Multiclass Logistic Regression

Last modified: 2026-11-19

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

Source

Adapted from Brian Hutchinson's Fall 2025 lecture notes on multinomial logistic regression, reorganized as exercises, and drawing on Logan Sizemore's Spring 2025 lectures for the redundant weight vector. This material was a section of the logistic regression page. Solution 5.1 corrects the spurious $1/n$ in the source's multinomial gradient and flags the correction rather than fixing it silently.

1 Regressing a class code

Exercise 1.1 (What a numeric class code assumes)**.** The Auto data record where each car was made as `origin`, coded 1 for American, 2 for European and 3 for Japanese. A label with more than two classes[a] could be predicted by linear regression on such a code: fit a least-squares line to the code, and assign each car the class whose code is nearest its fitted value[b]. That approach reuses a method we already have, so it is worth knowing whether it is sound.

Hint

Compare the distances between the codes with what the three classes[c] are.

State what this coding assumes about the three origins.

[a] https://morrison-lab.github.io/sds/classification.html#def-class
[b] https://morrison-lab.github.io/sds/prediction.html#def-fitted-value
[c] https://morrison-lab.github.io/sds/classification.html#def-class

Solution

Solution 1.1. The coding of `origin` in the Auto data assumes two facts about the origins that the data do not supply:

- **An order.** European lies between American and Japanese, so a car whose fitted value[a] moves from 1 toward 3 passes through European on the way.
- **Even spacing.** The step from American to European is the same size as the step from European to Japanese.

The origins are names with no order, so neither assumption is justified. Any of the $3! = 6$ assignments of the codes $1, 2, 3$ to the origins is as defensible as the one in the file, and a different assignment gives a different line and different predictions [1, Sec. 4.2].

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

Exercise 1.2 (Regressing the origin code on fuel economy)**.** Use the Auto data and the origin code of Exercise 1.1, with fuel economy `mpg` as the only feature. Fit a least-squares line of the code on `mpg`, in R or Python, and round each fitted value[a] to the nearest of $1, 2, 3$ to get a predicted origin.

Hint

Fit models with tidymodels shows `linear_reg()` in R, and Fit and inspect a model with statsmodels shows `sm.OLS` in Python. Recode, count and combine data frames shows how to tally the values of a variable.

Report how many cars the line predicts to be of each origin.

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

Solution

Solution 1.2.

1.0.1 R

```
# Read the Auto data; `origin` holds the codes 1, 2 and 3.
cc_auto <- readr::read_csv("data/auto.csv", show_col_types = FALSE)

# Fit a least-squares line to the code, treating the code as a number.
cc_fit <- parsnip::linear_reg() |>
  parsnip::fit(origin ~ mpg, data = cc_auto)

# Round each fitted value `.pred` to the nearest code,
# then move a value below 1 or above 3 to the nearest end.
cc_aug <- cc_fit |>
  parsnip::augment(new_data = cc_auto) |>
  dplyr::mutate(
    pred = round(.pred),
    pred = pmax(pred, 1),
    pred = pmin(pred, 3)
  )

# Count the cars predicted to be of each origin.
cc_aug |> dplyr::count(pred)
```

```
# A tibble: 3 x 2
   pred     n
  <dbl> <int>
1     1   193
2     2   190
3     3     9
```

1.0.2 Python

```
import pandas as pd
import statsmodels.api as sm

# Read the Auto data; `origin` holds the codes 1, 2 and 3.
cc_auto = pd.read_csv("data/auto.csv")

# Fit a least-squares line to the code, treating the code as a number.
cc_X = sm.add_constant(cc_auto["mpg"])
cc_fit = sm.OLS(cc_auto["origin"], cc_X).fit()

# Round each fitted value to the nearest code,
# then move a value below 1 or above 3 to the nearest end.
cc_pred = cc_fit.fittedvalues.round().clip(1, 3)

# Count the cars predicted to be of each origin.
print(cc_pred.value_counts().sort_index())
```

```
1.0    193
2.0    190
3.0      9
Name: count, dtype: int64
```

The line predicts 193 cars to be American, 190 European and 9 Japanese, while the data hold only 68 European cars. European is coded between the other two origins, so the line predicts European for every car whose fitted value[a] is nearer 2 than 1 or 3.

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

Exercise 1.3 (Changing the class code)**.** Use the Auto data and the line of Exercise 1.2, but recode `origin` as 1 for European, 2 for American and 3 for Japanese. Refit the least-squares line to the new code, round its fitted values[a] as before, and translate each predicted code back to an origin.

Hint

Recode, count and combine data frames shows how to recode values.

Report how many cars get a different predicted origin from the one in Solution 1.0.2.

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

Solution

Solution 1.3. Swapping the codes 1 and 2 twice gives back the original codes, so one lookup both recodes the origins and translates the predicted codes back.

1.0.3 R

```r
# Swap the codes of American (1) and European (2); Japanese stays 3.
cc_swap <- c(2, 1, 3)
cc_auto2 <- cc_auto |>
  dplyr::mutate(code = cc_swap[origin])

# Refit the line to the new code, and round as before.
cc_fit2 <- parsnip::linear_reg() |>
  parsnip::fit(code ~ mpg, data = cc_auto2)
cc_aug2 <- cc_fit2 |>
  parsnip::augment(new_data = cc_auto2) |>
  dplyr::mutate(
    code_pred = round(.pred),
    code_pred = pmax(code_pred, 1),
    code_pred = pmin(code_pred, 3),
    # The same swap translates a predicted code back to an origin.
    pred = cc_swap[code_pred]
  )

# Count the predicted origins, and the cars whose prediction changed.
cc_aug2 |> dplyr::count(pred)
```

```
# A tibble: 1 x 2
   pred     n
  <dbl> <int>
1     1   392
```

```r
cc_changed <- sum(cc_aug2$pred != cc_aug$pred)
cc_changed
```

```
[1] 199
```

1.0.4 Python

```python
# Swap the codes of American (1) and European (2); Japanese stays 3.
cc_swap = {1: 2, 2: 1, 3: 3}
cc_code = cc_auto["origin"].map(cc_swap)

# Refit the line to the new code, and round as before.
cc_fit2 = sm.OLS(cc_code, cc_X).fit()
cc_code_pred = cc_fit2.fittedvalues.round().clip(1, 3)

# The same swap translates a predicted code back to an origin.
cc_pred2 = cc_code_pred.map(cc_swap)

# Count the predicted origins, and the cars whose prediction changed.
print(cc_pred2.value_counts().sort_index())
```

```
1    392
Name: count, dtype: int64
```

```python
print((cc_pred2 != cc_pred).sum())
```

```
199
```

The predicted origin changes for 199 of the 392 cars, although the data and the feature are the same and only the codes differ. With American now coded between the other two origins, the line predicts American for 392 of the 392 cars. A model for a label with unordered classes[a] should treat the classes symmetrically. The rest of this page builds one that does.

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

2 The multinomial model

A logistic regression model has one coefficient vector, for a label with two classes[1]. This section gives each of several classes its own coefficient vector and normalizes the exponentiated linear predictors into class[2] probabilities. With two classes, that model reduces to logistic regression, and it is the output layer of multi-class neural network classifiers in the neural networks chapter.

Exercise 2.1 (More than two classes)**.** Logistic regression gives the label two classes[a] and one coefficient vector. For C classes, give each class[b] m its own coefficient vector B_m and set

$$P(Y = m \mid \underset{\sim}{x}) = \frac{\exp\{{B_m}^\top \underset{\sim}{x}\}}{\sum_k \exp\{{B_k}^\top \underset{\sim}{x}\}},$$

where the sum runs over all C classes.
Set $C = 2$, label the two classes 0 and 1 as in logistic regression, and show that this expression[c] reduces to $P(Y = 1 \mid \underset{\sim}{x}) = \operatorname{expit}\left(\vec{\beta} \cdot \underset{\sim}{x}\right)$. Express $\vec{\beta}$ in terms of B_0 and B_1.

[a] https://morrison-lab.github.io/sds/classification.html#def-class
[b] https://morrison-lab.github.io/sds/classification.html#def-class
[c] https://morrison-lab.github.io/mcs/expressions-algorithms.html#def-expression

Solution

Solution 2.1. With two classes[a] the sum in the denominator has two terms. Divide the numerator and the denominator by $e^{{B_1}^\top \underset{\sim}{x}}$:

$$\begin{aligned}
P(Y = 1 \mid \underset{\sim}{x}) &= \frac{e^{{B_1}^\top \underset{\sim}{x}}}{e^{{B_0}^\top \underset{\sim}{x}} + e^{{B_1}^\top \underset{\sim}{x}}} && \text{(the model with } C = 2\text{)} \\
&= \frac{e^{{B_1}^\top \underset{\sim}{x}} / e^{{B_1}^\top \underset{\sim}{x}}}{\left(e^{{B_0}^\top \underset{\sim}{x}} + e^{{B_1}^\top \underset{\sim}{x}}\right) / e^{{B_1}^\top \underset{\sim}{x}}} && \text{(divide numerator and denominator)} \\
&= \frac{1}{\left(e^{{B_0}^\top \underset{\sim}{x}} + e^{{B_1}^\top \underset{\sim}{x}}\right) / e^{{B_1}^\top \underset{\sim}{x}}} && \text{(a number divided by itself is 1)} \\
&= \frac{1}{e^{{B_0}^\top \underset{\sim}{x}} / e^{{B_1}^\top \underset{\sim}{x}} + 1} && \text{(divide each term of the sum)} \\
&= \frac{1}{e^{{B_0}^\top \underset{\sim}{x} - {B_1}^\top \underset{\sim}{x}} + 1} && \text{(quotient of exponentials)} \\
&= \frac{1}{e^{-({B_1}^\top \underset{\sim}{x} - {B_0}^\top \underset{\sim}{x})} + 1} && \text{(factor out } -1\text{)} \\
&= \frac{1}{e^{-(B_1 - B_0)^\top \underset{\sim}{x}} + 1} && \text{(factor out } \underset{\sim}{x}\text{)} \\
&= \frac{1}{1 + e^{-(B_1 - B_0)^\top \underset{\sim}{x}}} && \text{(reorder the sum)} \\
&= \operatorname{expit}\left((B_1 - B_0)^\top \underset{\sim}{x}\right) && \text{(definition of the logistic function).}
\end{aligned}$$

So $\vec{\beta} = B_1 - B_0$: the single coefficient vector of the binary model is the *difference* between the two class[b] coefficient vectors, not either vector alone.
This difference is why every logistic regression coefficient compares the two classes: by the odds-ratio corollary, e^{β_j} is an odds ratio[c], and the odds[d] of class 1 are its probability divided by the probability of class 0.

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

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

[c] https://morrison-lab.github.io/sds/categorical-tests.html#def-odds-ratio
[d] https://morrison-lab.github.io/pds/probability-basics.html#def-odds

Definition 2.1 (Softmax function)**.** The **softmax function** maps a vector of scores $\underset{\sim}{z} = (z_1, \dots, z_C)^\top$ in $\mathbb{R}^C$ to the vector $\text{softmax}(\underset{\sim}{z})$ in $\mathbb{R}^C$ whose mth entry is

$$\text{softmax}(\underset{\sim}{z})_m = \frac{\mathrm{e}^{z_m}}{\sum_{k=1}^{C} \mathrm{e}^{z_k}}, \qquad m = 1, \dots, C \tag{1}$$

[2, Sec. 6.2.2.3], [3, Sec. 4.2]. Each entry is positive, and the entries sum to 1, so $\text{softmax}(\underset{\sim}{z})$ is a probability mass function[a] on the C classes[b]. The denominator is the normalizer, written $Z = \sum_{k=1}^{C} \mathrm{e}^{z_k}$.

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

Example 2.1 (The softmax of three scores)**.** Apply the softmax function to the scores $\underset{\sim}{z} = (1, 2, 3)^\top$.

2.0.1 R

```
sm_z <- c(1, 2, 3)
sm_exp <- exp(sm_z)
sm_norm <- sum(sm_exp)
sm_prob <- sm_exp / sm_norm
# nolint start: undesirable_function_linter. A matrix.
rbind(exponential = sm_exp, softmax = sm_prob) |> round(3)
```

```
             [,1]  [,2]   [,3]
exponential 2.718 7.389 20.086
softmax     0.090 0.245  0.665
```

```
# nolint end
```

2.0.2 Python

```
import numpy as np

z_sm = np.array([1.0, 2.0, 3.0])
exp_sm = np.exp(z_sm)
print(np.vstack([exp_sm, exp_sm / exp_sm.sum()]).round(3))
```

```
[[ 2.718  7.389 20.086]
 [ 0.09   0.245  0.665]]
```

The exponentials are 2.718, 7.389 and 20.086, and their sum is $Z = 30.193$. Dividing each exponential by Z gives the probabilities 0.090, 0.245 and 0.665, which sum to 1. The scores are equally spaced, but each probability is $e \approx 2.718$ times the one before it.

Read

GeeksforGeeks explains the softmax function[a] [4].

[a] https://www.geeksforgeeks.org/deep-learning/the-role-of-softmax-in-neural-networks-detailed-explanation-and-applications/

Definition 2.2 (Multinomial logistic regression (softmax regression))**.** For a label Y with C classes[a] and a feature vector $\underset{\sim}{x}$, a **multinomial logistic regression** model, also called **softmax regression**,

gives each class[b] m a coefficient vector B_m, collects them as the columns of a matrix B, and sets the vector of class probabilities to the softmax of the linear predictors $B^\top \underset{\sim}{x}$:

$$P(Y = m \mid \underset{\sim}{x}) = \text{softmax}(B^\top \underset{\sim}{x})_m, \qquad m = 1, \ldots, C \tag{2}$$

[3, Sec. 4.3.4], [5, Sec. 4.4]. The linear predictor of class m is $z_m = {B_m}^\top \underset{\sim}{x}$. With two classes, Exercise 2.1 labels them 0 and 1 rather than 1 and 2. That is the labeling of logistic regression.

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

Example 2.2 (Class probabilities from three coefficient vectors)**.** Let there be $C = 3$ classes[a] and one feature x, so that $\underset{\sim}{x} = (1, x)^\top$, with the intercept[b] first. Let the coefficient vectors be $B_1 = (0, 0)^\top$, $B_2 = (1, 0.5)^\top$ and $B_3 = (-1, 2)^\top$, and compute the class[c] probabilities at $x = 1$.

2.0.3 R

```
mlr_entries <- c(0, 0, 1, 0.5, -1, 2)
mlr_b <- matrix(mlr_entries, nrow = 2)
mlr_x <- c(1, 1)
mlr_linpred <- drop(t(mlr_b) %*% mlr_x)
mlr_exp <- exp(mlr_linpred)
mlr_prob <- mlr_exp / sum(mlr_exp)
# nolint start: undesirable_function_linter. A matrix.
rbind(linear_predictor = mlr_linpred, probability = mlr_prob) |> round(3)
```

```
                  [,1]  [,2]  [,3]
linear_predictor 0.000 1.500 1.000
probability      0.122 0.547 0.331
```

```
# nolint end
```

```r
mlr_x_grid <- seq(from = -3, to = 3, length.out = 200)
mlr_ones <- rep(1, times = 200)
# nolint start: undesirable_function_linter. A matrix.
mlr_design <- cbind(mlr_ones, mlr_x_grid)
# nolint end
mlr_grid_linpred <- mlr_design %*% mlr_b
mlr_grid_exp <- exp(mlr_grid_linpred)
mlr_grid_prob <- mlr_grid_exp / rowSums(mlr_grid_exp)
mlr_prob_long <- as.vector(mlr_grid_prob)
mlr_classes <- c("1", "2", "3")
mlr_x_rep <- rep(mlr_x_grid, times = 3)
mlr_class_rep <- rep(mlr_classes, each = 200)
mlr_curves <- tibble::tibble(
  x = mlr_x_rep,
  class = mlr_class_rep,
  probability = mlr_prob_long
)
mlr_marked <- tibble::tibble(
  x = 1,
  class = mlr_classes,
  probability = mlr_prob
)
mlr_curves |>
  ggplot2::ggplot() +
  ggplot2::aes(x = x, y = probability, colour = class) +
  ggplot2::geom_line() +
  ggplot2::geom_point(data = mlr_marked, size = 3) +
  ggplot2::labs(x = "Feature x", y = "Probability", colour = "Class")
```

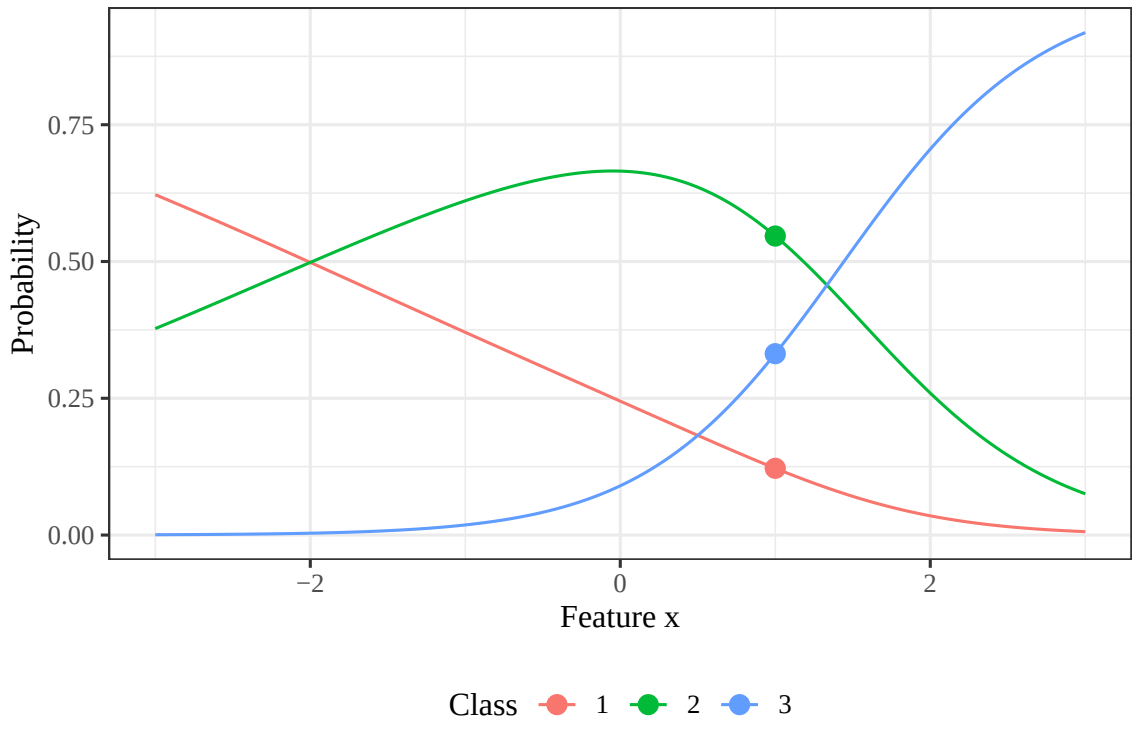


Figure 1

2.0.4 Python

```
import numpy as np

b_mlr = np.array([[0.0, 1.0, -1.0], [0.0, 0.5, 2.0]])
x_mlr = np.array([1.0, 1.0])
linpred_mlr = b_mlr.T @ x_mlr
print(np.vstack([linpred_mlr, np.exp(linpred_mlr) / np.exp(linpred_mlr).sum()]).round(3))
```

```
[[0.    1.5   1.   ]
 [0.122 0.547 0.331]]
```

```
import pandas as pd
import plotnine as p9

x_grid = np.linspace(-3, 3, 200)
design = np.column_stack([np.ones_like(x_grid), x_grid])
grid_exp = np.exp(design @ b_mlr)
grid_prob = grid_exp / grid_exp.sum(axis=1, keepdims=True)
curves = pd.DataFrame({
    "x": np.tile(x_grid, 3),
    "class": np.repeat(["1", "2", "3"], 200),
    "probability": grid_prob.T.ravel(),
})
marked = pd.DataFrame({
    "x": 1.0,
    "class": ["1", "2", "3"],
    "probability": np.exp(linpred_mlr) / np.exp(linpred_mlr).sum(),
})
mlr_plot = (
    p9.ggplot(data=curves, mapping=p9.aes(x="x", y="probability", colour="class"))
    + p9.geom_line()
    + p9.geom_point(data=marked, size=3)
    + p9.labs(x="Feature x", y="Probability", colour="Class")
    + p9.theme_bw()
)
mlr_plot.show();
```

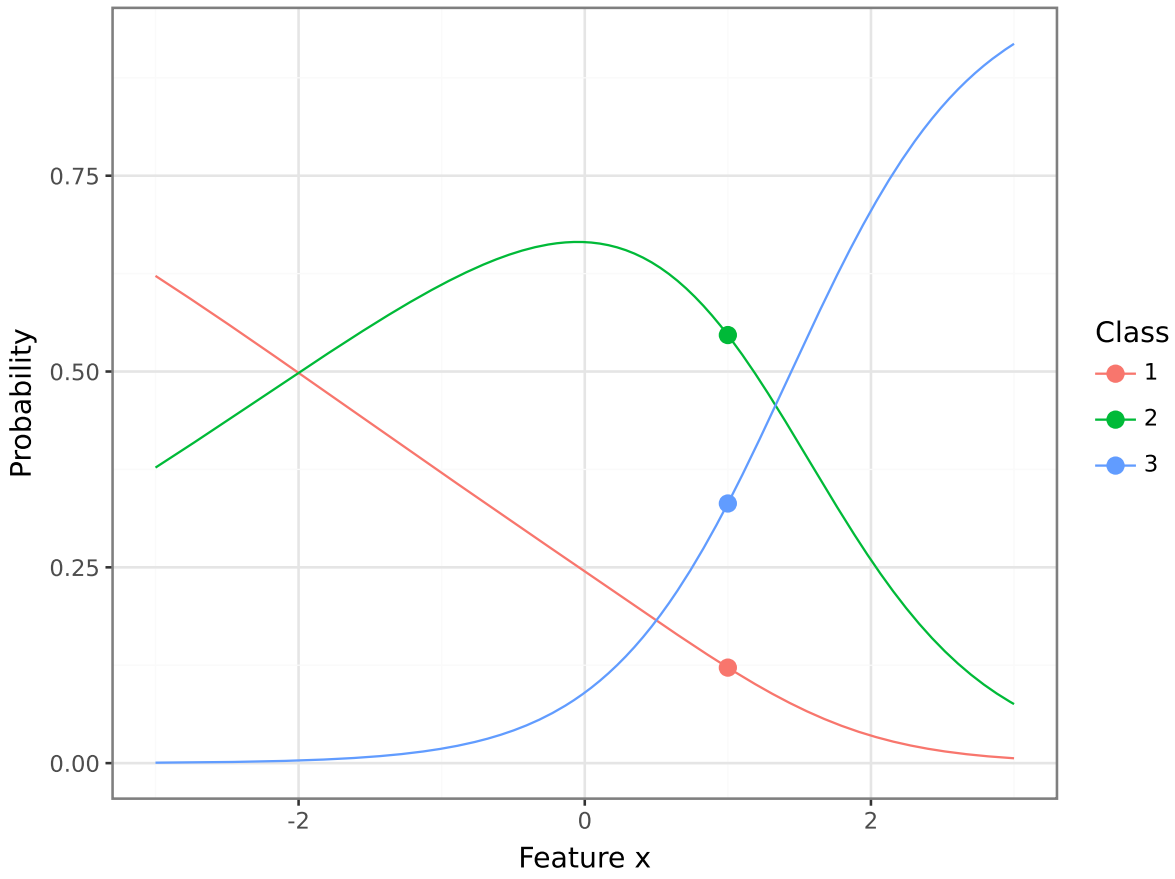


Figure 2

The linear predictors $B^\top \underset{\sim}{x}$ are 0, 1.5 and 1, and their softmax gives the class[d] probabilities 0.122, 0.547 and 0.331. Class 2 has the highest linear predictor, so it has the highest probability.

[a] https://morrison-lab.github.io/sds/classification.html#def-class
[b] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[c] https://morrison-lab.github.io/sds/classification.html#def-class
[d] https://morrison-lab.github.io/sds/classification.html#def-class

Theorem 2.1 (The two-class multinomial model is logistic regression)**.** *In the multinomial model with $C = 2$ and classes[a] labeled 0 and 1,*

$$\Pr(Y = 1 \mid \underset{\sim}{x}) = \operatorname{expit}((B_1 - B_0) \cdot \underset{\sim}{x}),$$

so it is a logistic regression model with coefficient vector $\vec{\beta} = B_1 - B_0$.

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

Proof

Proof. Dividing the numerator[a] and the denominator[b] of the model by $\mathrm{e}^{{B_1}^\top \underset{\sim}{x}}$ gives

$$\Pr(Y = 1 \mid \underset{\sim}{x}) = \frac{1}{1 + \mathrm{e}^{-(B_1 - B_0) \cdot \underset{\sim}{x}}},$$

which is the logistic function[c] of $(B_1 - B_0) \cdot \underset{\sim}{x}$. □

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

Exercise 2.2 (More than two classes (part 2))**.** Use the setup of Exercise 2.1. By Theorem 2.1, the case $C = 2$ is logistic regression. Here C is general.
Show that replacing every B_m by $B_m + $, for any fixed vector , leaves all C probabilities unchanged. How many free parameters does the model have? Give two ways to remove the excess, and state which one Solution 2.1 used.

Solution

Solution 2.2. Adding to each B_k multiplies the numerator and every term of the denominator by the same factor $\mathrm{e}^{\tilde{x}^\top}$, which cancels:

$$\begin{aligned}
\frac{\mathrm{e}^{(B_m+)^\top \tilde{x}}}{\sum_k \mathrm{e}^{(B_k+)^\top \tilde{x}}} &= \frac{\mathrm{e}^{\tilde{x}^\top + B_m{}^\top \tilde{x}}}{\sum_k \mathrm{e}^{\tilde{x}^\top + B_k{}^\top \tilde{x}}} && \text{(distribute the dot product over the sum)} \\
&= \frac{\mathrm{e}^{\tilde{x}^\top} \mathrm{e}^{B_m{}^\top \tilde{x}}}{\sum_k \mathrm{e}^{\tilde{x}^\top} \mathrm{e}^{B_k{}^\top \tilde{x}}} && \text{(exponential of a sum)} \\
&= \frac{\mathrm{e}^{\tilde{x}^\top} \mathrm{e}^{B_m{}^\top \tilde{x}}}{\mathrm{e}^{\tilde{x}^\top} \sum_k \mathrm{e}^{B_k{}^\top \tilde{x}}} && \text{(factor the common term out of the sum)} \\
&= \frac{\mathrm{e}^{B_m{}^\top \tilde{x}}}{\sum_k \mathrm{e}^{B_k{}^\top \tilde{x}}} && \text{(cancel the common factor } \mathrm{e}^{\tilde{x}^\top} > 0\text{)} \\
&= P(Y = m \mid \tilde{x}) && \text{(the model).}
\end{aligned}$$

The model is therefore *overparameterized*. With $\tilde{x} \in \mathbb{R}^{p+1}$, it has $C(p+1)$ parameters, but only $(C-1)(p+1)$ of them are identified: adding a common vector to all C columns moves B within a $(p+1)$-dimensional set of matrices on which the likelihood[a] is constant[b]. The Hessian[c] of the log-likelihood[d] is singular in those directions, so an unregularized Newton solver cannot invert it without a constraint.
Two standard ways to remove this non-identifiability are:

- **Designate a reference class:** set $B_1 = \vec{0}$ and interpret every other B_m relative to class 1. Solution 2.1 used this convention for $C = 2$ by setting $B_0 = \vec{0}$, so that $\vec{\beta} = B_1$.
- **Add a ridge penalty.** Subtracting $\frac{\lambda}{2} \sum_m \|B_m\|^2$ from the log-likelihood[e], for some $\lambda > 0$, makes the objective strictly concave, so it has at most one maximizer. Among the matrices $B + 1^\top$ that give the same probabilities, the penalty is smallest when the C columns sum to $\vec{0}$, so the maximizer satisfies $\sum_m B_m = \vec{0}$. Neural network classifiers typically retain all C columns and rely on weight decay or on first-order optimization, which does not invert the Hessian.

Retaining all C columns treats all classes[f] symmetrically, uses the same gradient formula for every class, and stores the parameters in a single $(p+1) \times C$ matrix, with $p+1$ redundant parameters.
The two approaches differ in what they do to the fitted probabilities. A reference class changes no fitted probability: it selects one matrix B among those that give the same probabilities, which is why different software packages can report different coefficients for the same fitted model. A ridge penalty also shrinks the coefficients toward $\vec{0}$, so it changes the fitted probabilities, by less the smaller λ is.

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik
[b] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function
[c] https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian
[d] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik
[e] https://morrison-lab.github.io/sds/intro-MLEs.html#def-loglik
[f] https://morrison-lab.github.io/sds/classification.html#def-class

3 Ordered classes

When the classes[3] have a natural order, such as ratings from poor to excellent, the multinomial model discards that order. It also needs a separate coefficient vector for each class. Modeling the cumulative probabilities instead uses one coefficient vector and a threshold for each boundary between classes.

Definition 3.1 (Ordinal logistic regression (cumulative logit model, proportional odds model))**.** For a label Y with C ordered classes[a] $1 < \cdots < C$ and a feature vector x with no entry for the intercept[b], an **ordinal logistic regression** model, also called a **cumulative logit model** or **proportional odds model**, has a coefficient vector $\vec{\beta}$ and thresholds $\theta_1 < \cdots < \theta_{C-1}$ that satisfy

$$\operatorname{logit}\{\Pr(Y \leq m \mid \underset{\sim}{x})\} = \theta_m - \vec{\beta} \cdot \underset{\sim}{x}, \qquad m = 1, \ldots, C-1 \tag{3}$$

[6].

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

Example 3.1 (Class probabilities from ordered thresholds)**.** Let there be $C = 3$ ordered classes[a], and one feature x. Let $\theta_1 = -1$, $\theta_2 = 1$ and $\beta_x = 1$. Compute the class[b] probabilities[c] at $x = 1$.

3.0.1 R

```
ord_theta <- c(-1, 1)
ord_beta <- 1
ord_x <- 1
ord_cumulative <- plogis(ord_theta - ord_beta * ord_x)
ord_cumulative_full <- c(0, ord_cumulative, 1)
ord_prob <- diff(ord_cumulative_full)
ord_cumulative_top <- c(ord_cumulative, 1)
# nolint start: undesirable_function_linter. A two-row matrix.
rbind(cumulative = ord_cumulative_top, probability = ord_prob) |> round(3)
```

```
             [,1]  [,2] [,3]
cumulative  0.119 0.500  1.0
probability 0.119 0.381  0.5
```

```
# nolint end
```

3.0.2 Python

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

theta_ord = np.array([-1.0, 1.0])
beta_ord = 1.0
x_ord = 1.0
cumulative_ord = expit(theta_ord - beta_ord * x_ord)
cumulative_full_ord = np.concatenate([[0.0], cumulative_ord, [1.0]])
print(np.vstack([cumulative_full_ord[1:], np.diff(cumulative_full_ord)]).round(3))
```

```
[[0.119 0.5   1.   ]
 [0.119 0.381 0.5  ]]
```

The cumulative probabilities are 0.119, 0.5 and 1. The class probabilities are their consecutive differences: 0.119, 0.381 and 0.5. Changing a threshold moves one boundary between classes. Changing the coefficient moves both boundaries together.

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

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

Exercise 3.1 (Probabilities of ordered classes)**.** Use the model with C ordered classes[a]. Write $\Pr(Y = m \mid \underset{\sim}{x})$ for $m = 1, \ldots, C$ using the sigmoid function[b].

[a] https://morrison-lab.github.io/sds/classification.html#def-class
[b] https://morrison-lab.github.io/mcs/functions.html#def-logistic-function

Solution

Solution 3.1. The event[a] $Y = m$ is $Y \leq m$ without $Y \leq m-1$. Its probability[b] is therefore the difference of two consecutive cumulative probabilities. Define $\theta_0 = -\infty$ and $\theta_C = \infty$, so that $\operatorname{expit}\left(\theta_0 - \vec{\beta} \cdot \underset{\sim}{x}\right) = 0$ and $\operatorname{expit}\left(\theta_C - \vec{\beta} \cdot \underset{\sim}{x}\right) = 1$. Then

$$\Pr(Y = m \mid \underset{\sim}{x}) = \operatorname{expit}\left(\theta_m - \vec{\beta} \cdot \underset{\sim}{x}\right) - \operatorname{expit}\left(\theta_{m-1} - \vec{\beta} \cdot \underset{\sim}{x}\right), \qquad m = 1, \ldots, C.$$

The probabilities are nonnegative because the sigmoid function[c] is increasing and $\theta_{m-1} < \theta_m$.

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-event
[b] https://morrison-lab.github.io/pds/probability-basics.html#def-probability
[c] https://morrison-lab.github.io/mcs/functions.html#def-logistic-function

Exercise 3.2 (Shifting the features (part 2))**.** Use the model. Take two feature vectors $\underset{\sim}{x}$ and $\underset{\sim}{x}'$. Compare the log-odds[a] of $Y \leq m$ at the two, for each m.

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

Solution

Solution 3.2. The log-odds[a] function[b] is the inverse[c] of the sigmoid function[d], so by Equation 3

$$\operatorname{logit}\{\Pr(Y \leq m \mid \underset{\sim}{x})\} = \theta_m - \vec{\beta} \cdot \underset{\sim}{x}.$$

Subtracting the same expression[e] at $\underset{\sim}{x}'$ gives

$$\operatorname{logit}\{\Pr(Y \leq m \mid \underset{\sim}{x})\} - \operatorname{logit}\{\Pr(Y \leq m \mid \underset{\sim}{x}')\} = -\vec{\beta} \cdot (\underset{\sim}{x} - \underset{\sim}{x}'),$$

which does not depend on m. The odds[f] of $Y \leq m$ are multiplied by the same factor $\mathrm{e}^{-\vec{\beta} \cdot (\underset{\sim}{x} - \underset{\sim}{x}')}$ for every m. That shared factor is why the model is called the proportional odds model.

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds
[b] https://morrison-lab.github.io/mcs/functions.html#def-function
[c] https://morrison-lab.github.io/mcs/functions.html#def-inverse-function
[d] https://morrison-lab.github.io/mcs/functions.html#def-logistic-function
[e] https://morrison-lab.github.io/mcs/expressions-algorithms.html#def-expression
[f] https://morrison-lab.github.io/pds/probability-basics.html#def-odds

4 Comparing softmax probability vectors

Exercise 4.1 (Comparing softmax probability vectors)**.** Sketch the probability mass function[a] that the softmax function produces for each of these score vectors, showing only the relative heights of the three probabilities in each:

$$\underset{\sim}{v}_1 = \text{softmax}\begin{pmatrix}0\\0\\0\end{pmatrix}, \quad \underset{\sim}{v}_2 = \text{softmax}\begin{pmatrix}3\\3\\3\end{pmatrix}, \quad \underset{\sim}{v}_3 = \text{softmax}\begin{pmatrix}0\\-1\\2\end{pmatrix}, \quad \underset{\sim}{v}_4 = \text{softmax}\begin{pmatrix}2\\0\\0\end{pmatrix}.$$

Then state which two are identical, and why, without computing any exponentials.

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

Solution

Solution 4.1. The probability vectors $\underset{\sim}{v}_1$ and $\underset{\sim}{v}_2$ are equal.
Solution 2.2 showed that adding the same number, $^{\top}\underset{\sim}{x}$, to every score leaves the softmax unchanged. Every entry of $(3,3,3)^{\top}$ is 3 more than the matching entry of $(0,0,0)^{\top}$, so the two score vectors give the same probability vector.
Both $\underset{\sim}{v}_1$ and $\underset{\sim}{v}_2$ are uniform, with 1/3 in each coordinate (Figure 3). In $\underset{\sim}{v}_3$, the third coordinate is the largest and the second is the smallest, close to 0. In $\underset{\sim}{v}_4$, the first coordinate is the largest, and the second and third are equal and smaller.

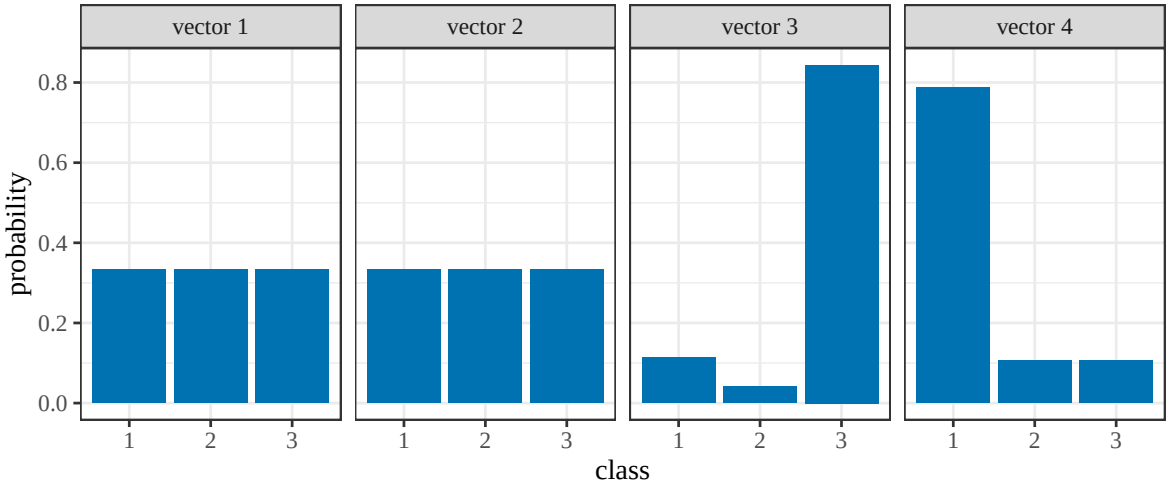


Figure 3: The probability mass functions[a] of the four softmax vectors. Vectors 1 and 2 give the same probabilities, one third each. Vector 3, from the scores $(0,-1,2)^{\top}$, puts most of its mass on the third class, and vector 4, from $(2,0,0)^{\top}$, on the first.

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

Exercise 4.2 (Comparing softmax probability vectors (part 2))**.** Use the setup of Exercise 4.1. Give all four to three decimal places.

Solution

Solution 4.2.

Table 1: The three values for each of four score vectors.

	class 1	class 2	class 3
$\underset{\sim}{v}_1$	0.333	0.333	0.333
$\underset{\sim}{v}_2$	0.333	0.333	0.333
$\underset{\sim}{v}_3$	0.114	0.042	0.844

$\underset{\sim}{v}_4$	0.787	0.107	0.107

Exercise 4.3 (Comparing softmax probability vectors (part 3))**.** Use the setup of Exercise 4.1. In $\underset{\sim}{v}_3$, the largest score exceeds the smallest by 3. Give the ratio of the corresponding two probabilities, exactly, and state what determines it.

Solution

Solution 4.3. Write $p_m = \text{softmax}(\underset{\sim}{z})_m$ and Z for the normalizer of Definition 2.1. The ratio of any two softmax probabilities is

$$\begin{aligned} \frac{p_m}{p_k} &= \frac{e^{z_m}/Z}{e^{z_k}/Z} && \text{(definition of the softmax)} \\ &= \frac{e^{z_m}}{e^{z_k}} && \text{(the normalizer } Z \text{ cancels)} \\ &= e^{z_m - z_k} && \text{(quotient of exponentials).} \end{aligned}$$

Here $z_3 - z_2 = 2 - (-1) = 3$, so

$$\begin{aligned} p_3/p_2 &= e^3 \\ &= 20.09, \end{aligned}$$

which is the ratio of the unrounded probabilities of Solution 4.2, 20.09.
This ratio depends only on the *difference* $z_m - z_k$, not on the individual values of z_m and z_k, the number of classes[a], or the scores of the other classes. This property is the pairwise form of the shift invariance in Solution 4.1. Taking logarithms gives $\log(p_m/p_k) = z_m - z_k$: the difference between two class[b] scores is the log of the ratio of their probabilities. With two classes, this log ratio is the log-odds[c] of the log-odds exercise.

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

Exercise 4.4 (Comparing softmax probability vectors (part 4))**.** Use the setup of Exercise 4.1. Raising the first score of $\underset{\sim}{v}_4$ moves the output toward the indicator vector[a] $(1, 0, 0)$. What must the scores approach for the softmax output to equal $(1, 0, 0)$, and how does this relate to the separation exercise?

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

Solution

Solution 4.4. No finite score vector produces $(1, 0, 0)^\top$. Because $e^{z_m} > 0$ for every finite z_m, every softmax probability p_m is strictly positive. The softmax output approaches $(1, 0, 0)^\top$ only in the limit as $z_1 - z_2 \to +\infty$ and $z_1 - z_3 \to +\infty$.
This behavior is the C-class version of the separation exercise. Suppose some coefficient matrix B gives every training observation a strictly larger score for its own class[a] than for any other class, the multi-class version of linearly separable classes[b]. Multiplying B by a growing positive number makes every score difference grow without bound, so every fitted probability of an observed class approaches 1, and the likelihood[c] approaches 1. No finite B reaches likelihood 1, because every probability is less than 1, so the coefficients grow without bound and no finite maximum likelihood estimate[d] exists. For $C = 2$, the fact that every probability is in $(0, 1)$ reduces to the sigmoid properties exercise: $\text{expit}(\eta) \in (0, 1)$ for all finite η.

[a] https://morrison-lab.github.io/sds/classification.html#def-class
[b] https://morrison-lab.github.io/sds/classification.html#def-class
[c] https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik
[d] https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle

5 Fitting the model

Exercise 5.1 (More than two classes (part 3))**.** Use the setup of Exercise 2.1.
Derive $\nabla_{B_m} \ell$ for a dataset of n observations, where $\ell = \sum_i \log P(Y = y_i \mid \underset{\sim}{x}_i)$. Some references write a factor of $1/n$ in front of this gradient. Explain when that factor is correct and when it is a mistake.

Solution

Solution 5.1. Substitute the model into the log-likelihood[a]:

$$\begin{aligned}
\ell &= \sum_i \log \frac{e^{B_{y_i}{}^\top \underset{\sim}{x}_i}}{\sum_k e^{B_k{}^\top \underset{\sim}{x}_i}} && \text{(the model)} \\
&= \sum_i \left[\log e^{B_{y_i}{}^\top \underset{\sim}{x}_i} - \log \sum_k e^{B_k{}^\top \underset{\sim}{x}_i} \right] && \text{(log of a quotient)} \\
&= \sum_i \left[B_{y_i}{}^\top \underset{\sim}{x}_i - \log \sum_k e^{B_k{}^\top \underset{\sim}{x}_i} \right] && \text{(log of an exponential).}
\end{aligned}$$

The first term contributes $\underset{\sim}{x}_i$ to ∇_{B_m} when $y_i = m$ and $\vec{0}$ otherwise, which equals $\delta(y_i = m)\, \underset{\sim}{x}_i$. In the second term, only the $k = m$ summand depends on B_m, so

$$\begin{aligned}
\nabla_{B_m} \log \sum_k e^{B_k{}^\top \underset{\sim}{x}_i} &= \frac{1}{\sum_k e^{B_k{}^\top \underset{\sim}{x}_i}} \nabla_{B_m} \sum_k e^{B_k{}^\top \underset{\sim}{x}_i} && \text{(chain rule for the logarithm)} \\
&= \frac{1}{\sum_k e^{B_k{}^\top \underset{\sim}{x}_i}} \nabla_{B_m} e^{B_m{}^\top \underset{\sim}{x}_i} && \text{(only the } k = m \text{ summand depends on } B_m\text{)} \\
&= \frac{1}{\sum_k e^{B_k{}^\top \underset{\sim}{x}_i}} e^{B_m{}^\top \underset{\sim}{x}_i}\, \underset{\sim}{x}_i && \text{(chain rule for the exponential)} \\
&= P(Y = m \mid \underset{\sim}{x}_i)\, \underset{\sim}{x}_i && \text{(the model).}
\end{aligned}$$

Combining the two terms gives

$$\nabla_{B_m} \ell = \sum_{i=1}^{n} [\delta(y_i = m) - P(Y = m \mid \underset{\sim}{x}_i)] \underset{\sim}{x}_i,$$

which has the same form as the binary[b] gradient: the residual times the feature vector, summed over observations[c].
Whether the factor of $1/n$ belongs depends on the objective. Multiplying $\nabla_{B_m} \ell$ by $-1/n$ gives the gradient of the *mean*[d] cross-entropy[e], an empirical risk, and multiplying it by -1 alone gives the gradient of the *summed* cross-entropy, the negative log-likelihood. So the factor is correct for the mean and a mistake for the sum or for ℓ itself. Multiplying an objective by a nonzero constant[f] does not change where its gradient vanishes, so the two conventions yield the same maximum likelihood estimate[g].
The factor of $1/n$ does change step sizes and curvature. Including $1/n$ in the gradient rescales the effective step size by $1/n$, as analyzed in the stochastic-gradient exercises of the gradient descent chapter, and it rescales the Hessian[h] by $1/n$, so standard errors[i] computed from the rescaled Hessian without accounting for n would be off by a factor of $\sqrt{n}$.

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

[b] https://morrison-lab.github.io/sds/classification.html#def-binary-classification
[c] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-observation
[d] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean
[e] https://morrison-lab.github.io/pds/expectation.html#def-cross-entropy
[f] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function
[g] https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle
[h] https://morrison-lab.github.io/mds/vector-calculus.html#def-hessian
[i] https://morrison-lab.github.io/sds/estimation.html#def-SE

Theorem 5.1 (Gradient of the multinomial log-likelihood)**.** *In the multinomial logistic regression model, the gradient of the log-likelihood*[a] $\ell = \sum_i \log P(Y = y_i \mid \underset{\sim}{x}_i)$ *with respect to the coefficient vector* B_m *of class*[b] m *is*

$$\nabla_{B_m} \ell = \sum_{i=1}^{n} [\delta(y_i = m) - P(Y = m \mid \underset{\sim}{x}_i)] \underset{\sim}{x}_i.$$

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

Proof

Proof. By Solution 5.1. □

Example 5.1 (The gradient for one observation)**.**
Use the three coefficient vectors of Example 2.2 and a single observation[a] with $\underset{\sim}{x} = (1, 1)^\top$ and class[b] 3. At $x = 1$, the probabilities[c] $P(Y = m \mid \underset{\sim}{x})$ are 0.122, 0.547 and 0.331 for $m = 1, 2, 3$.
By Theorem 5.1, the gradient for class m is the residual $\delta(y = m) - P(Y = m \mid \underset{\sim}{x})$ times $\underset{\sim}{x}$. The residuals are -0.122, -0.547 and 0.669, so $\nabla_{B_1} \ell = (-0.122, -0.122)^\top$, $\nabla_{B_2} \ell = (-0.547, -0.547)^\top$ and $\nabla_{B_3} \ell = (0.669, 0.669)^\top$. Only the observed class has a residual above zero, so a step up the gradient raises the probability of class 3 and lowers the others.

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

Exercise 5.2 (More than two classes (part 4))**.** Use the setup of Exercise 2.1.
Check Exercise 5.1 against the gradient exercise, part 2 by specializing to $C = 2$ under the convention you chose in Exercise 2.2.

Solution

Solution 5.2. For $C = 2$ with class 0 fixed at $B_0 = \vec{0}$ and $B_1 = \vec{\beta}$, Solution 2.1 gives $P(Y = 1 \mid \underset{\sim}{x}_i) = \pi_i$, and $\delta(y_i = 1) = y_i$ because $y_i \in \{0, 1\}$. The gradient from Solution 5.1 for $m = 1$ then becomes

$$\begin{aligned} \nabla_{\vec{\beta}} \ell &= \sum_i (y_i - \pi_i) \underset{\sim}{x}_i \quad \text{(substitute } \pi_i \text{ and } y_i\text{)} \\ &= \mathbf{D}(\mathbf{x})^\top (\underset{\sim}{y} - \underset{\sim}{\pi}) \quad \text{(the rows } \underset{\sim}{x}_i \text{ of } \mathbf{D}(\mathbf{x}) \text{, weighted and summed).} \end{aligned}$$

This gradient is minus the gradient of the cost in the gradient exercise, part 2, because maximizing ℓ is the same as minimizing $-\ell$.
Once $B_0 = \vec{0}$ is fixed, the $m = 0$ score equation is redundant: its residuals are minus the $m = 1$ residuals, because the two fitted probabilities sum to 1 and the two class indicators sum to 1. This redundancy reflects the reduction from $C(p + 1)$ parameters to $(C - 1)(p + 1)$ identified parameters in Solution 2.2.

Exercise 5.3 (More than two classes (part 5))**.** Use the setup of Exercise 2.1.
Setting Exercise 5.1 to zero gives C score equations. Assume that $\underset{\sim}{x}$ has an intercept[a] component. Interpret in words the intercept entry of the score equation for class[b] m, and compare it with the gradient exercise, part 5.

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

Solution

Solution 5.3. The intercept[a] entry of $\underset{\sim}{x}_i$ is 1, so the intercept entry of the mth score equation is

$$\sum_{i=1}^{n} \delta(y_i = m) = \sum_{i=1}^{n} P(Y = m \mid \underset{\sim}{x}_i).$$

The left side is the observed count in class[b] m, and the right side is the sum of the fitted probabilities of class m. So the fitted model matches the sample[c] count of every class.
The gradient exercise, part 5 is the $C = 2$ special case, where matching the count of class 1 also matches the count of class 0, because the two probabilities sum to 1. With C classes[d], summing the C intercept equations over m gives n on both sides, so any $C-1$ of them imply the last, and only $C-1$ are linearly independent[e].
These equations make the fitted model match the class totals, not predict each observation accurately: a model can match every class total while assigning inaccurate probabilities to individual observations. An intercept-only model, which gives every observation the same probabilities, is an example.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[b] https://morrison-lab.github.io/sds/classification.html#def-class
[c] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample
[d] https://morrison-lab.github.io/sds/classification.html#def-class
[e] https://morrison-lab.github.io/mds/linear-algebra-subspaces.html#def-linearly-independent

Exercise 5.4 (More than two classes (part 6))**.** Use the setup of Exercise 2.1.
Fit the intercept-only model to 100 observations, of which 10, 30 and 60 are in classes[a] 1, 2 and 3. Give the fitted probabilities, and the coefficients under the convention of Exercise 2.2 with class 1 as the reference.

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

Solution

Solution 5.4. With only an intercept[a], every $\underset{\sim}{x}_i = 1$, so the fitted probabilities p_m are constant[b] across observations. Let n_m be the number of observations in class[c] m. The score equation of Solution 5.3 is then $n_m = 100\, p_m$, so

$$\hat{p}_1 = 0.1, \qquad \hat{p}_2 = 0.3, \qquad \hat{p}_3 = 0.6,$$

which are the sample proportions.
Fixing class 1 as the reference class sets $b_1 = 0$. By Solution 4.3, the ratio of two softmax probabilities depends only on the difference of their scores:

$$\begin{aligned} \frac{p_m}{p_1} &= \mathrm{e}^{b_m - b_1} && \text{(ratio of two softmax probabilities)} \\ &= \mathrm{e}^{b_m} && \text{(reference class, } b_1 = 0\text{)}. \end{aligned}$$

Taking logarithms gives $\hat{b}_m = \log(\hat{p}_m / \hat{p}_1)$, so

$$\hat{b}_2 = \log\frac{0.3}{0.1} \qquad \hat{b}_3 = \log\frac{0.6}{0.1}$$
$$= \log 3 \qquad = \log 6$$
$$= 1.0986, \qquad = 1.7918.$$

As a check, the exponentials of the scores are $e^0 = 1$, $e^{\log 3} = 3$ and $e^{\log 6} = 6$, and

$$1 + 3 + 6 = 10,$$

so dividing the exponentials $1, 3, 6$ by the normalizer 10 gives $0.1, 0.3, 0.6$.
Each coefficient is the log of the ratio of its class's probability to the reference class's probability, so its value depends on the choice of reference class, as in the worked fit exercise, part 5. If class 3 is the reference class instead ($b_3 = 0$), the same fitted model has parameters $\hat{b}_1 = \log\frac{1}{6}$ and $\hat{b}_2 = \log\frac{1}{2}$, while all fitted probabilities are unchanged.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[b] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function
[c] https://morrison-lab.github.io/sds/classification.html#def-class

6 Large scores and weight penalties

Exercise 6.1 (A softmax with large scores)**.** Consider four classes with the score vector $\underset{\sim}{z} = (23,\ -100,\ -15,\ 20)^\top$.
Exponentiate each score, give the normalizer $Z = \sum_k e^{z_k}$, and give the four probabilities $\text{softmax}(\underset{\sim}{z})$, each to two significant figures.

Solution

Solution 6.1.

6.0.1 R

```
ls_z <- c(23, -100, -15, 20)
ls_exp <- exp(ls_z)
ls_norm <- sum(ls_exp)
ls_prob <- ls_exp / ls_norm
# nolint start: undesirable_function_linter. A matrix.
rbind(exponential = ls_exp, softmax = ls_prob) |> signif(3)
```

```
                [,1]     [,2]     [,3]     [,4]
exponential 9.74e+09 3.72e-44 3.06e-07 4.85e+08
softmax     9.53e-01 3.64e-54 2.99e-17 4.74e-02
```

```
# nolint end
```

6.0.2 Python

```
import numpy as np

z_ls = np.array([23.0, -100.0, -15.0, 20.0])
exp_ls = np.exp(z_ls)
print(f"normalizer: {exp_ls.sum():.4g}\n{np.vstack([exp_ls, exp_ls / exp_ls.sum()])}")
```

```
normalizer: 1.023e+10
[[9.74480345e+09 3.72007598e-44 3.05902321e-07 4.85165195e+08]
 [9.52574127e-01 3.63644905e-54 2.99025668e-17 4.74258732e-02]]
```

Exponentiating gives 9.74×10^{9}, 3.72×10^{-44}, 3.06×10^{-7} and 4.85×10^{8}. Only the first and the last are comparable in magnitude to the normalizer $Z = 1.023 \times 10^{10}$, so dividing each exponential by Z gives, to two significant figures,

$$\operatorname{softmax}(\underset{\sim}{z}) = \left(\, 0.95,\ 3.6 \times 10^{-54},\ 3 \times 10^{-17},\ 0.047 \,\right)^{\top}.$$

Exercise 6.2 (A softmax with large scores (part 2))**.** Use the setup of Exercise 6.1.
State what the softmax keeps of the scores' ordering and what it keeps of their spacing, using the gap of 3 between the two leading scores and the gap of 38 between the leading score and the third. Explain how that relates to the name *softmax*.

Solution

Solution 6.2. The softmax keeps the ordering of the scores, because the exponential function is strictly increasing. It does not keep their spacing: by Solution 4.3, a difference between two scores becomes a ratio between their probabilities. The difference of 3 between the two leading scores becomes a factor of $e^{3} \approx 20$ between their probabilities, and the difference of 38 between the leading score and the third gives the third class[a] a probability of 3×10^{-17}. The name "softmax" refers to this behavior: the softmax is a smooth approximation to the indicator vector[b] of the largest score, and at this spacing it gives nearly all of the probability to the largest score.

[a] https://morrison-lab.github.io/sds/classification.html#def-class
[b] https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-indicator-vector

Exercise 6.3 (A softmax with large scores (part 3))**.** Use the setup of Exercise 6.1.
In double-precision floating-point arithmetic, e^{z} overflows to infinity just below $z = 710$. State what a direct implementation returns for the scores $\underset{\sim}{z} + 700$, and use Exercise 2.2 to give an implementation that cannot overflow.

Solution

Solution 6.3. Adding 700 to every score gives the scores 723, 600, 685 and 720. The exponentials e^{723} and e^{720} exceed the overflow threshold, so they become infinity, and the normalizer becomes infinity with them. Direct evaluation then divides infinity by infinity for the first and last classes[a], returning `NaN`, and divides a finite number by infinity for the other two classes, returning 0. The exact probability vector equals the vector of Solution 6.0.2, because adding the same number to every score leaves every softmax probability unchanged, by Solution 2.2.
That shift invariance also prevents overflow. Before exponentiating, subtract the maximum score $\max_k z_k$ from every entry. This shift leaves the probabilities unchanged and makes the largest exponential $e^{0} = 1$, so no term can overflow, and because the normalizer is then at least 1, it cannot underflow to 0.

6.0.3 R

```r
# nolint start: function_location_linter. Two helpers used only here.
softmax_direct <- function(z) {
  e <- exp(z)
  e / sum(e)
}
softmax_stable <- function(z) {
  e <- exp(z - max(z))
  e / sum(e)
}
# nolint end
ls_big <- ls_z + 700
ls_direct <- softmax_direct(ls_big)
ls_stable <- softmax_stable(ls_big)
# nolint start: undesirable_function_linter. A matrix.
rbind(direct = ls_direct, stable = ls_stable) |> signif(2)
```

```
       [,1]    [,2]  [,3]  [,4]
direct  NaN 0.0e+00 0e+00   NaN
stable 0.95 3.6e-54 3e-17 0.047
```

```r
# nolint end
```

6.0.4 Python

```python
import numpy as np
from scipy.special import softmax

z_ls = np.array([23.0, -100.0, -15.0, 20.0]) + 700
with np.errstate(over="ignore", invalid="ignore"):
    direct_ls = np.exp(z_ls) / np.exp(z_ls).sum()
stable_ls = np.exp(z_ls - z_ls.max()) / np.exp(z_ls - z_ls.max()).sum()
print(f"direct: {direct_ls}\nstable: {stable_ls}\nscipy:  {softmax(z_ls)}")
```

```
direct: [nan  0.  0. nan]
stable: [9.52574127e-01 3.63644905e-54 2.99025668e-17 4.74258732e-02]
scipy:  [9.52574127e-01 3.63644905e-54 2.99025668e-17 4.74258732e-02]
```

SciPy's `scipy.special.softmax` returns the same vector as the stable version, because it subtracts the maximum before exponentiating [7].

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

Exercise 6.4 (Penalizing the softmax weights)**.** Collect the class vectors as the columns of a $(p+1)\times C$ matrix B, with entries B_{rk} for row r and class k. Index the rows by $r = 0, 1, \dots, p$, keeping i for observations. Row 0 holds the intercepts and rows 1 through p the feature weights. Leave the intercepts out of the penalty, as the cross-entropy exercise, part 10b argued for the binary model, and maximize the penalized objective

$$J(B) = \ell(B) - \lambda \sum_{r=1}^{p} \sum_{k=1}^{C} B_{rk}^2.$$

Derive the gradient of J with respect to B_m, using Exercise 5.1 for the gradient of ℓ.

Solution

Solution 6.4.
The gradient of a difference is the difference of the gradients, so

$$\nabla_{B_m} J(B) = \nabla_{B_m} \ell(B) - \lambda \nabla_{B_m} \sum_{r=1}^{p} \sum_{k=1}^{C} B_{rk}^2,$$

and Solution 5.1 gives the first term. In the second gradient, only the terms from column m and rows $r \geq 1$ depend on B_m, and the derivative of B_{rm}^2 with respect to B_{rm} is $2B_{rm}$, so

$$\nabla_{B_m} \sum_{r=1}^{p} \sum_{k=1}^{C} B_{rk}^2 = 2\tilde{B}_m, \qquad \tilde{B}_m = \big(\, 0,\ B_{1m},\ \ldots,\ B_{pm} \,\big)^\top,$$

where $\tilde{B}_m$ is B_m with its intercept[a] entry set to zero. Substituting both terms gives

$$\nabla_{B_m} J(B) = \sum_{i=1}^{n} [\delta(y_i = m) - P(Y = m \mid \underset{\sim}{x}_i)] \underset{\sim}{x}_i - 2\lambda \tilde{B}_m.$$

The penalty term pulls each non-intercept entry of B_m toward 0 in proportion to its magnitude, and it leaves the intercept entry unpenalized.

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

Exercise 6.5 (Penalizing the softmax weights (part 2))**.** Use the setup of Exercise 6.4. Exercise 2.2 showed that adding the same vector to every column leaves the likelihood[a] unchanged. State which of those shifts change the penalty, which representative of each equivalence class the penalized objective selects, and whether the remaining unpenalized direction in the intercept[b] row affects the predicted probabilities.

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

Solution

Solution 6.5. Let the shift vector be $= (c_0, c_1, \ldots, c_p)^\top$. The penalty depends only on the feature rows $r \geq 1$, where after the shift it equals $\sum_{r \geq 1} \sum_k (B_{rk} + c_r)^2$. This sum depends on $c_1, \ldots, c_p$, so the shifts that change any of $c_1, \ldots, c_p$ change the penalty, although they give the same likelihood[a]. Setting the derivative with respect to c_r to zero gives $\sum_k (B_{rk} + c_r) = 0$. So the penalized objective selects the representative with the smallest penalty, in which each feature row sums to zero across the C classes[b]. Neither the likelihood nor the penalty depends on the intercept[c] shift c_0, so adding the same number to all C intercepts leaves the penalized objective unchanged. This shift does not change any predicted probability, but the objective is constant[d] along it, so an optimizer that requires a unique maximizer must still fix one class[e] intercept (for example, to 0) or constrain the intercepts to sum to zero.

[a] https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik
[b] https://morrison-lab.github.io/sds/classification.html#def-class
[c] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[d] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function
[e] https://morrison-lab.github.io/sds/classification.html#def-class

7 Videos

Watch

- Hastie and Tibshirani extend logistic regression to several classes, from 5:43[a] [8], video 4.4.
- Hutchinson teaches multinomial logistic regression and the softmax[b] (20 min) [9].
- Starmer explains argmax and softmax[c] [10].

[a] https://www.youtube.com/watch?v=sYDDk6R-be0&t=343s
[b] https://facultyweb.cs.wwu.edu/~hutchib2/video_lectures/data371/#mult_log_regression

[c] https://www.youtube.com/watch?v=KpKog-L9veg

References

[1] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning with applications in R*, 2nd ed. Springer, 2021. Available: https://www.statlearning.com/

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, 2016. Available: https://www.deeplearningbook.org/

[3] C. M. Bishop, *Pattern recognition and machine learning*. in Information science and statistics. New York: Springer, 2006.

[4] GeeksforGeeks, "Softmax activation function in neural networks." GeeksforGeeks, tutorial article, 2025. Available: https://www.geeksforgeeks.org/deep-learning/the-role-of-softmax-in-neural-networks-detailed-explanation-and-applications/

[5] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: Data mining, inference, and prediction*, 2nd ed. Springer, 2009. Available: https://hastie.su.domains/ElemStatLearn/

[6] P. McCullagh, "Regression models for ordinal data," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 42, no. 2, pp. 109–142, 1980, doi: 10.1111/j.2517-6161.1980.tb01109.x[4].

[7] The SciPy Developers, "Source code of `scipy.special.softmax`, in `scipy/special/_logsumexp.py`." SciPy version 1.18.1, 2026. Available: https://github.com/scipy/scipy/blob/v1.18.1/scipy/special/_logsumexp.py

[8] T. Hastie, R. Tibshirani, and J. Taylor, "Statistical learning with python." Stanford Online, lecture videos on YouTube, 2023. Available: https://www.youtube.com/playlist?list=PLoROMvodv4rPP6braWoRt5UCXYZ71GZIQ

[9] B. Hutchinson, "DATA 471/571 (machine learning) and CSCI 481/581 (deep learning) video lectures." Western Washington University. Accessed: Sep. 28, 2026. [Online]. Available: https://facultyweb.cs.wwu.edu/~hutchib2/video_lectures/data371/

[10] J. Starmer, "Neural networks part 5: ArgMax and SoftMax." StatQuest, video on YouTube, 2021. Available: https://www.youtube.com/watch?v=KpKog-L9veg

[4] https://doi.org/10.1111/j.2517-6161.1980.tb01109.x