[a] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-diagonal-matrix
[b] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-positive-definite
[c] https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian
[d] https://morrison-lab.github.io/mds/optimization-convexity.html#def-strictly-convex
[e] https://morrison-lab.github.io/mds/optimization-convexity.html#thm-convex-hessian
[f] https://morrison-lab.github.io/mds/optimization-convexity.html#cor-convex-stationary

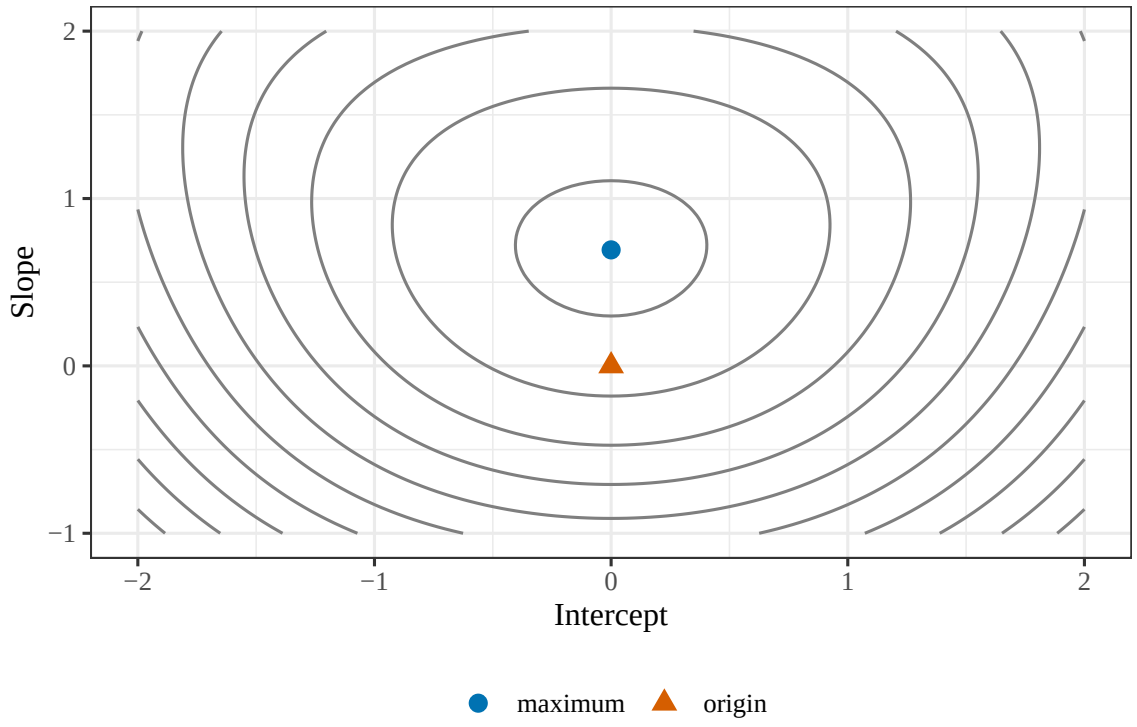
Example 6.1 (The six-patient study is strictly concave)**.** In Exercise 5.1, the columns of $\mathbf{D}$ are the ones vector and $\underset{\sim}{x} = (-1,-1,-1,1,1,1)^\top$. They are orthogonal[a], each with squared length[b] 6, so $\mathbf{D}$ has full column rank[c]. At $\vec{\beta} = \vec{0}$, every $\pi_i = \frac{1}{2}$, so $W = \frac{1}{4}\mathbb{I}$ and

$$\begin{aligned} \nabla^2 \ell(\vec{0}) &= -\tfrac{1}{4}\mathbf{D}^\top \mathbf{D} \\ &= -\tfrac{3}{2}\mathbb{I}, \end{aligned}$$

which is negative definite. At any other $\vec{\beta}$, the entries of W change, but by Theorem 6.1 the Hessian[d] stays negative definite.
Figure 25 plots the function.

6.0.1 R

```
llc_x <- c(-1, -1, -1, 1, 1, 1)
llc_y <- c(1, 0, 0, 1, 1, 0)
llc_patients <- tibble::tibble(x = llc_x, y = llc_y)
llc_b0 <- seq(-2, 2, length.out = 121)
llc_b1 <- seq(-1, 2, length.out = 121)
llc_grid <- tidyr::expand_grid(b0 = llc_b0, b1 = llc_b1, llc_patients)
llc_terms <- llc_grid |>
  dplyr::mutate(z = b0 + b1 * x, ez = exp(z), term = y * z - log1p(ez))
llc_surface <- llc_terms |>
  dplyr::group_by(b0, b1) |>
  dplyr::summarise(llik = sum(term), .groups = "drop")
llc_where <- c("origin", "maximum")
llc_slope_max <- log(2)
llc_pts_b1 <- c(0, llc_slope_max)
llc_pts_b0 <- c(0, 0)
llc_pts <- tibble::tibble(b0 = llc_pts_b0, b1 = llc_pts_b1, where = llc_where)
llc_colours <- c(origin = "#D55E00", maximum = "#0072B2")
llc_shapes <- c(origin = 17, maximum = 16)
llc_surface |>
  ggplot2::ggplot(ggplot2::aes(b0, b1)) +
  ggplot2::geom_contour(ggplot2::aes(z = llik), colour = "grey50", bins = 12) +
  ggplot2::geom_point(
    data = llc_pts,
    ggplot2::aes(colour = where, shape = where),
    size = 3
  ) +
  ggplot2::scale_colour_manual(values = llc_colours) +
  ggplot2::scale_shape_manual(values = llc_shapes) +
  ggplot2::labs(x = "Intercept", y = "Slope", colour = NULL, shape = NULL)
```

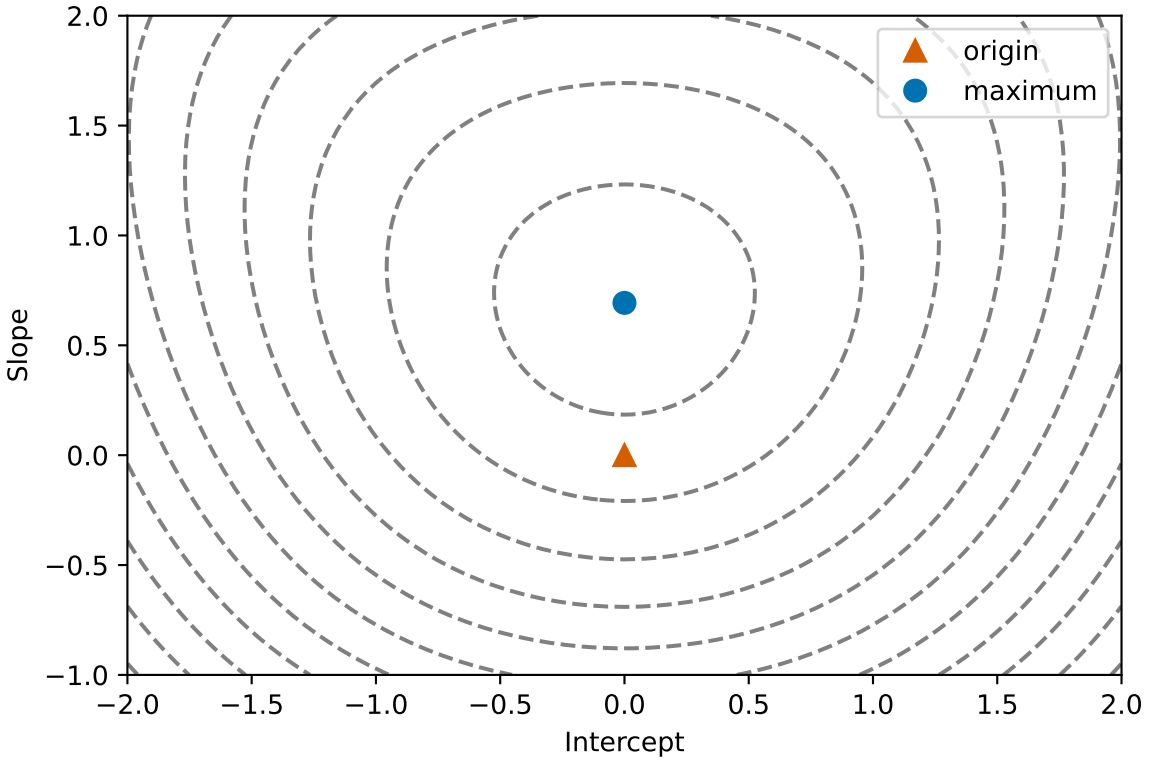


(a)

6.0.2 Python

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

x_llc = np.array([-1, -1, -1, 1, 1, 1])
y_llc = np.array([1, 0, 0, 1, 1, 0])
b0_llc, b1_llc = np.meshgrid(np.linspace(-2, 2, 121), np.linspace(-1, 2, 121))
z_llc = b0_llc[..., None] + b1_llc[..., None] * x_llc
llik_llc = (y_llc * z_llc - np.log1p(np.exp(z_llc))).sum(axis=-1)
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.contour(b0_llc, b1_llc, llik_llc, levels=12, colors="grey");
ax.plot(0, 0, color="#D55E00", marker="^", markersize=8, linestyle="", label="origin");
ax.plot(0, np.log(2), color="#0072B2", marker="o", markersize=8, linestyle="", label="maximum");
ax.set_xlabel("Intercept");
ax.set_ylabel("Slope");
ax.legend();
plt.show()
```



(b)

Figure 25: The log-likelihood[e] of Exercise 5.1, drawn as nested curves around a single peak at the maximum (blue), as strict concavity requires.

6.0.3 R

(This 3D surface is interactive; see the HTML or slide version of this page to view it.)

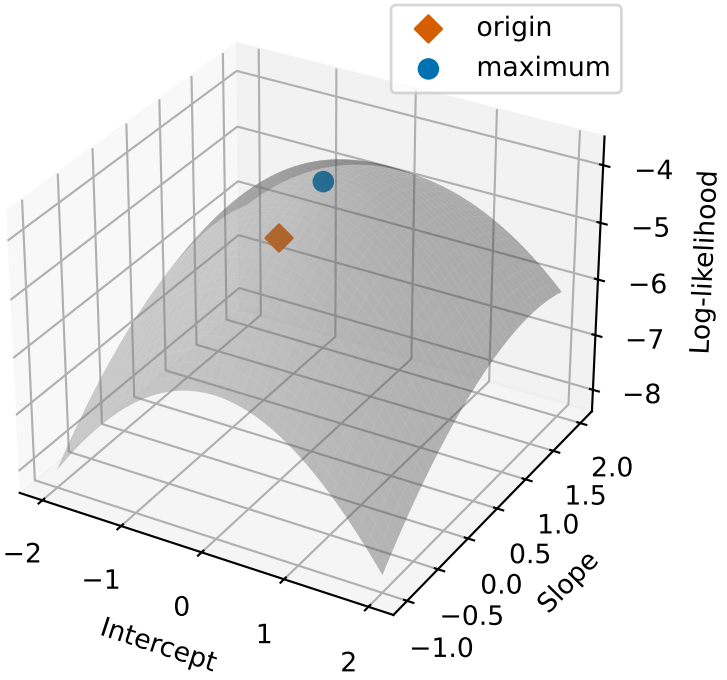
6.0.4 Python

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

x_llc3 = np.array([-1, -1, -1, 1, 1, 1])
y_llc3 = np.array([1, 0, 0, 1, 1, 0])


def loglik_llc3(b0, b1):
    z = np.asarray(b0)[..., None] + np.asarray(b1)[..., None] * x_llc3
    return (y_llc3 * z - np.log1p(np.exp(z))).sum(axis=-1)


b0_llc3, b1_llc3 = np.meshgrid(np.linspace(-2, 2, 81), np.linspace(-1, 2, 81))
fig = plt.figure(figsize=(6, 4.5))
ax = fig.add_subplot(projection="3d")
ax.plot_surface(b0_llc3, b1_llc3, loglik_llc3(b0_llc3, b1_llc3), color="grey",
                alpha=0.4);
ax.scatter([0], [0], [loglik_llc3(0, 0)], marker="D", color="#D55E00", s=50,
           label="origin");
ax.scatter([0], [np.log(2)], [loglik_llc3(0, np.log(2))], marker="o",
           color="#0072B2", s=50, label="maximum");
ax.set_xlabel("Intercept");
ax.set_ylabel("Slope");
ax.set_zlabel("Log-likelihood");
ax.legend(loc="upper right");
plt.show()
```



(a)

Figure 26: The log-likelihood[f] of the six-patient study, as a surface over the intercept and slope. The surface is a single dome with one peak at the maximum[g] (blue), as strict concavity[h] requires. The origin (vermilion) is on the side of the dome.

[a] https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-orthogonal-vectors
[b] https://morrison-lab.github.io/mds/linear-algebra-inner-products.html#def-norm
[c] https://morrison-lab.github.io/mds/linear-algebra-subspaces.html#def-rank
[d] https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian
[e] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik
[f] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik
[g] https://morrison-lab.github.io/mds/algebra-basics.html#def-maximum
[h] https://morrison-lab.github.io/mds/optimization-convexity.html#def-strictly-concave

Theorem 6.2 (The Newton update for logistic regression)**.** *Let* $\underset{\sim}{\pi}^{(t)}$ *and* $W^{(t)}$ *be the fitted probabilities*[a] *and the weight*[b] *matrix*[c] W *from Theorem 4.3, evaluated at the current iterate*[d] $\vec{\beta}^{(t)}$*. If* $\mathbf{D}$ *has full column rank*[e]*, the Newton step*[f] *from* $\vec{\beta}^{(t)}$ *on the log-likelihood*[g] ℓ *is*

$$\vec{\beta}^{(t+1)} = \vec{\beta}^{(t)} + \left(\mathbf{D}^\top W^{(t)} \mathbf{D}\right)^{-1} \mathbf{D}^\top \left(\underset{\sim}{y} - \underset{\sim}{\pi}^{(t)}\right).$$

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-probability
[b] https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-weight
[c] https://morrison-lab.github.io/mds/linear-algebra-matrices.html#def-matrix
[d] https://morrison-lab.github.io/mds/optimization-newton.html#def-iterate
[e] https://morrison-lab.github.io/mds/linear-algebra-subspaces.html#def-rank
[f] https://morrison-lab.github.io/mds/optimization-newton.html#thm-newton-model
[g] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik

Proof

Proof. Maximizing ℓ is minimizing $C = -\ell$. By Theorem 4.2, $\nabla C(\vec{\beta}^{(t)}) = -\mathbf{D}^\top\big(y - \underset{\sim}{\pi}^{(t)}\big)$, and by Theorem 6.1, the Hessian[a] $\mathbf{D}^\top W^{(t)}\mathbf{D}$ of C is positive definite[b], hence invertible[c]. So the Newton step[d] minimizing the quadratic model[e] of C is

$$-\big(\mathbf{D}^\top W^{(t)}\mathbf{D}\big)^{-1}\nabla C(\vec{\beta}^{(t)}) = \big(\mathbf{D}^\top W^{(t)}\mathbf{D}\big)^{-1}\mathbf{D}^\top\big(\underset{\sim}{y} - \underset{\sim}{\pi}^{(t)}\big),$$

and adding it to $\vec{\beta}^{(t)}$ gives the update. □

[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/mds/linear-algebra-special-matrices.html#def-matrix-inverse
[d] https://morrison-lab.github.io/mds/optimization-newton.html#def-newton-method
[e] https://morrison-lab.github.io/mds/optimization-newton.html#def-quadratic-model

Exercise 6.4 (Newton's method, and why it is a weighted least squares fit (part 3))**.** Use the setup of Exercise 6.1.
Compare the Newton update of Exercise 6.3 with the gradient ascent[a] update of Exercise 5.14. Which quantity in the gradient ascent update has no counterpart in the Newton update?

[a] https://morrison-lab.github.io/mds/optimization-gradient-descent.html#def-gradient-ascent

Solution

Solution 6.4. The step size α has no counterpart. Gradient ascent[a] moves in the direction[b] of the gradient, and the step size α, chosen separately, sets how far it moves; Exercise 5.14 studied that choice. Newton's method[c] moves straight to the maximizer of the quadratic[d] expansion q of Solution 6.1, so the matrix[e] $\big(\mathbf{D}^\top W^{(t)}\mathbf{D}\big)^{-1}$, computed from the curvature, sets both the direction and the length[f] of the step, and nothing is left to choose. Far from the maximizer the quadratic expansion can be a poor guide, and a full Newton step can overshoot, so a damped Newton method shortens a step that makes the fit worse [11, Sec. 9.5].

[a] https://morrison-lab.github.io/mds/optimization-gradient-descent.html#def-gradient-ascent
[b] https://morrison-lab.github.io/mds/vector-calculus.html#def-direction
[c] https://morrison-lab.github.io/mds/optimization-newton.html#def-newton-method
[d] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-quadratic-cubic
[e] https://morrison-lab.github.io/mds/linear-algebra-matrices.html#def-matrix
[f] https://morrison-lab.github.io/mds/linear-algebra-inner-products.html#def-norm

Exercise 6.5 (Newton's method, and why it is a weighted least squares fit (part 4))**.** Use the setup of Exercise 6.1.
The working responses[a] at $\vec{\beta}^{(t)}$ are

$$\underset{\sim}{z}^{(t)} = \mathbf{D}\vec{\beta}^{(t)} + \big(W^{(t)}\big)^{-1}\big(\underset{\sim}{y} - \underset{\sim}{\pi}^{(t)}\big).$$

Show that the Newton update of Exercise 6.3 equals

$$\vec{\beta}^{(t+1)} = \big(\mathbf{D}^\top W^{(t)}\mathbf{D}\big)^{-1}\mathbf{D}^\top W^{(t)}\underset{\sim}{z}^{(t)},$$

the weighted least squares fit of $\underset{\sim}{z}^{(t)}$ on $\mathbf{D}$ with weight[b] matrix[c] $W^{(t)}$.

[a] https://morrison-lab.github.io/sds/logistic-inference.html#def-irls
[b] https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-weight
[c] https://morrison-lab.github.io/mds/linear-algebra-matrices.html#def-matrix

Solution

Solution 6.5. Start from the right-hand side:

$$\begin{aligned}
\left(\mathbf{D}^\top W^{(t)}\mathbf{D}\right)^{-1}\mathbf{D}^\top W^{(t)}\underset{\sim}{z}^{(t)} &= \left(\mathbf{D}^\top W^{(t)}\mathbf{D}\right)^{-1}\mathbf{D}^\top W^{(t)}\left[\mathbf{D}\vec{\beta}^{(t)} + \left(W^{(t)}\right)^{-1}\left(\underset{\sim}{y} - \underset{\sim}{\pi}^{(t)}\right)\right] && \text{(definition of } \underset{\sim}{z}^{(t)}\text{)} \\
&= \left(\mathbf{D}^\top W^{(t)}\mathbf{D}\right)^{-1}\left[\mathbf{D}^\top W^{(t)}\mathbf{D}\vec{\beta}^{(t)} + \mathbf{D}^\top W^{(t)}\left(W^{(t)}\right)^{-1}\left(\underset{\sim}{y} - \underset{\sim}{\pi}^{(t)}\right)\right] && \text{(distribute } \mathbf{D}^\top W^{(t)}\text{)} \\
&= \left(\mathbf{D}^\top W^{(t)}\mathbf{D}\right)^{-1}\left[\mathbf{D}^\top W^{(t)}\mathbf{D}\vec{\beta}^{(t)} + \mathbf{D}^\top\left(\underset{\sim}{y} - \underset{\sim}{\pi}^{(t)}\right)\right] && \left(W^{(t)}\left(W^{(t)}\right)^{-1} = \mathbb{I}\right) \\
&= \vec{\beta}^{(t)} + \left(\mathbf{D}^\top W^{(t)}\mathbf{D}\right)^{-1}\mathbf{D}^\top\left(\underset{\sim}{y} - \underset{\sim}{\pi}^{(t)}\right), && \text{(distribute the inverse)}
\end{aligned}$$

which is the update of Exercise 6.3. So every Newton step is a weighted least squares fit, and both the working responses[a] $\underset{\sim}{z}^{(t)}$ and the weight[b] matrix[c] $W^{(t)}$ change from one iteration to the next. That repeated fit is the iteratively reweighted least squares[d] (IRLS) algorithm[e] of the statistics notes, which proves the same equivalence in its own theorem[f].

[a] https://morrison-lab.github.io/sds/logistic-inference.html#def-irls
[b] https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-weight
[c] https://morrison-lab.github.io/mds/linear-algebra-matrices.html#def-matrix
[d] https://morrison-lab.github.io/sds/logistic-inference.html#def-irls
[e] https://morrison-lab.github.io/mcs/expressions-algorithms.html#def-algorithm
[f] https://morrison-lab.github.io/sds/logistic-inference.html#thm-irls-newton

Theorem 6.3 (A Newton step is a weighted least squares fit)**.** *Under the assumptions of Theorem 6.2, let*

$$\underset{\sim}{z}^{(t)} = \mathbf{D}\vec{\beta}^{(t)} + \left(W^{(t)}\right)^{-1}\left(\underset{\sim}{y} - \underset{\sim}{\pi}^{(t)}\right)$$

be the working responses[a]*. Then the Newton update is the weighted least squares fit of* $\underset{\sim}{z}^{(t)}$ *on* $\mathbf{D}$ *with weight*[b] *matrix*[c] $W^{(t)}$*:*

$$\vec{\beta}^{(t+1)} = \left(\mathbf{D}^\top W^{(t)}\mathbf{D}\right)^{-1}\mathbf{D}^\top W^{(t)}\underset{\sim}{z}^{(t)}.$$

[a] https://morrison-lab.github.io/sds/logistic-inference.html#def-irls
[b] https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-weight
[c] https://morrison-lab.github.io/mds/linear-algebra-matrices.html#def-matrix

Proof

Proof. Substituting $\underset{\sim}{z}^{(t)}$ and using $W^{(t)}\left(W^{(t)}\right)^{-1} = \mathbb{I}$ gives

$$\left(\mathbf{D}^\top W^{(t)}\mathbf{D}\right)^{-1}\mathbf{D}^\top W^{(t)}\underset{\sim}{z}^{(t)} = \vec{\beta}^{(t)} + \left(\mathbf{D}^\top W^{(t)}\mathbf{D}\right)^{-1}\mathbf{D}^\top\left(\underset{\sim}{y} - \underset{\sim}{\pi}^{(t)}\right),$$

which is $\vec{\beta}^{(t+1)}$ by Theorem 6.2. □

Exercise 6.6 (The working responses of the hand-sized fit)**.** Use the setup of Exercise 6.5. Evaluate the working responses[a] $\underset{\sim}{z}^{(0)}$ for Exercise 5.1's six patients at $\vec{\beta}^{(0)} = \vec{0}$.

[a] https://morrison-lab.github.io/sds/logistic-inference.html#def-irls

Solution

Solution 6.6. At $\vec{\beta}^{(0)} = \vec{0}$, $\mathbf{D}\vec{\beta}^{(0)} = \vec{0}$ and every $\pi_i = \operatorname{expit}(0) = \frac{1}{2}$, so every weight[a] is $\frac{1}{2}\cdot\frac{1}{2} = \frac{1}{4}$ and $W^{(0)} = \frac{1}{4}\mathbb{I}$. In the order of Solution 5.1, $\underset{\sim}{y} = (0,0,1,0,1,1)^\top$, so

$$\begin{aligned} \underset{\sim}{z}^{(0)} &= \vec{0} + 4\big(\underset{\sim}{y} - \tfrac{1}{2}\vec{1}\big) && \big((W^{(0)})^{-1} = 4\mathbb{I}\big) \\ &= (-2,-2,2,-2,2,2)^\top. \end{aligned}$$

[a] https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-weight

Exercise 6.7 (The scale of the working responses)**.** Use the setup of Exercise 6.6.
On what scale are the six numbers in $\underset{\sim}{z}^{(0)}$?

Solution

Solution 6.7. These numbers are on the log-odds[a] scale. The log-odds of an outcome $y_i \in \{0,1\}$ is undefined: $\text{logit}\{0\}$ and $\text{logit}\{1\}$ would be $-\infty$ and $+\infty$, so IRLS uses a first-order approximation instead. The derivative of $\text{logit}\{\pi\} = \log\frac{\pi}{1-\pi}$ is $\frac{1}{\pi(1-\pi)}$, so the first-order Taylor expansion of logit about the current probability π_i, evaluated at y_i, is

$$\text{logit}\{\pi_i\} + \frac{y_i - \pi_i}{\pi_i(1-\pi_i)} = \underset{\sim}{x}_i \cdot \vec{\beta}^{(t)} + \frac{y_i - \pi_i}{\pi_i(1-\pi_i)},$$

the ith working response[b]. Here each one is $0 \pm \frac{1/2}{1/4} = \pm 2$.

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds
[b] https://morrison-lab.github.io/sds/logistic-inference.html#def-irls

Exercise 6.8 (The first Newton step of the hand-sized fit)**.** Use the setup of Exercise 6.6.
Compute the first Newton iterate[a] $\vec{\beta}^{(1)}$ from the update of Exercise 6.3.

[a] https://morrison-lab.github.io/mds/optimization-newton.html#def-iterate

Solution

Solution 6.8. For the first Newton iterate, the columns of $\mathbf{D}$ are the ones vector and $\underset{\sim}{x} = (-1,-1,-1,1,1,1)^\top$, which are orthogonal[a], so

$$\begin{aligned} \mathbf{D}^\top W^{(0)} \mathbf{D} &= \tfrac{1}{4}\mathbf{D}^\top\mathbf{D} && (W^{(0)} = \tfrac{1}{4}\mathbb{I}) \\ &= \tfrac{1}{4}\begin{pmatrix} 6 & 0 \\ 0 & 6 \end{pmatrix} && \text{(orthogonal columns, each with squared length 6)} \\ &= \tfrac{3}{2}\mathbb{I}, \end{aligned}$$

and Solution 5.2 found $\mathbf{D}^\top\big(\underset{\sim}{y} - \underset{\sim}{\pi}^{(0)}\big) = (0,1)^\top$. So

$$\begin{aligned} \vec{\beta}^{(1)} &= \vec{0} + \big(\tfrac{3}{2}\mathbb{I}\big)^{-1}(0,1)^\top \\ &= \big(0, \tfrac{2}{3}\big)^\top. \end{aligned}$$

[a] https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-orthogonal-vectors

Exercise 6.9 (Newton's method, and why it is a weighted least squares fit (part 5))**.** Use the setup of Exercise 6.1.
Starting from $\vec{\beta}^{(0)} = \vec{0}$, what are β_0, β_1 and the error $|\beta_1 - \log 2|$ after each of four steps of the update of Exercise 6.3 on Exercise 5.1?

Solution

Solution 6.9.

6.0.5 R

```
newton_irls <- function(d, y, steps) {
  beta <- rep(0, ncol(d))
  path <- matrix(NA_real_, nrow = steps, ncol = ncol(d))
  for (t in seq_len(steps)) {
    p <- plogis(drop(d %*% beta))
    w <- p * (1 - p)
    beta <- beta + drop(solve(crossprod(d, w * d), crossprod(d, y - p)))
    path[t, ] <- beta
  }
  path
}

newton_d <- cbind(1, c(-1, -1, -1, 1, 1, 1))
newton_y <- c(0, 0, 1, 0, 1, 1)
newton_path <- newton_irls(newton_d, newton_y, steps = 4)
newton_err <- abs(newton_path[, 2] - log(2))
data.frame(
  step = 1:4,
  b0 = newton_path[, 1],
  b1 = sprintf("%.12f", newton_path[, 2]),
  error = signif(newton_err, 4)
)
```

```
# A tibble: 4 x 4
   step        b0 b1                error
  <int>     <dbl> <chr>             <dbl>
1     1  0        0.666666666667 2.65e- 2
2     2 -1.65e-16 0.693033399004 1.14e- 4
3     3 -1.65e-16 0.693147178402 2.16e- 9
4     4 -8.18e-17 0.693147180560 1.11e-16
```

6.0.6 Python

```
import numpy as np
import pandas as pd

def newton_irls(d, y, steps):
    beta = np.zeros(d.shape[1])
    path = []
    for _ in range(steps):
        p = 1 / (1 + np.exp(-d @ beta))
        w = p * (1 - p)
        beta = beta + np.linalg.solve(d.T @ (w[:, None] * d), d.T @ (y - p))
        path.append(beta)
    return np.array(path)

newton_d = np.column_stack([np.ones(6), [-1, -1, -1, 1, 1, 1]])
newton_y = np.array([0, 0, 1, 0, 1, 1])
newton_path = newton_irls(newton_d, newton_y, steps=4)
print(pd.DataFrame({
    "step": range(1, 5),
    "b0": newton_path[:, 0],
    "b1": [f"{b:.12f}" for b in newton_path[:, 1]],
    "error": np.abs(newton_path[:, 1] - np.log(2)),
}))
```

```
   step            b0              b1         error
0     1  0.000000e+00  0.666666666667  2.648051e-02
1     2 -1.650958e-16  0.693033399004  1.137816e-04
2     3 -2.483499e-16  0.693147178402  2.157462e-09
3     4  1.450327e-18  0.693147180560  1.110223e-16
```

6.0.7 Julia

```
using LinearAlgebra, Printf

function newton_irls(d, y, steps)
    beta = zeros(size(d, 2))
    path = zeros(steps, size(d, 2))
    for t in 1:steps
        p = 1 ./ (1 .+ exp.(-d * beta))
        w = p .* (1 .- p)
        beta = beta + (d' * (w .* d)) \ (d' * (y - p))
        path[t, :] = beta
    end
    return path
end;

newton_d = [ones(6) [-1.0, -1, -1, 1, 1, 1]];
newton_y = [0.0, 0, 1, 0, 1, 1];
newton_path = newton_irls(newton_d, newton_y, 4);
for t in 1:4
    @printf("%d  %g  %.12f  %.4g\n", t, newton_path[t, 1], newton_path[t, 2], abs(newton_path[t, 2] -
end
```

```
1  0  0.666666666667  0.02648
2  -1.65096e-16  0.693033399004  0.0001138
3  -1.65091e-16  0.693147178402  2.157e-09
4  -1.65091e-16  0.693147180560  1.11e-16
```

The intercept[a] stays at 0, up to rounding error. At $\beta_0 = 0$ the intercept component of the gradient is 0 (Solution 5.14), and the controls and the treated patients have the same diagonal[b] entry $\pi(1-\pi)$ of W, because $\text{expit}(-\beta_1) = 1 - \text{expit}(\beta_1)$. With equal entries and orthogonal[c] columns, $\mathbf{D}^\top W \mathbf{D}$ is diagonal, so the intercept component of the step is 0. The error falls from 2.648×10^{-2} after one step to 2.157×10^{-9} after three.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[b] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-diagonal-matrix
[c] https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-orthogonal-vectors

Exercise 6.10 (Newton's method, and why it is a weighted least squares fit (part 6))**.** Use the setup of Exercise 6.1.
Compare each error with the *square* of the one before it.

Solution

Solution 6.10.

```
newton_ratio <- newton_err[-1] / newton_err[-4]^2
data.frame(
  step = 2:4,
  error = newton_err[-1],
  previous_squared = newton_err[-4]^2,
  ratio = newton_ratio
)
```

```
# A tibble: 3 x 4
   step    error previous_squared  ratio
  <int>    <dbl>            <dbl>  <dbl>
1     2 1.14e- 4         7.01e- 4  0.162
2     3 2.16e- 9         1.29e- 8  0.167
3     4 1.11e-16         4.65e-18 23.9
```

After steps 2 and 3, each error is close to a fixed multiple of the square of the error before it: the ratios are 0.162 and 0.167.

Exercise 6.11 (Newton's method, and why it is a weighted least squares fit (part 7))**.** Use the setup of Exercise 6.1.
From the ratios of Exercise 6.10, give the approximate value of the multiplier K in $e^{(t+1)} \approx K(e^{(t)})^2$, where $e^{(t)} = |\beta_1 - \log 2|$ after step t.

Solution

Solution 6.11. From the ratio after step 3, the multiplier is about $K = 0.167$, close to $1/6$.

Exercise 6.12 (Newton's method: the last step)**.** Use the setup of Exercise 6.11.
Explain why the ratio[a] after step 4 in Exercise 6.10 does not fit the pattern $e^{(t+1)} \approx K(e^{(t)})^2$.

[a] https://morrison-lab.github.io/mds/algebra-sums.html#def-ratio

Solution

Solution 6.12. After step 4 the error is 1.1×10^{-16}, no more than a few times $2^{-53} \approx 1.1 \times 10^{-16}$, the spacing between consecutive double-precision numbers in $[\frac{1}{2}, 1)$. At that size the error comes from rounding in the arithmetic and in the stored value of $\log 2$, not from the method, so step 4 cannot

reduce it further.

Exercise 6.13 (Newton's method, and why it is a weighted least squares fit (part 8))**.** Use the setup of Exercise 6.1.
For Newton's method[a] applied to an equation $g(\beta_1) = 0$, the error[b] near the root satisfies $e^{(t+1)} \approx \left|\frac{g''}{2g'}\right|(e^{(t)})^2$, with g' and g'' evaluated at the root. Here $g(\beta_1) = 4 - 6\,\mathrm{expit}(\beta_1)$ is the slope[c] component of the gradient from Exercise 5.15. Derive K as a closed-form expression[d] by evaluating $|g''/2g'|$ at $\hat{\beta}_1 = \log 2$, using $\mathrm{expit}' = \mathrm{expit}(1 - \mathrm{expit})$ and $\mathrm{expit}'' = \mathrm{expit}(1 - \mathrm{expit})(1 - 2\,\mathrm{expit})$.

[a] https://morrison-lab.github.io/mds/optimization-newton.html#def-newton-method
[b] https://morrison-lab.github.io/mds/optimization-newton.html#def-iterate-error
[c] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope
[d] https://morrison-lab.github.io/mcs/expressions-algorithms.html#def-closed-form-expression

Solution

Solution 6.13. Differentiating $g(\beta_1) = 4 - 6\,\mathrm{expit}(\beta_1)$ twice gives

$$g'(\beta_1) = -6\ \mathrm{expit}(1 - \mathrm{expit}), \qquad g''(\beta_1) = -6\ \mathrm{expit}(1 - \mathrm{expit})(1 - 2\,\mathrm{expit}),$$

with $\mathrm{expit} = \mathrm{expit}(\beta_1)$. At $\hat{\beta}_1 = \log 2$, $\mathrm{expit} = \frac{2}{3}$ (Solution 5.6), so $\mathrm{expit}(1-\mathrm{expit}) = \frac{2}{9}$ and $1-2\,\mathrm{expit} = -\frac{1}{3}$. Then $g' = -6 \cdot \frac{2}{9} = -\frac{4}{3}$ and $g'' = -6 \cdot \frac{2}{9} \cdot \left(-\frac{1}{3}\right) = \frac{4}{9}$, so

$$\begin{aligned} K &= \left|\frac{g''}{2g'}\right| \\ &= \frac{4/9}{8/3} \qquad \text{(substitute } g'' \text{ and } g'\text{)} \\ &= \frac{4}{9} \cdot \frac{3}{8} \qquad \text{(divide by multiplying by the reciprocal)} \\ &= \frac{1}{6}, \end{aligned}$$

close to the 0.167 of Solution 6.11. As a check, $\frac{1}{6}e_2^2 = 2.158 \times 10^{-9}$, and the observed e_3 is 2.157×10^{-9}. The factor $1 - 2\,\mathrm{expit}$ in g'' is 0 when $\mathrm{expit} = \frac{1}{2}$. If the fitted probability at the root were $\frac{1}{2}$, K would be 0, and the error would shrink faster than quadratically.

Theorem 6.4 (The Newton error constant of the worked fit)**.** *For $g(\beta_1) = 4 - 6\,\mathrm{expit}(\beta_1)$, the slope[a] score equation[b] of the worked fit,*

$$\left|\frac{g''(\hat{\beta}_1)}{2g'(\hat{\beta}_1)}\right| = \frac{1}{6} \qquad \textit{at the root } \hat{\beta}_1 = \log 2.$$

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope
[b] https://morrison-lab.github.io/sds/intro-MLEs.html#def-score-equation

Proof

Proof. At the root $\mathrm{expit} = \frac{2}{3}$, so $g' = -6\,\mathrm{expit}(1 - \mathrm{expit}) = -\frac{4}{3}$ and $g'' = -6\,\mathrm{expit}(1 - \mathrm{expit})(1 - 2\,\mathrm{expit}) = \frac{4}{9}$, and $\left|\frac{4/9}{2\cdot(-4/3)}\right| = \frac{1}{6}$. □

Exercise 6.14 (Newton's method, and why it is a weighted least squares fit (part 9))**.** Use the setup of Exercise 6.1.

Compare gradient ascent[a] with $\alpha = \frac{1}{2}$ and Newton's method[b] on Exercise 5.1: what is the error $|\beta_1 - \log 2|$ of each after three steps and after five steps?

[a] https://morrison-lab.github.io/mds/optimization-gradient-descent.html#def-gradient-ascent
[b] https://morrison-lab.github.io/mds/optimization-newton.html#def-newton-method

Solution

Solution 6.14.

6.0.8 R

```
gradient_ascent <- function(d, y, rate, steps) {
  beta <- rep(0, ncol(d))
  path <- matrix(NA_real_, nrow = steps, ncol = ncol(d))
  for (t in seq_len(steps)) {
    beta <- beta + rate * drop(crossprod(d, y - plogis(drop(d %*% beta))))
    path[t, ] <- beta
  }
  path
}

ga_path <- gradient_ascent(newton_d, newton_y, rate = 0.5, steps = 5)
ga_err <- abs(ga_path[, 2] - log(2))
newton_err5 <- abs(newton_irls(newton_d, newton_y, steps = 5)[, 2] - log(2))
rbind(gradient_ascent = ga_err, newton = newton_err5)[, c(3, 5)]
```

```
                        [,1]        [,2]
gradient_ascent 1.977644e-02 0.002178238
newton          2.157462e-09 0.000000000
```

6.0.9 Python

```
def gradient_ascent(d, y, rate, steps):
    beta = np.zeros(d.shape[1])
    path = []
    for _ in range(steps):
        beta = beta + rate * d.T @ (y - 1 / (1 + np.exp(-d @ beta)))
        path.append(beta)
    return np.array(path)

ga_err_py = np.abs(gradient_ascent(newton_d, newton_y, 0.5, 5)[:, 1] - np.log(2))
newton_err5_py = np.abs(newton_irls(newton_d, newton_y, 5)[:, 1] - np.log(2))
print(pd.DataFrame({"gradient ascent": ga_err_py, "Newton": newton_err5_py},
                   index=range(1, 6)).loc[[3, 5]])
```

```
   gradient ascent        Newton
3         0.019776  2.157462e-09
5         0.002178  0.000000e+00
```

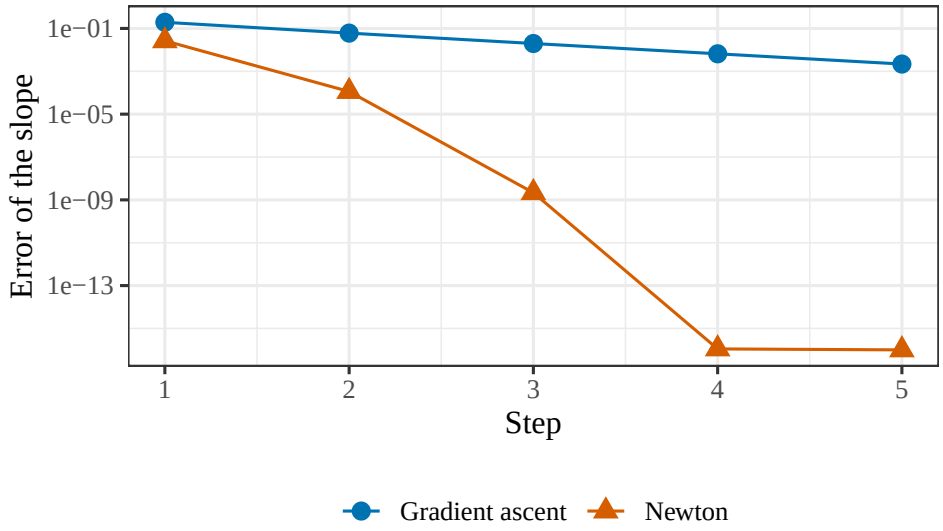
Table 2: How far each method is from the answer after three and after five steps.

	after 3 steps	after 5 steps
Gradient ascent[a], $\alpha = \frac{1}{2}$	2×10^{-2}	2.2×10^{-3}
Newton's method[b]	2.2×10^{-9}	0

Newton's error after five steps is at the level of rounding, as after four (Solution 6.11).
Gradient ascent[c] multiplies the error by about $\frac{1}{3}$ at each step (Solution 5.23), so it gains $\log_{10} 3 \approx 0.48$ correct decimal digits per step. Newton's method roughly squares the error at each step, so the number of correct decimal digits, $-\log_{10} e^{(t)}$, roughly doubles: 1.6, 3.9, 8.7 after steps 1, 2 and 3.
Figure 27 plots both errors at each of the five steps.

6.0.10 R

```
nva_floor <- 1e-16
nva_ga <- pmax(ga_err, nva_floor)
nva_newton <- pmax(newton_err5, nva_floor)
nva_method <- c("Gradient ascent", "Newton")
nva_steps <- 1:5
nva_ga_df <- tibble::tibble(
  step = nva_steps,
  error = nva_ga,
  method = nva_method[1]
)
nva_newton_df <- tibble::tibble(
  step = nva_steps,
  error = nva_newton,
  method = nva_method[2]
)
nva_df <- dplyr::bind_rows(nva_ga_df, nva_newton_df)
nva_colours <- c("Gradient ascent" = "#0072B2", "Newton" = "#D55E00")
nva_shapes <- c("Gradient ascent" = 16, "Newton" = 17)
nva_df |>
  ggplot2::ggplot(
    ggplot2::aes(step, error, colour = method, shape = method)
  ) +
  ggplot2::geom_line() +
  ggplot2::geom_point(size = 3) +
  ggplot2::scale_y_log10() +
  ggplot2::scale_colour_manual(values = nva_colours) +
  ggplot2::scale_shape_manual(values = nva_shapes) +
  ggplot2::labs(
    x = "Step",
    y = "Error of the slope",
    colour = NULL,
    shape = NULL
  )
```

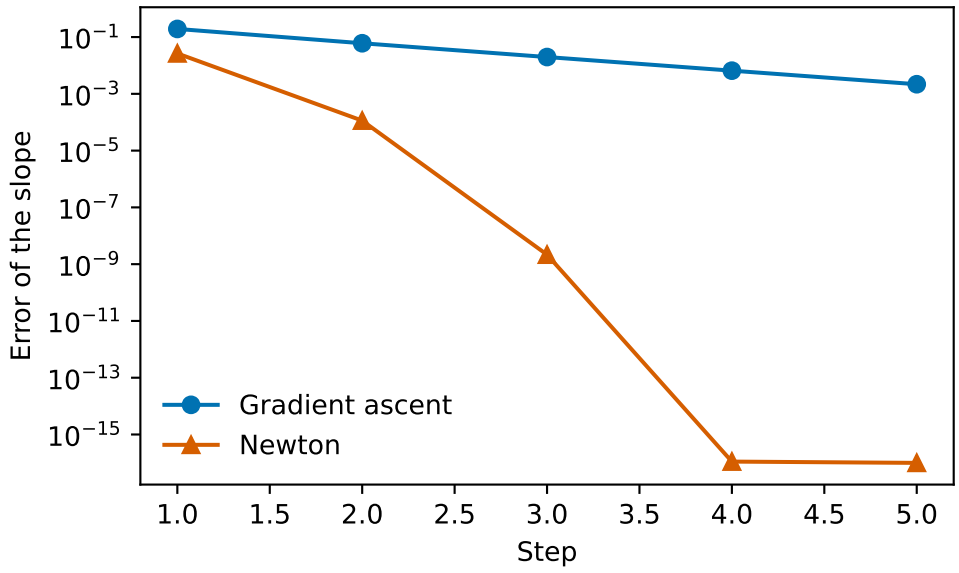


(a)

6.0.11 Python

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(5, 3), layout="constrained")
steps_nva = range(1, 6)
ax.semilogy(steps_nva, np.maximum(ga_err_py, 1e-16), "o-", color="#0072B2", label="Gradient ascent");
ax.semilogy(steps_nva, np.maximum(newton_err5_py, 1e-16), "^-", color="#D55E00", label="Newton");
ax.set_xlabel("Step");
ax.set_ylabel("Error of the slope");
ax.legend(frameon=False);
plt.show()
```



(b)

Figure 27: The error[d] of the slope[e] coefficient at each step of gradient ascent[f] and of Newton's method[g], with errors below 10^{-16} drawn at 10^{-16}.

[c] https://morrison-lab.github.io/mds/optimization-gradient-descent.html#def-gradient-ascent
[d] https://morrison-lab.github.io/mds/optimization-newton.html#def-iterate-error
[e] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope
[f] https://morrison-lab.github.io/mds/optimization-gradient-descent.html#def-gradient-ascent
[g] https://morrison-lab.github.io/mds/optimization-newton.html#def-newton-method

Exercise 6.15 (Newton's method: the cost of one step)**.** Use the setup of Exercise 6.14.
How many arithmetic operations does one step of gradient ascent[a] take, and how many does one step of Newton's method[b] take, for n observations and a design matrix with $p+1$ columns?

[a] https://morrison-lab.github.io/mds/optimization-gradient-descent.html#def-gradient-ascent
[b] https://morrison-lab.github.io/mds/optimization-newton.html#def-newton-method

Solution

Solution 6.15. A gradient step computes $\mathbf{D}\vec{\beta}$ and $\mathbf{D}^\top(y-\underset{\sim}{\pi})$, each $O(np)$. A Newton step also forms $\mathbf{D}^\top W\mathbf{D}$, which is $O(np^2)$, and solves a $(p+1)\times(p+1)$ system of equations, which is $O(p^3)$.

Exercise 6.16 (Newton's method, and why it is a weighted least squares fit (part 10))**.** Use the setup of Exercise 6.1.
Using the operation counts of Exercise 6.15, state when Newton's method[a] needs less computation overall than gradient ascent[b].

[a] https://morrison-lab.github.io/mds/optimization-newton.html#def-newton-method
[b] https://morrison-lab.github.io/mds/optimization-gradient-descent.html#def-gradient-ascent

Solution

Solution 6.16. Dividing the operation count of a Newton step by that of a gradient step gives

$$\frac{np^2+p^3}{np} = p + \frac{p^2}{n},$$

which is at most $2p$ when $n \geq p$. So one Newton step costs on the order of p gradient steps, and Newton's method[a] needs less computation overall when it saves more steps than that. Gradient ascent[b] can need many more steps: the simulated fit of Solution 4.0.29 ran 10,000. Newton's method needs a handful (Exercise 6.9), so for p up to the hundreds Newton's method is usually cheaper, which is one reason R's `glm()` fits by IRLS [12, Ch. 3].

[a] https://morrison-lab.github.io/mds/optimization-newton.html#def-newton-method
[b] https://morrison-lab.github.io/mds/optimization-gradient-descent.html#def-gradient-ascent

Exercise 6.17 (Newton's method, and why it is a weighted least squares fit (part 11))**.** Use the setup of Exercise 6.1.
Neural networks are usually fitted by first-order methods, such as gradient descent[a], rather than by Newton's method[b]. Explain why.

[a] https://morrison-lab.github.io/mds/optimization-gradient-descent.html#def-gradient-descent
[b] https://morrison-lab.github.io/mds/optimization-newton.html#def-newton-method

Solution

Solution 6.17. Newton's method[a] needs the Hessian[b], a matrix with one row and one column for each parameter. A neural network can have millions of parameters, so its Hessian has 10^{12} or more entries, which at 8 bytes each is 8 terabytes or more: too large to store, and far too large to factorize at

$O(\text{parameters}^3)$ operations per step. The chapter on neural networks therefore fits them by first-order methods, which brings back the choice of learning rate that Newton's method does not need.

[a] https://morrison-lab.github.io/mds/optimization-newton.html#def-newton-method
[b] https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian

Exercise 6.18 (Newton's method, and why it is a weighted least squares fit (part 12))**.** Use the setup of Exercise 6.1.
$W^{(t)}$ depends on $\vec{\beta}^{(t)}$ and $\mathbf{D}$ but not on $\underset{\sim}{y}$. What follows about the observed information[a] and the expected (Fisher) information[b]?

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-oinf
[b] https://morrison-lab.github.io/sds/intro-MLEs.html#def-einf

Solution

Solution 6.18. The weight matrix W is computed from the fitted probabilities, which depend on $\vec{\beta}$ and $\mathbf{D}$ alone, so no outcome y_i appears in the Hessian[a] $-\mathbf{D}^\top W\mathbf{D}$. The Hessian is therefore not random: its expectation[b] over $\underset{\sim}{Y}$ is the Hessian itself. The observed information[c] $\mathbf{I} = \mathbf{D}^\top W\mathbf{D}$ and the expected information[d] $\mathcal{I} = \mathbb{E}[\mathbf{D}^\top W\mathbf{D}]$ are the same matrix, at every $\vec{\beta}$ and for every dataset.

[a] https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian
[b] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[c] https://morrison-lab.github.io/sds/intro-MLEs.html#def-oinf
[d] https://morrison-lab.github.io/sds/intro-MLEs.html#def-einf

Exercise 6.19 (Newton's method and Fisher scoring)**.** Use the setup of Exercise 6.18.
What follows for Newton's method[a] and Fisher scoring[b] on logistic regression?

[a] https://morrison-lab.github.io/mds/optimization-newton.html#def-newton-method
[b] https://morrison-lab.github.io/sds/intro-MLEs.html#def-fisher-scoring

Solution

Solution 6.19. Newton's method[a] steps with the observed information[b], and Fisher scoring[c] steps with the expected information[d], so for logistic regression the two algorithms[e] take the same steps. The equality comes from the logit[f] link, which is the canonical link of the Bernoulli[g] family [13], not from binary regression in general. Fitting the same data with a probit link, $\pi_i = \Phi(\underset{\sim}{x}_i \cdot \vec{\beta})$, gives a Hessian[h] that involves the y_i, so there the two information matrices differ and the two algorithms take different steps.

[a] https://morrison-lab.github.io/mds/optimization-newton.html#def-newton-method
[b] https://morrison-lab.github.io/sds/intro-MLEs.html#def-oinf
[c] https://morrison-lab.github.io/sds/intro-MLEs.html#def-fisher-scoring
[d] https://morrison-lab.github.io/sds/intro-MLEs.html#def-einf
[e] https://morrison-lab.github.io/mcs/expressions-algorithms.html#def-algorithm
[f] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds
[g] https://morrison-lab.github.io/pds/random-variables.html#def-bernoulli
[h] https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian

Remark (Fisher information is the curvature of the Kullback–Leibler divergence)

Remark 6.1 (Fisher information is the curvature of the Kullback–Leibler divergence).
The expected (Fisher) information[a] at the true parameter $\vec{\beta}_0$ equals the Hessian[b], with respect to $\vec{\beta}$ and at the true parameter, of the Kullback–Leibler divergence[c] from the true distribution[d] of $\underset{\sim}{Y}$ to the model's distribution with parameter $\vec{\beta}$ (proved in the statistics notes[e]):

$$\nabla_{\vec{\beta}} \nabla_{\vec{\beta}} D_{\mathrm{KL}}\left(\mathbb{p}_{\vec{\beta}_0} \,\|\, \mathbb{p}_{\vec{\beta}}\right)^{\top}\Big|_{\vec{\beta}=\vec{\beta}_0} = \mathcal{I}\left(\vec{\beta}_0\right).$$

The expected cross-entropy loss equals that divergence plus the entropy[f] of the true distribution:

$$\mathbb{E}\left[L\left(\vec{\beta}\right)\right] = D_{\mathrm{KL}}\left(\mathbb{p}_{\vec{\beta}_0} \,\|\, \mathbb{p}_{\vec{\beta}}\right) + \mathbb{H}\left(\mathbb{p}_{\vec{\beta}_0}\right).$$

The entropy does not depend on $\vec{\beta}$, so the expected cross-entropy loss and the divergence have the same Hessian at every $\vec{\beta}$:

$$\nabla_{\vec{\beta}} \nabla_{\vec{\beta}} \mathbb{E}\left[L\left(\vec{\beta}\right)\right]^{\top} = \nabla_{\vec{\beta}} \nabla_{\vec{\beta}} D_{\mathrm{KL}}\left(\mathbb{p}_{\vec{\beta}_0} \,\|\, \mathbb{p}_{\vec{\beta}}\right)^{\top}.$$

For logistic regression, the Hessian of the cross-entropy loss does not involve the outcomes (Solution 6.18), so at every $\vec{\beta}$, not only at the true parameter, it equals the Hessian of the expected cross-entropy loss. That Hessian is the Fisher information:

$$\nabla_{\vec{\beta}} \nabla_{\vec{\beta}} L\left(\vec{\beta}\right)^{\top} = \nabla_{\vec{\beta}} \nabla_{\vec{\beta}} \mathbb{E}\left[L\left(\vec{\beta}\right)\right]^{\top} = \mathcal{I}\left(\vec{\beta}\right) = \mathbf{D}^{\top} W \mathbf{D}.$$

Fisher scoring[g] therefore scales each step by the curvature of the expected cross-entropy loss at the current iterate, whose sample[h] version logistic regression minimizes:

$$\vec{\beta}^{(t+1)} = \vec{\beta}^{(t)} - \left(\mathcal{I}\left(\vec{\beta}^{(t)}\right)\right)^{-1} \nabla_{\vec{\beta}} L\left(\vec{\beta}^{(t)}\right).$$

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-einf
[b] https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian
[c] https://morrison-lab.github.io/pds/expectation.html#def-kl-divergence
[d] https://morrison-lab.github.io/pds/random-variables.html#def-cdf
[e] https://morrison-lab.github.io/sds/intro-MLEs.html#thm-fisher-kl-curvature
[f] https://morrison-lab.github.io/pds/expectation.html#def-entropy
[g] https://morrison-lab.github.io/sds/intro-MLEs.html#def-fisher-scoring
[h] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample

Watch

Andrew Ng fits logistic regression by Newton's method[a], from 1:05:57[b] [14], Lecture 3.

[a] https://morrison-lab.github.io/mds/optimization-newton.html#def-newton-method
[b] https://www.youtube.com/watch?v=het9HFqo1TQ&t=3957s

Read

GeeksforGeeks explains Newton's method[a] [15].

[a] https://www.geeksforgeeks.org/machine-learning/newtons-method-in-machine-learning/

[1] Wikipedia contributors, "Cross-entropy — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Cross-entropy

[2] V. Guruswami and P. Raghavendra, "Hardness of learning halfspaces with noise," *SIAM Journal on Computing*, vol. 39, no. 2, pp. 742–765, 2009, doi: 10.1137/070685798[36].

[3] GeeksforGeeks, "What is cross-entropy loss function?" GeeksforGeeks, tutorial article, 2025. Available: https://www.geeksforgeeks.org/machine-learning/what-is-cross-entropy-loss-function/

[4] 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.x[37].

[5] K. Weinberger, "CS4780: Machine learning for intelligent systems, spring 2017." Cornell University, lecture videos on YouTube, 2017. Available: https://www.youtube.com/playlist?list=PLl8OlHZGYOQ7bkVbuRthEsaLr7bONzbXS

[36] https://doi.org/10.1137/070685798
[37] https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

[6] J. Starmer, "Logistic regression details, part 2: Maximum likelihood." StatQuest, video on YouTube, 2018. Available: https://www.youtube.com/watch?v=BfKanl1aSG0

[7] D. M. Rocke, "EPI 204 lecture notes, Spring 2021." University of California, Davis, 2021. Available: https://dmrocke.ucdavis.edu/Class/EPI204-Spring-2021/EPI204-Spring-2021.html

[8] J. Starmer, "Neural networks part 6: Cross entropy." StatQuest, video on YouTube, 2021. Available: https://www.youtube.com/watch?v=6ArSys5qHAU

[9] 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/

[10] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011, Available: https://jmlr.org/papers/v12/pedregosa11a.html

[11] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge University Press, 2004. doi: 10.1017/CBO9780511804441[38].

[12] S. N. Wood, *Generalized additive models: An introduction with R*, 2nd ed. Chapman; Hall/CRC, 2017. doi: 10.1201/9781315370279[39].

[13] J. A. Nelder and R. W. M. Wedderburn, "Generalized linear models," *Journal of the Royal Statistical Society: Series A (General)*, vol. 135, no. 3, pp. 370–384, 1972, doi: 10.2307/2344614[40].

[14] A. Ng *et al.*, "CS229: Machine learning, autumn 2018." Stanford Online, lecture videos on YouTube, 2018. Available: https://www.youtube.com/playlist?list=PLoROMvodv4rMiGQp3WXShtMGgzqpfVfbU

[15] GeeksforGeeks, "Newton's method in machine learning." GeeksforGeeks, tutorial article, 2025. Available: https://www.geeksforgeeks.org/machine-learning/newtons-method-in-machine-learning/

[38] https://doi.org/10.1017/CBO9780511804441
[39] https://doi.org/10.1201/9781315370279
[40] https://doi.org/10.2307/2344614