Discriminant Analysis

Last modified: 2026-11-19

```
# The Python chunks on this page import these packages; reticulate provisions them.
library(reticulate)
library(tidymodels) # nolint: undesirable_function_linter
```

```
-- Attaching packages --------------------------------------- tidymodels 1.5.0 --

v broom        1.0.13     v recipes      1.4.0
v dials        1.4.5      v rsample      1.3.2
v dplyr        1.2.1      v tailor       0.1.0
v ggplot2      4.0.3      v tidyr        1.3.2
v infer        1.1.0      v tune         2.1.0
v modeldata    1.6.0      v workflows    1.3.0
v parsnip      1.6.1      v workflowsets 1.1.1
v purrr        1.2.2      v yardstick    1.4.0

-- Conflicts ------------------------------------------ tidymodels_conflicts() --
x purrr::discard() masks scales::discard()
x dplyr::filter()  masks stats::filter()
x dplyr::lag()     masks stats::lag()
x recipes::step()  masks stats::step()
```

```
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. Its sources are credited in that page's Source note.
> The naive Bayes assumption and its Spambase example moved here from the probability page; they are adapted from Brian Hutchinson's Fall 2024 DATA 471 probability refresher [1].

1 Classifying with Bayes' rule

A logistic regression model gives the probability of each value of the label given the features. An alternative is to model the distribution[1] of the features within each class[2], and to compute the posterior probability of the label by Bayes' rule[3]. Choosing a multivariate normal distribution[4] for the features of each class gives linear and quadratic discriminant analysis. Their decision boundaries are linear or quadratic[5], and they are

[1] https://morrison-lab.github.io/pds/random-variables.html#def-cdf
[2] https://morrison-lab.github.io/sds/classification.html#def-class
[3] https://morrison-lab.github.io/pds/probability-basics.html#thm-bayes
[4] https://morrison-lab.github.io/pds/distributions.html#def-mvn
[5] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-quadratic-cubic

fitted by maximizing the joint[6] likelihood[7] of the labels and the features, not the conditional likelihood of the labels. This page writes K for the number of classes[8], as James et al. [2] does.

Exercise 1.1 (Classifying with Bayes' rule)**.** A logistic regression model gives $\Pr[Y = k \mid \underset{\sim}{X} = \underset{\sim}{x}]$ directly. Suppose instead that the label has K classes[a] and that we knew, for each class[b] $k = 1, \ldots, K$:

- the class prior $\pi_k = \Pr[Y = k]$, the probability of class k before the features are seen;
- the class-conditional density $f_k(\underset{\sim}{x})$, the conditional density[c] of the features given $Y = k$.

Use Bayes' rule[d] to express $\Pr[Y = k \mid \underset{\sim}{X} = \underset{\sim}{x}]$ in terms of π_k and f_k.

[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/expectation.html#def-cond-pdf
[d] https://morrison-lab.github.io/pds/probability-basics.html#thm-bayes

Solution

Solution 1.1. Bayes' rule[a] gives the posterior[b] as the prior[c] times the class[d] density[e], divided by the marginal density[f] of $\underset{\sim}{X}$. The law of total probability[g] writes that marginal density as $\sum_{l=1}^{K} \pi_l f_l(\underset{\sim}{x})$. So, wherever that sum is positive,

$$\Pr[Y = k \mid \underset{\sim}{X} = \underset{\sim}{x}] = \frac{\pi_k f_k(\underset{\sim}{x})}{\sum_{l=1}^{K} \pi_l f_l(\underset{\sim}{x})}.$$

With two classes[h], diseased and healthy, and a single binary feature, a positive test result, this formula is the positive predictive value of a diagnostic test[i]: the prior of the diseased class is the prevalence[j], the probability of a positive test is the sensitivity[k] in the diseased class, and it is one minus the specificity[l] in the healthy class.

[a] https://morrison-lab.github.io/pds/probability-basics.html#thm-bayes
[b] https://morrison-lab.github.io/sds/bayesian-inference.html#def-posterior
[c] https://morrison-lab.github.io/sds/bayesian-inference.html#def-prior
[d] https://morrison-lab.github.io/sds/classification.html#def-class
[e] https://morrison-lab.github.io/pds/random-variables.html#def-pdf
[f] https://morrison-lab.github.io/pds/random-variables.html#def-marginal
[g] https://morrison-lab.github.io/pds/probability-basics.html#thm-total-prob
[h] https://morrison-lab.github.io/sds/classification.html#def-class
[i] https://morrison-lab.github.io/sds/classification.html#thm-ppv-bayes
[j] https://morrison-lab.github.io/sds/classification.html#def-prevalence
[k] https://morrison-lab.github.io/sds/classification.html#def-sensitivity
[l] https://morrison-lab.github.io/sds/classification.html#def-specificity

Exercise 1.2 (Classifying with Bayes' rule (part 2))**.** Use the setup of Exercise 1.1. To classify $\underset{\sim}{x}$, we select the class with the largest posterior probability. Which factor in your formula can be omitted when comparing classes, and why?

Solution

Solution 1.2. In the posterior formula of Solution 1.1, the denominator is the same for every k, because it is the marginal density[a] of $\underset{\sim}{X}$, with the class[b] summed out. At a point $\underset{\sim}{x}$ where that density is positive, dividing every posterior by the same positive number does not change which posterior is largest. So the predicted class is

$$\hat{y}(\underset{\sim}{x}) = \arg\max_k \ \pi_k f_k(\underset{\sim}{x}),$$

and it is enough to compare the products $\pi_k f_k(\underset{\sim}{x})$.

[6] https://morrison-lab.github.io/pds/random-variables.html#def-joint-distribution
[7] https://morrison-lab.github.io/sds/intro-MLEs.html#def-lik
[8] https://morrison-lab.github.io/sds/classification.html#def-class

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

Exercise 1.3 (Classifying with Bayes' rule (part 3))**.** Use the setup of Exercise 1.1. Where would π_k and f_k come from in practice?

Solution

Solution 1.3. Both are estimated[a] from the training data. The proportion of training points in class k estimates[b] π_k. Estimating f_k means[c] choosing a family of densities for the features within a class and fitting it to the training points of that class. Each choice of family gives a different classifier. The rest of this section uses the multivariate normal[d] family.

[a] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[b] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[c] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean
[d] https://morrison-lab.github.io/pds/distributions.html#def-mvn

Theorem 1.1 (The posterior from a prior and class densities)**.** *Let class*[a] k *have prior* $\pi_k > 0$ *and feature density* $f_k(\underset{\sim}{x})$*, for* $k = 1, \ldots, K$*, and let* $\underset{\sim}{x}$ *be a point where the marginal*[b] *density* $\sum_{l=1}^{K} \pi_l f_l(\underset{\sim}{x})$ *is greater than* 0*. Then*

$$\Pr[Y = k \mid \underset{\sim}{X} = \underset{\sim}{x}] = \frac{\pi_k f_k(\underset{\sim}{x})}{\sum_{l=1}^{K} \pi_l f_l(\underset{\sim}{x})},$$

and the class with the largest posterior is $\hat{y}(\underset{\sim}{x}) = \arg\max_k \ \pi_k f_k(\underset{\sim}{x})$.

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

Proof

Proof. By Solution 1.1 and Solution 1.2. □

Example 1.1 (A posterior from two class densities)**.** Two classes[a] have priors $\pi_1 = 0.3$ and $\pi_2 = 0.7$, and at an observed $\underset{\sim}{x}$ their class[b] densities are $f_1(\underset{\sim}{x}) = 0.5$ and $f_2(\underset{\sim}{x}) = 0.1$. Find the posteriors and the class with the larger posterior.

```
br_prior <- c(0.3, 0.7)
br_dens <- c(0.5, 0.1)
br_joint <- br_prior * br_dens
br_post <- br_joint / sum(br_joint)
rbind(joint = br_joint, posterior = br_post)
```

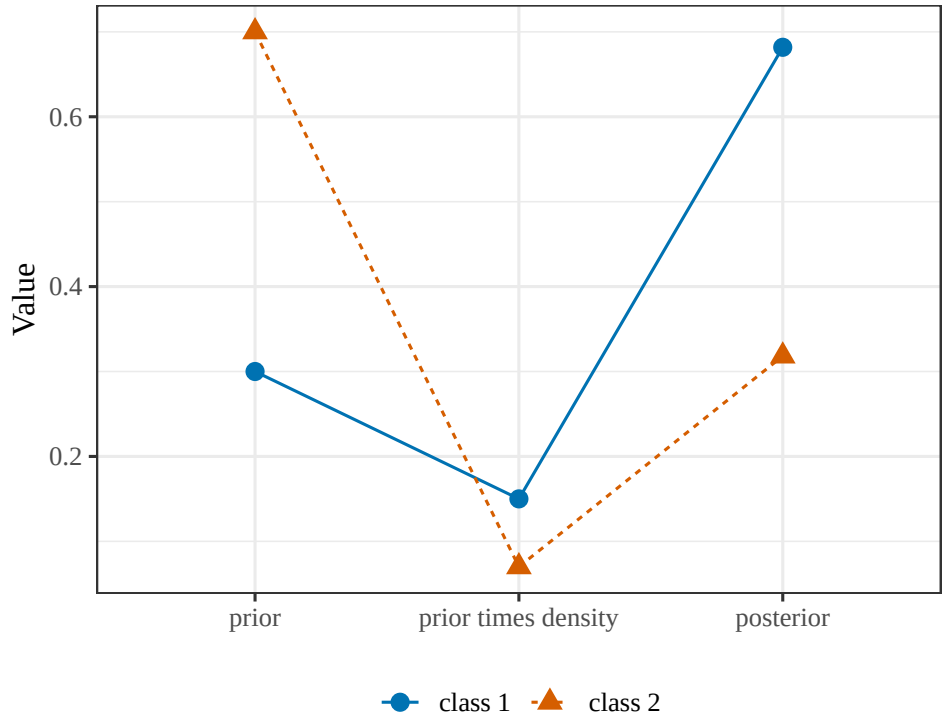
```
               [,1]      [,2]
joint     0.1500000 0.0700000
posterior 0.6818182 0.3181818
```

The products are 0.15 and 0.07, so by Theorem 1.1 the posteriors are 0.682 and 0.318. Class 1 has the larger posterior, although class 2 has the larger prior.

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

1.0.1 R

```r
library(ggplot2)
bf_steps <- c("prior", "prior times density", "posterior")
bf <- data.frame(
  step = factor(rep(bf_steps, each = 2), levels = bf_steps),
  class = rep(c("class 1", "class 2"), times = 3),
  value = c(br_prior, br_joint, br_post)
)
ggplot(bf) +
  aes(
    step, value,
    colour = class, shape = class, linetype = class, group = class
  ) +
  geom_line() +
  geom_point(size = 3) +
  scale_colour_manual(values = c("class 1" = "#0072B2", "class 2" = "#D55E00")) +
  labs(x = NULL, y = "Value", colour = NULL, shape = NULL, linetype = NULL)
```



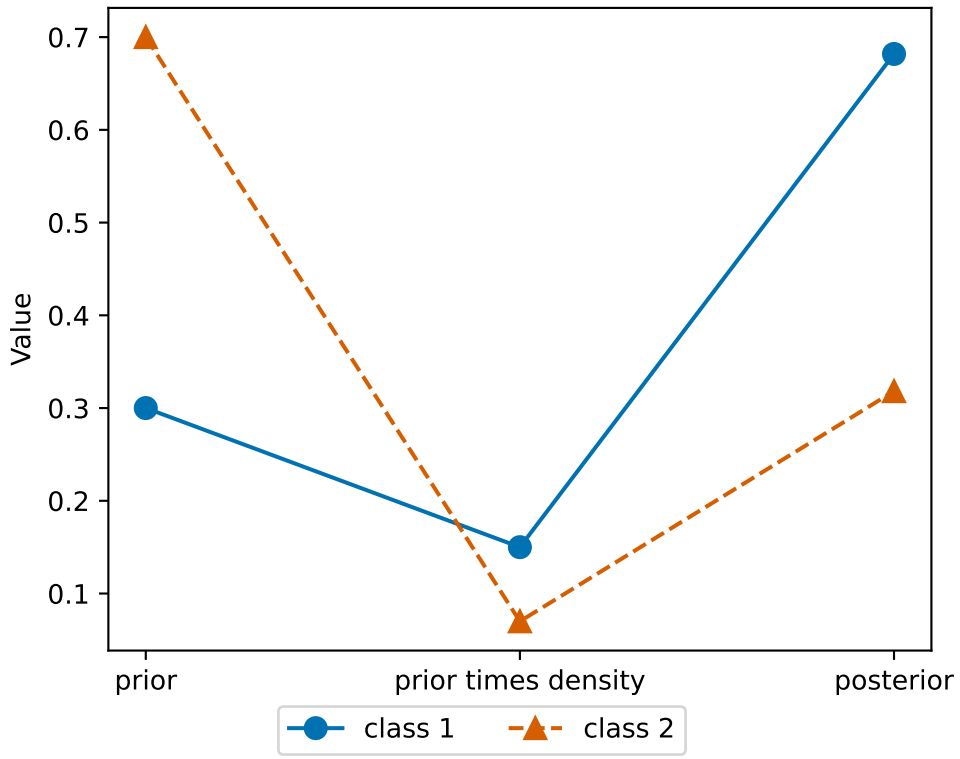
(a)

1.0.2 Python

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

prior_bf = np.array([0.3, 0.7])
dens_bf = np.array([0.5, 0.1])
joint_bf = prior_bf * dens_bf
post_bf = joint_bf / joint_bf.sum()
steps_bf = ["prior", "prior times density", "posterior"]
values_bf = np.vstack([prior_bf, joint_bf, post_bf])
fig, ax = plt.subplots(figsize=(5, 4), layout="constrained")
```

```
for k, (label, colour, marker, style) in enumerate(
    [("class 1", "#0072B2", "o", "-"), ("class 2", "#D55E00", "^", "--")]
):
    ax.plot(steps_bf, values_bf[:, k], color=colour, marker=marker, markersize=8,
            linestyle=style, label=label);
ax.set_ylabel("Value");
fig.legend(loc="outside lower center", ncol=2);
plt.show()
```



(b)

Figure 1: The prior, the prior times the density, and the posterior of the two classes of Example 1.1. Class 2 has the larger prior, but the density of class 1 is large enough at this observation that class 1 has the larger posterior.

Definition 1.1 (Bayes classifier)**.** Let π_k and f_k be the true class[a] priors and class-conditional densities of the features. The **Bayes classifier** assigns each $\underset{\sim}{x}$ to the class with the largest posterior probability,

$$\hat{y}(\underset{\sim}{x}) = \arg\max_k \ \pi_k f_k(\underset{\sim}{x}),$$

as in Theorem 1.1. No classifier has a smaller probability of misclassifying a new observation [2, Ch. 2], [3].

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

Example 1.2 (The Bayes classifier for two normal classes)**.** Suppose that a single feature X is normal[a] with mean[b] 0 and variance[c] 1 in class[d] 0, and with mean 2 and variance 1 in class 1, and that the true priors are $\pi_0 = 0.6$ and $\pi_1 = 0.4$. Find the rule of the Bayes classifier and its probability of misclassifying a new observation.

1.0.3 R

```r
bc_prior <- c(0.6, 0.4)
bc_mean <- c(0, 2)
# pi_0 f_0(x) = pi_1 f_1(x) where 2 x - 2 = log(pi_0 / pi_1); solve for x
bc_cut <- (2 + log(bc_prior[1] / bc_prior[2])) / 2
bc_error <- bc_prior[1] * pnorm(bc_cut, bc_mean[1], 1, lower.tail = FALSE) +
  bc_prior[2] * pnorm(bc_cut, bc_mean[2], 1)
c(cut = bc_cut, error = bc_error) |> round(4)
```

```
   cut  error
1.2027 0.1538
```

1.0.4 Python

```python
import numpy as np
from scipy.stats import norm

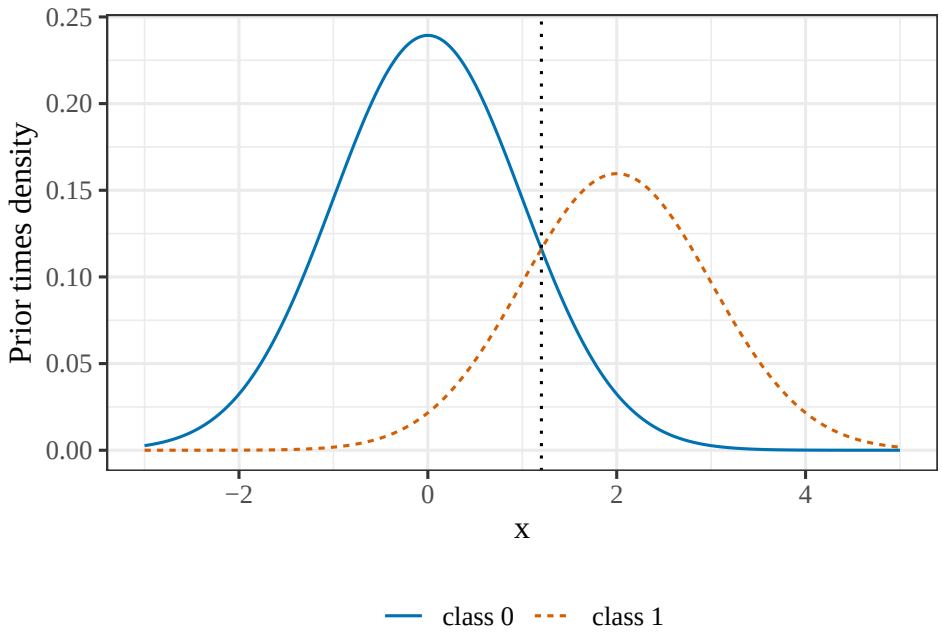
prior_bc = np.array([0.6, 0.4])
mean_bc = np.array([0, 2])
# pi_0 f_0(x) = pi_1 f_1(x) where 2 x - 2 = log(pi_0 / pi_1); solve for x
cut_bc = (2 + np.log(prior_bc[0] / prior_bc[1])) / 2
error_bc = prior_bc[0] * norm.sf(cut_bc, mean_bc[0], 1) + prior_bc[1] * norm.cdf(cut_bc, mean_bc[1], 1
print("cut:", round(cut_bc, 4), "error:", round(error_bc, 4))
```

```
cut: 1.2027 error: 0.1538
```

The products $\pi_0 f_0(x)$ and $\pi_1 f_1(x)$ are equal at $x = 1.203$, so the Bayes classifier assigns class[e] 0 to every x below 1.203 and class 1 to every x above it. The cut point is above the midpoint 1 of the two means[f], because class 0 has the larger prior. The classifier misclassifies a new observation with probability 0.154, and no other rule has a smaller probability for these two classes[g]. Figure 2 plots the two products: the class-0 curve is higher to the left of the dotted cut line, and the class-1 curve is higher to its right.

1.0.5 R

```r
# weight each class density by its prior, on a grid of feature values
bc_x <- seq(-3, 5, length.out = 401)
bc_w0 <- bc_prior[1] * dnorm(bc_x, bc_mean[1], 1)
bc_w1 <- bc_prior[2] * dnorm(bc_x, bc_mean[2], 1)
bc_curves <- tibble::tibble(x = bc_x, `class 0` = bc_w0, `class 1` = bc_w1) |>
  tidyr::pivot_longer(-x, names_to = "class", values_to = "weighted")
bc_colours <- c("class 0" = "#0072B2", "class 1" = "#D55E00")
# the Bayes classifier switches class where the two curves cross
bc_curves |>
  ggplot() +
  aes(x = x, y = weighted, colour = class, linetype = class) +
  geom_line() +
  geom_vline(xintercept = bc_cut, linetype = "dotted") +
  scale_colour_manual(values = bc_colours) +
  labs(x = "x", y = "Prior times density", colour = NULL, linetype = NULL)
```

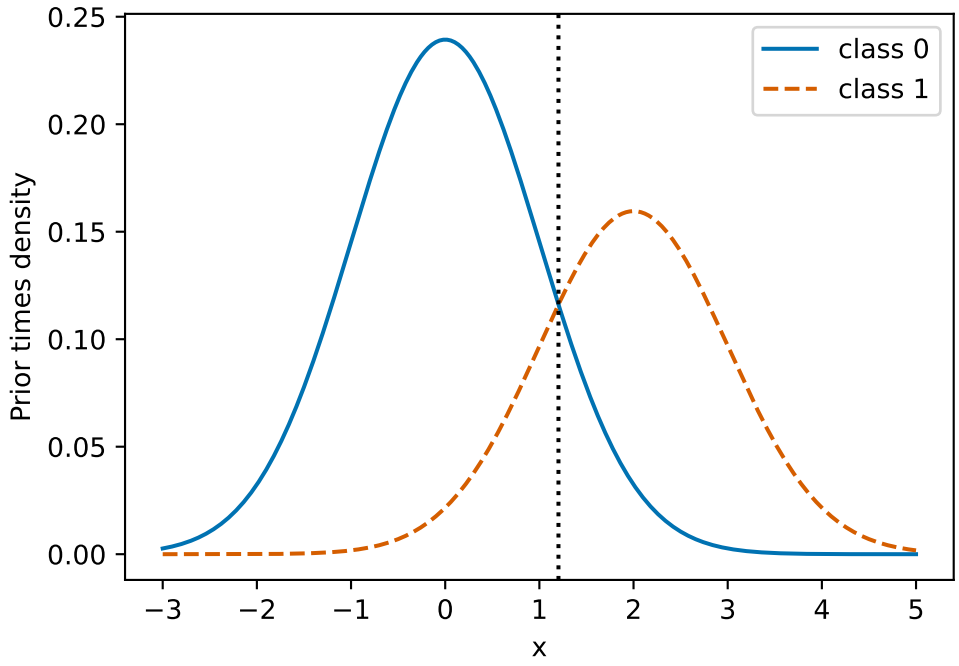


(a)

1.0.6 Python

```
import matplotlib.pyplot as plt

# weight each class density by its prior, on a grid of feature values
x_bc = np.linspace(-3, 5, 401)
w0_bc = prior_bc[0] * norm.pdf(x_bc, mean_bc[0], 1)
w1_bc = prior_bc[1] * norm.pdf(x_bc, mean_bc[1], 1)
fig, ax = plt.subplots(figsize=(5, 3.5), layout="constrained")
ax.plot(x_bc, w0_bc, color="#0072B2", linestyle="-", label="class 0");
ax.plot(x_bc, w1_bc, color="#D55E00", linestyle="--", label="class 1");
# the Bayes classifier switches class where the two curves cross
ax.axvline(cut_bc, color="black", linestyle=":");
ax.set_xlabel("x");
ax.set_ylabel("Prior times density");
ax.legend();
plt.show()
```



(b)

Figure 2: The joint density[h] $\mathbb{p}(Y = k, X = x)$ of class[i] k and the feature value x, for each class k of Example 1.2, plotted against x. The joint density equals $\pi_k f_k(x)$, the class's prior[j] times the class's density[k]. The dotted line marks where the Bayes classifier switches classes.

[a] https://morrison-lab.github.io/pds/random-variables.html#def-normal
[b] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[c] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance
[d] https://morrison-lab.github.io/sds/classification.html#def-class
[e] https://morrison-lab.github.io/sds/classification.html#def-class
[f] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[g] https://morrison-lab.github.io/sds/classification.html#def-class
[h] https://morrison-lab.github.io/pds/random-variables.html#def-joint-density-mass
[i] https://morrison-lab.github.io/sds/classification.html#def-class
[j] https://morrison-lab.github.io/sds/bayesian-inference.html#def-prior
[k] https://morrison-lab.github.io/pds/random-variables.html#def-pdf

Definition 1.2 (Generative classifier (plug-in Bayes classifier))**.** A **generative classifier** is a classifier built on a generative model. It models the joint distribution[a] of the label and the features through a class[b] prior π_k and a class-conditional density $f_k(\underset{\sim}{x})$ for each class k, estimates[c] both from training data [4], and classifies each $\underset{\sim}{x}$ by the largest estimated[d] posterior probability

$$\widehat{\Pr}[Y = k \mid \underset{\sim}{X} = \underset{\sim}{x}] = \frac{\hat{\pi}_k \hat{f}_k(\underset{\sim}{x})}{\sum_{l=1}^{K} \hat{\pi}_l \hat{f}_l(\underset{\sim}{x})},$$

the formula of Theorem 1.1 with the estimates $\hat{\pi}_k$ and $\hat{f}_k$ in place of the true priors and densities. Because it puts estimates into the Bayes classifier, it is also called a **plug-in Bayes classifier**.

[a] https://morrison-lab.github.io/pds/random-variables.html#def-joint-distribution
[b] https://morrison-lab.github.io/sds/classification.html#def-class
[c] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[d] https://morrison-lab.github.io/sds/estimation.html#def-estimate

Example 1.3 (A Gaussian generative classifier for the stock market)**.**
Return to the `Smarket` data of the Smarket fit exercise. Model the previous day's return, `Lag1`, within each direction as a Gaussian distribution[a], fitted to the days before 2005. Then classify a day whose previous return was 1%.

1.0.7 R

```
da <- readr::read_csv("data/smarket.csv", show_col_types = FALSE)
da_train <- da[da$Year < 2005, ]

direction_counts <- table(da_train$Direction)
prior <- prop.table(direction_counts)
class_mean <- tapply(da_train$Lag1, da_train$Direction, mean)
class_sd <- tapply(da_train$Lag1, da_train$Direction, sd)

x0 <- 1
joint <- prior * dnorm(x0, class_mean, class_sd)
posterior <- joint / sum(joint)
# nolint start: undesirable_function_linter. A matrix.
gen_table <- cbind(prior, mean = class_mean, sd = class_sd, posterior)
# nolint end
round(gen_table, 4)
```

```
     prior    mean     sd posterior
Down 0.492  0.0428 1.2274    0.5059
Up   0.508 -0.0395 1.2317    0.4941
```

1.0.8 Python

```
import numpy as np
import pandas as pd
from scipy.stats import norm

da_py = pd.read_csv("data/smarket.csv")
da_train_py = da_py[da_py["Year"] < 2005]

by_dir_py = da_train_py.groupby("Direction")["Lag1"]
prior_py = by_dir_py.size() / len(da_train_py)
mean_py = by_dir_py.mean()
sd_py = by_dir_py.std()

x0_py = 1.0
joint_py = prior_py * norm.pdf(x0_py, mean_py, sd_py)
posterior_py = joint_py / joint_py.sum()
print(pd.DataFrame({"prior": prior_py, "mean": mean_py, "sd": sd_py,
                    "posterior": posterior_py}).round(4))
```

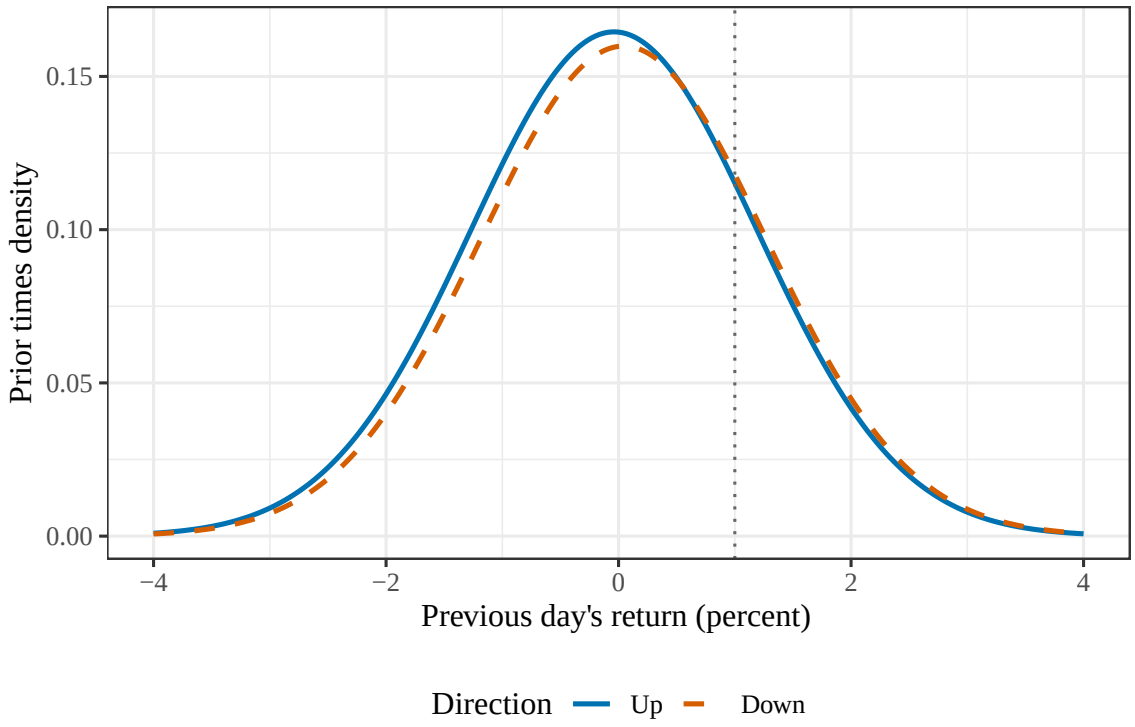
```
           prior    mean      sd  posterior
Direction
Down       0.492  0.0428  1.2274     0.5059
Up         0.508 -0.0395  1.2317     0.4941
```

The classes are nearly balanced, with 50.8% of the training days up. The mean[b] of `Lag1` is -0.040 on days that went up and 0.043 on days that went down, and the standard deviations are 1.232 and 1.227, almost equal. A day that follows a 1% return has posterior probability 0.494 of going up, so the classifier predicts Down, with a posterior close to one half. The two class-conditional densities overlap almost completely, which is the same difficulty that the Smarket probabilities figure shows for logistic regression.

[a] https://morrison-lab.github.io/pds/random-variables.html#def-normal
[b] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean

1.0.9 R

```
library(ggplot2)
gcf_grid <- seq(-4, 4, length.out = 241)
gcf_curves <- do.call(rbind, lapply(names(class_mean), function(dir) {
  data.frame(
    lag1 = gcf_grid,
    weighted = prior[[dir]] * dnorm(gcf_grid, class_mean[[dir]], class_sd[[dir]]),
    direction = dir
  )
}))
gcf_curves$direction <- factor(gcf_curves$direction, levels = c("Up", "Down"))
ggplot(gcf_curves) +
  aes(lag1, weighted, colour = direction, linetype = direction) +
  geom_line(linewidth = 0.9) +
  geom_vline(xintercept = x0, linetype = "dotted", colour = "grey40") +
  scale_colour_manual(values = c(Up = "#0072B2", Down = "#D55E00")) +
  scale_linetype_manual(values = c(Up = "solid", Down = "dashed")) +
  labs(
    x = "Previous day's return (percent)", y = "Prior times density",
    colour = "Direction", linetype = "Direction"
  )
```

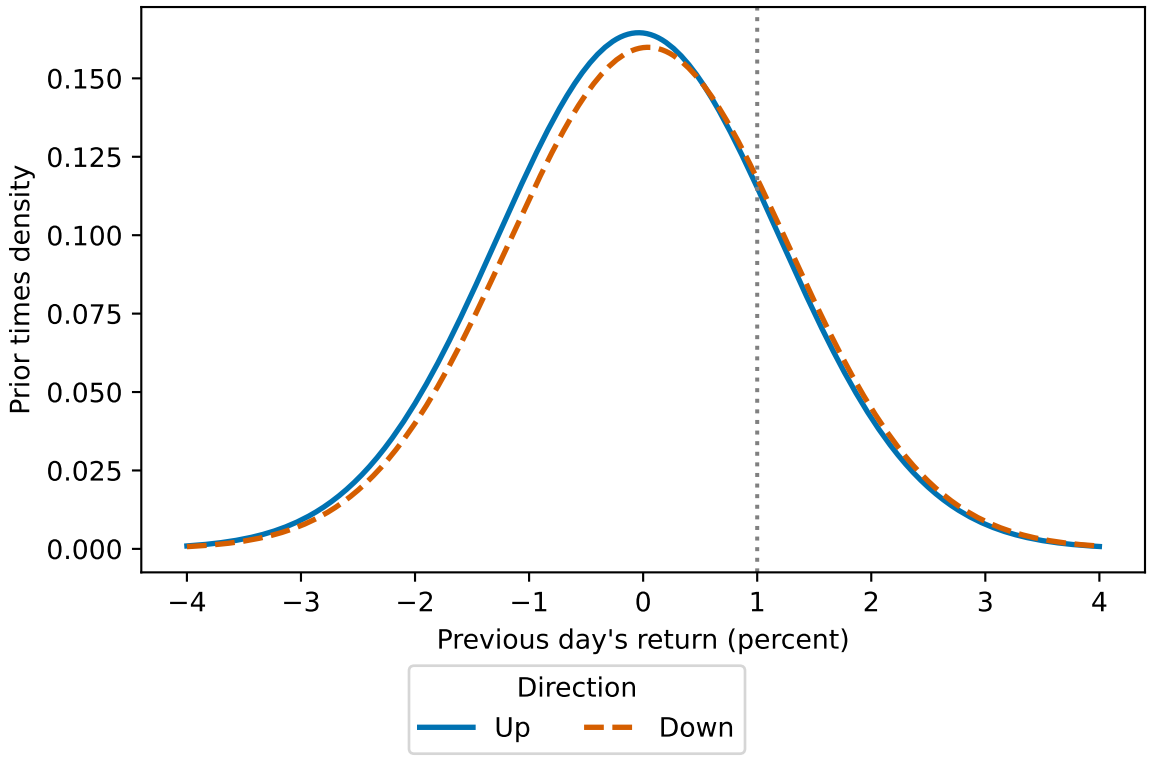


(a)

1.0.10 Python

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import norm

grid_gcf = np.linspace(-4, 4, 241)
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
for direction, colour, style in [("Up", "#0072B2", "-"), ("Down", "#D55E00", "--")]:
    ax.plot(grid_gcf, prior_py[direction] * norm.pdf(grid_gcf, mean_py[direction], sd_py[direction]),
            color=colour, linestyle=style, linewidth=2, label=direction);
ax.axvline(x0_py, color="grey", linestyle=":");
ax.set_xlabel("Previous day's return (percent)");
ax.set_ylabel("Prior times density");
fig.legend(title="Direction", loc="outside lower center", ncol=2);
plt.show()
```



(b)

Figure 3: Each class's prior times its density for the previous day's return, from Example 1.3. At a return of 1% (dotted line) the two curves almost coincide, so the classifier cannot separate the classes.

> **Remark (A class-conditional density can be a PMF)**
>
> *Remark* 1.1 (A class-conditional density can be a PMF). When the features are discrete[a], such as counts or qualitative features, the class-conditional density[b] $f_k(\underset{\sim}{x})$ of Definition 1.2 is the PMF[c] $\Pr[\underset{\sim}{X} = \underset{\sim}{x} \mid Y = k]$ [5, p. 146]. The word "density" still fits: a PMF is a density with respect to counting measure[d], just as a PDF[e] is a density with respect to Lebesgue measure[f], so PMFs and PDFs differ only in the reference measure[g].
>
> [a] https://morrison-lab.github.io/pds/random-variables.html#def-discrete-rv
> [b] https://morrison-lab.github.io/pds/random-variables.html#def-density-measure

[c] https://morrison-lab.github.io/pds/random-variables.html#def-pmf
[d] https://morrison-lab.github.io/pds/random-variables.html#prp-pmf-density-counting
[e] https://morrison-lab.github.io/pds/random-variables.html#def-pdf
[f] https://morrison-lab.github.io/mds/measures-measure-spaces.html#def-lebesgue-measure
[g] https://morrison-lab.github.io/pds/random-variables.html#rem-pmf-pdf-reference-measures

Remark (A discriminative classifier for the Smarket training days)

Remark 1.2 (A discriminative classifier for the Smarket training days). Logistic regression is a discriminative model: it models $\Pr[Y = k \mid \underset{\sim}{X} = \underset{\sim}{x}]$ directly and does not model the distribution[a] of the features. For comparison with Example 1.3, the code fits a logistic regression of the direction on `Lag1` to the `Smarket` training days before 2005 and computes the probability of an up day after a 1% return.

1.0.11 R

```
disc_fit <- glm(
  I(Direction == "Up") ~ Lag1,
  family = binomial, data = da_train
)
disc_day <- tibble::tibble(Lag1 = x0)
disc_prob <- predict(disc_fit, disc_day, type = "response")
disc_coef <- coef(disc_fit)
disc_out <- c(disc_coef, p_up = disc_prob[[1]])
round(disc_out, 4)
```

```
(Intercept)        Lag1        p_up
     0.0322     -0.0546      0.4944
```

1.0.12 Python

```
import statsmodels.api as sm
import statsmodels.formula.api as smf

da_train_py = da_train_py.assign(up=(da_train_py["Direction"] == "Up").astype(int))
disc_py = smf.glm("up ~ Lag1", data=da_train_py, family=sm.families.Binomial()).fit()
print(disc_py.params.round(4).to_dict())
```

```
{'Intercept': 0.0322, 'Lag1': -0.0546}
```

```
print("p_up:", round(float(disc_py.predict(pd.DataFrame({"Lag1": [x0_py]})).iloc[0]), 4))
```

```
p_up: 0.4944
```

The fitted model has an intercept[b] and a slope[c], and nothing else: it has no prior, no class[d] mean[e] and no class variance[f], because it does not model how `Lag1` is distributed. After a 1% return it gives an up day probability 0.4944, close to the 0.4941 of the generative classifier.

[a] https://morrison-lab.github.io/pds/random-variables.html#def-cdf
[b] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[c] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope
[d] https://morrison-lab.github.io/sds/classification.html#def-class
[e] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[f] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance

2 Linear discriminant analysis

Exercise 2.1 (Gaussian classes with one covariance matrix)**.** Let $\underset{\sim}{X} \in \mathbb{R}^p$, and suppose that within class[a] k the features are multivariate normal[b], $\underset{\sim}{X} \mid Y = k \sim \mathrm{N}(\vec{\mu}_k, \Sigma)$, with a mean[c] $\vec{\mu}_k$ for each class and *one* covariance matrix[d] Σ shared by all of them. The density is

$$f_k(\underset{\sim}{x}) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}\vec{\mathrm{d}}_k^{\top}(\Sigma)^{-1}\vec{\mathrm{d}}_k\right),$$

where $\vec{\mathrm{d}}_k \overset{\text{def}}{=} (\underset{\sim}{x} - \vec{\mu}_k)$ is the deviation[e] of $\underset{\sim}{x}$ from the class mean.
Take the logarithm of $\pi_k f_k(\underset{\sim}{x})$, and expand the quadratic form.

[a] https://morrison-lab.github.io/sds/classification.html#def-class
[b] https://morrison-lab.github.io/pds/distributions.html#def-mvn
[c] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[d] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x
[e] https://morrison-lab.github.io/pds/variance-covariance.html#def-deviation

Solution

Solution 2.1. Assume that Σ is positive definite[a] and every $\pi_k > 0$. Then $(\Sigma)^{-1}$ exists, because a positive definite matrix is invertible[b], and $|\Sigma| > 0$, so $\log|\Sigma|$ is defined. Every normal[c] density is positive, so every product $\pi_k f_k(\underset{\sim}{x})$ is positive and has a logarithm. Write $q_k(\underset{\sim}{x}) = (\underset{\sim}{x} - \vec{\mu}_k)^{\top}(\Sigma)^{-1}(\underset{\sim}{x} - \vec{\mu}_k)$ for the quadratic form. Taking the logarithm of the product turns it into a sum:

$$\begin{aligned}
\log(\pi_k f_k(\underset{\sim}{x})) &= \log \pi_k + \log f_k(\underset{\sim}{x}) && \text{(log of a product)} \\
&= \log \pi_k + \log\left(\frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}q_k(\underset{\sim}{x})\right)\right) && \text{(the Gaussian density)} \\
&= \log \pi_k + \log \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} + \log\exp\left(-\frac{1}{2}q_k(\underset{\sim}{x})\right) && \text{(log of a product)} \\
&= \log \pi_k - \log\left((2\pi)^{p/2}|\Sigma|^{1/2}\right) + \log\exp\left(-\frac{1}{2}q_k(\underset{\sim}{x})\right) && \text{(log of a reciprocal)} \\
&= \log \pi_k - \log\left((2\pi)^{p/2}|\Sigma|^{1/2}\right) - \frac{1}{2}q_k(\underset{\sim}{x}) && \text{(log of an exponential)} \\
&= \log \pi_k - \log(2\pi)^{p/2} - \log|\Sigma|^{1/2} - \frac{1}{2}q_k(\underset{\sim}{x}) && \text{(log of a product)} \\
&= \log \pi_k - \frac{p}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}q_k(\underset{\sim}{x}) && \text{(log of a power).}
\end{aligned}$$

Next, expand the quadratic form. Because $(\Sigma)^{-1}$ is symmetric[d], the two cross terms are equal: $\vec{\mu}_k^{\top}(\Sigma)^{-1}\underset{\sim}{x} = \underset{\sim}{x}^{\top}(\Sigma)^{-1}\vec{\mu}_k$.

$$\begin{aligned}
q_k(\underset{\sim}{x}) &= \underset{\sim}{x}^{\top}(\Sigma)^{-1}\underset{\sim}{x} - \underset{\sim}{x}^{\top}(\Sigma)^{-1}\vec{\mu}_k - \vec{\mu}_k^{\top}(\Sigma)^{-1}\underset{\sim}{x} + \vec{\mu}_k^{\top}(\Sigma)^{-1}\vec{\mu}_k && \text{(distribute)} \\
&= \underset{\sim}{x}^{\top}(\Sigma)^{-1}\underset{\sim}{x} - 2\,\underset{\sim}{x}^{\top}(\Sigma)^{-1}\vec{\mu}_k + \vec{\mu}_k^{\top}(\Sigma)^{-1}\vec{\mu}_k && \text{(symmetry of } (\Sigma)^{-1}\text{)}.
\end{aligned}$$

Substituting the expansion into the logarithm gives

$$\begin{aligned}
\log(\pi_k f_k(\underset{\sim}{x})) &= \log \pi_k - \frac{p}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}\left(\underset{\sim}{x}^{\top}(\Sigma)^{-1}\underset{\sim}{x} - 2\,\underset{\sim}{x}^{\top}(\Sigma)^{-1}\vec{\mu}_k + \vec{\mu}_k^{\top}(\Sigma)^{-1}\vec{\mu}_k\right) && \text{(substitute } q_k(\underset{\sim}{x})\text{)} \\
&= \log \pi_k - \frac{p}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}\underset{\sim}{x}^{\top}(\Sigma)^{-1}\underset{\sim}{x} + \underset{\sim}{x}^{\top}(\Sigma)^{-1}\vec{\mu}_k - \frac{1}{2}\vec{\mu}_k^{\top}(\Sigma)^{-1}\vec{\mu}_k && \text{(distribute } -\tfrac{1}{2}\text{)}.
\end{aligned}$$

[a] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-positive-definite
[b] https://morrison-lab.github.io/mds/linear-algebra-diagonalization.html#thm-pd-invertible
[c] https://morrison-lab.github.io/pds/random-variables.html#def-normal
[d] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-symmetric-matrix

Exercise 2.2 (Gaussian classes with one covariance matrix (part 2))**.** Use the setup of Exercise 2.1. Drop every term that is the same for all classes[a], and denote the remaining expression[b] by $\delta_k(\underset{\sim}{x})$. Why does dropping those terms preserve $\arg\max_k \pi_k f_k(\underset{\sim}{x})$?

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

Solution

Solution 2.2. Three terms, $-\frac{p}{2}\log(2\pi)$, $-\frac{1}{2}\log|\Sigma|$ and $-\frac{1}{2}\underset{\sim}{x}^\top(\Sigma)^{-1}\underset{\sim}{x}$, do not depend on k. Adding the same number to every class's score does not change which score is largest, so they can be dropped. The score that remains is

$$\delta_k(\underset{\sim}{x}) = \underset{\sim}{x}^\top(\Sigma)^{-1}\vec{\mu}_k - \frac{1}{2}\vec{\mu}_k^\top(\Sigma)^{-1}\vec{\mu}_k + \log\pi_k.$$

Exercise 2.3 (Gaussian classes with one covariance matrix (part 3))**.** Use the setup of Exercise 2.1. Is $\delta_k(\underset{\sim}{x})$ an affine function of $\underset{\sim}{x}$, a linear function[a] plus a constant[b]?

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

Solution

Solution 2.3. Yes: $\delta_k(\underset{\sim}{x})$ is an affine function of $\underset{\sim}{x}$. The quadratic term $\underset{\sim}{x}^\top(\Sigma)^{-1}\underset{\sim}{x}$ is the same in every class and was dropped, which is possible only because the covariance matrix[a] is shared. What remains is the dot product[b] of $\underset{\sim}{x}$ with the fixed vector $(\Sigma)^{-1}\vec{\mu}_k$, plus the term $-\frac{1}{2}\vec{\mu}_k^\top(\Sigma)^{-1}\vec{\mu}_k + \log\pi_k$, which does not depend on $\underset{\sim}{x}$. The "linear" in linear discriminant analysis refers to this affine form, whose decision boundaries are hyperplanes.

[a] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x
[b] https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-dot-product

Definition 2.1 (Linear discriminant function)**.** For class[a] k with prior π_k and mean[b] $\vec{\mu}_k$, and a covariance matrix[c] Σ shared by all classes, the **linear discriminant function** is

$$\delta_k(\underset{\sim}{x}) = \underset{\sim}{x}^\top(\Sigma)^{-1}\vec{\mu}_k - \frac{1}{2}\vec{\mu}_k^\top(\Sigma)^{-1}\vec{\mu}_k + \log\pi_k.$$

[a] https://morrison-lab.github.io/sds/classification.html#def-class
[b] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[c] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x

Example 2.1 (The linear discriminant function with one feature)**.** Let $p = 1$, with class[a] means[b] $\vec{\mu}_0 = 0$ and $\vec{\mu}_1 = 2$, shared variance[c] $\Sigma = 1$, and priors $\pi_0 = 0.6$ and $\pi_1 = 0.4$. Evaluate the linear discriminant function of each class at $\underset{\sim}{x} = 1.2$.

```
ldf_m <- c(0, 2)
ldf_pi <- c(0.6, 0.4)
ldf_x <- 1.2
ldf_delta <- ldf_x * ldf_m - ldf_m^2 / 2 + log(ldf_pi)
ldf_delta
```

```
[1] -0.5108256 -0.5162907
```

With one feature and $\Sigma = 1$, $\delta_k(\underset{\sim}{x}) = \underset{\sim}{x}\vec{\mu}_k - \vec{\mu}_k^2/2 + \log\pi_k$, so $\delta_0(1.2)$ = -0.511 and $\delta_1(1.2)$ = -0.516.

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

Theorem 2.1 (The linear discriminant function)**.** *Suppose that within class*[a] *k the p features are multivariate Gaussian*[b]*, $\underset{\sim}{X} \mid Y = k \sim \mathrm{N}(\vec{\mu}_k, \Sigma)$, with one positive definite*[c] *covariance matrix*[d] *Σ shared by all classes*[e]*, and prior $\pi_k > 0$ for every class. Then the marginal density*[f] *of $\underset{\sim}{X}$ is positive everywhere:*

$$\mathbb{p}(\underset{\sim}{x}) > 0 \quad \textit{for every } \underset{\sim}{x} \in \mathbb{R}^p. \tag{1}$$

The class that maximizes $\pi_k f_k(\underset{\sim}{x})$ also maximizes the linear discriminant function $\delta_k(\underset{\sim}{x})$:

$$\arg\max_k \pi_k f_k(\underset{\sim}{x}) = \arg\max_k \delta_k(\underset{\sim}{x}). \tag{2}$$

Each δ_k is an affine function of $\underset{\sim}{x}$. More precisely, $\log(\pi_k f_k(\underset{\sim}{x})) = \delta_k(\underset{\sim}{x}) + c(\underset{\sim}{x})$, where $c(\underset{\sim}{x}) = -\frac{p}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}\underset{\sim}{x}^\top(\Sigma)^{-1}\underset{\sim}{x}$ is the same for every class.

[a] https://morrison-lab.github.io/sds/classification.html#def-class
[b] https://morrison-lab.github.io/pds/distributions.html#def-mvn
[c] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-positive-definite
[d] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x
[e] https://morrison-lab.github.io/sds/classification.html#def-class
[f] https://morrison-lab.github.io/pds/random-variables.html#def-marginal

Proof

Proof. By Solution 2.1, Solution 2.2 and Solution 2.3. □

Example 2.2 (The linear discriminant with one feature)**.** Let $p = 1$, with class[a] means[b] $\vec{\mu}_0 = 0$ and $\vec{\mu}_1 = 2$, shared variance[c] $\Sigma = 1$, and priors $\pi_0 = 0.6$ and $\pi_1 = 0.4$. Classify the point $\underset{\sim}{x} = 1.2$.

```
ld_m <- c(0, 2)
ld_pi <- c(0.6, 0.4)
ld_x <- 1.2
ld_delta <- ld_x * ld_m - ld_m^2 / 2 + log(ld_pi)
ld_joint <- ld_pi * dnorm(ld_x, ld_m, 1)
# nolint start: undesirable_function_linter. A matrix.
rbind(delta = ld_delta, pi_times_f = ld_joint)
```

```
                [,1]       [,2]
delta      -0.5108256 -0.5162907
pi_times_f  0.1165116  0.1158766
```

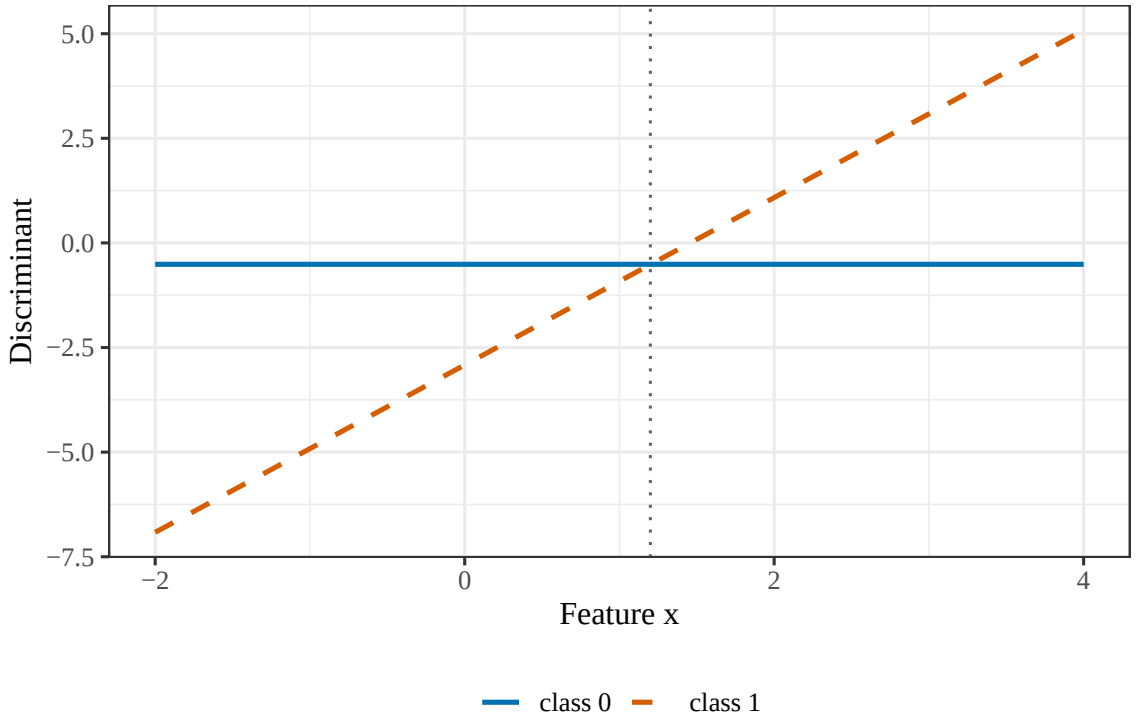
```
# nolint end
```

The discriminants are -0.511 and -0.516, so Theorem 2.1 predicts class 0. The products $\pi_k f_k(\underset{\sim}{x})$, 0.1165 and 0.1159, give the same class, 0. The point is close to the boundary: the two discriminants differ by only 0.0055.

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

2.0.1 R

```r
library(ggplot2)
ldf_grid <- seq(-2, 4, length.out = 121)
ldf_lines <- do.call(rbind, lapply(1:2, function(k) {
  data.frame(
    x = ldf_grid,
    delta = ldf_grid * ld_m[k] - ld_m[k]^2 / 2 + log(ld_pi[k]),
    class = paste("class", k - 1)
  )
}))
ggplot(ldf_lines) +
  aes(x, delta, colour = class, linetype = class) +
  geom_line(linewidth = 0.9) +
  geom_vline(xintercept = ld_x, linetype = "dotted", colour = "grey40") +
  scale_colour_manual(values = c("class 0" = "#0072B2", "class 1" = "#D55E00")) +
  scale_linetype_manual(values = c("class 0" = "solid", "class 1" = "dashed")) +
  labs(x = "Feature x", y = "Discriminant", colour = NULL, linetype = NULL)
```



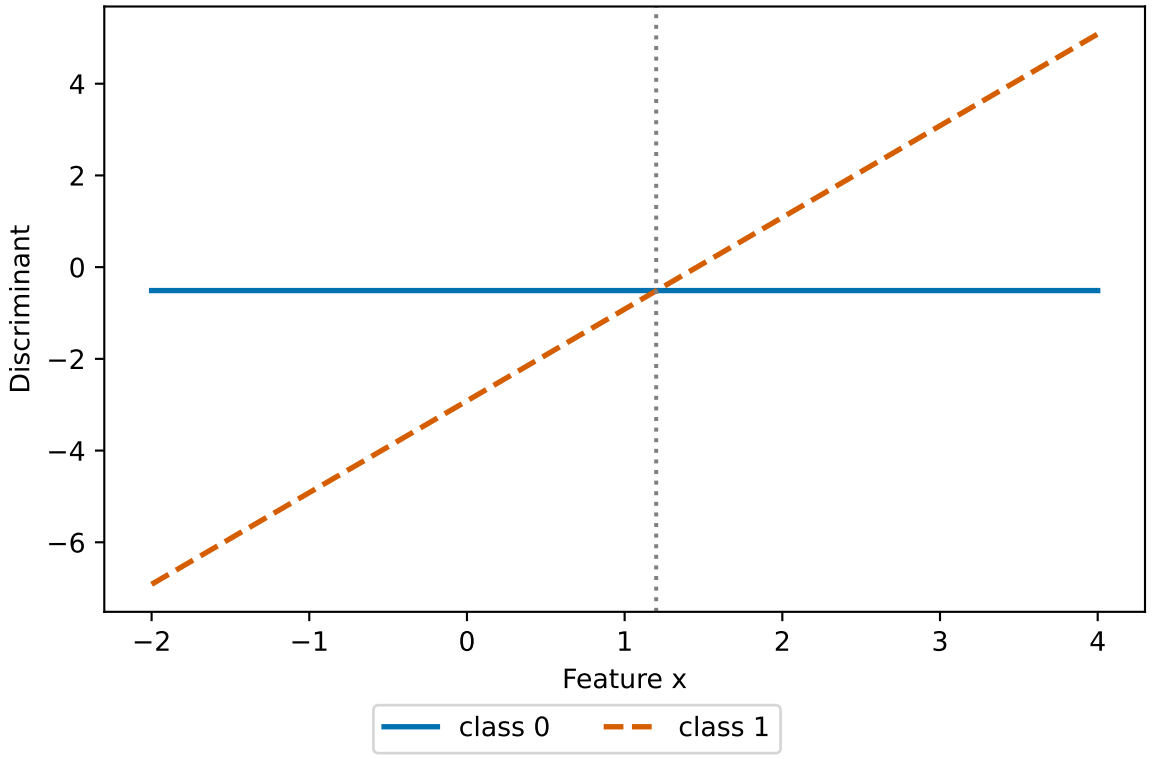
(a)

2.0.2 Python

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

grid_ldf = np.linspace(-2, 4, 121)
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
for k, colour, style in [(0, "#0072B2", "-"), (1, "#D55E00", "--")]:
    ax.plot(grid_ldf, grid_ldf * r.ld_m[k] - r.ld_m[k] ** 2 / 2 + np.log(r.ld_pi[k]),
            color=colour, linestyle=style, linewidth=2, label=f"class {k}");
ax.axvline(r.ld_x, color="grey", linestyle=":");
ax.set_xlabel("Feature x");
```

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



(b)

Figure 4: The linear discriminant functions of class 0 and class 1 plot as two straight-line graphs against the feature x, and the point $x = 1.2$ (dotted vertical line) is assigned to the class with the higher discriminant there (Example 2.2).

Definition 2.2 (Linear discriminant analysis (LDA))**. Linear discriminant analysis** is the generative classifier with a multivariate normal[a] density in each class[b] and one covariance matrix[c] Σ shared by all classes[d] [6]. It predicts the class with the largest linear discriminant function of Theorem 2.1, with each parameter replaced by its estimate[e] from n training points, n_k of them in class k:

- $\hat{\pi}_k = n_k/n$, the class proportion;
- $\hat{\underset{\sim}{\mu}}_k$, the average of the feature vectors in class k;
- $\hat{\Sigma} = \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i:y_i=k} \vec{\mathrm{d}}_i \vec{\mathrm{d}}_i^{\top}$, the pooled within-class covariance matrix, where $\vec{\mathrm{d}}_i \overset{\text{def}}{=} \left(\underset{\sim}{x}_i - \hat{\underset{\sim}{\mu}}_{y_i} \right)$ is the deviation[f] of training point i from the average of its class.

[a] https://morrison-lab.github.io/pds/distributions.html#def-mvn
[b] https://morrison-lab.github.io/sds/classification.html#def-class
[c] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x
[d] https://morrison-lab.github.io/sds/classification.html#def-class
[e] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[f] https://morrison-lab.github.io/pds/variance-covariance.html#def-deviation

Example 2.3 (LDA for one predictor)**.** Fit LDA to the `Smarket` days before 2005 with `Lag1` as the only predictor, so $p = 1$ and Σ is a single variance[a]. Compute the pooled variance and the discriminant functions at $x_0 = 1$ by hand, and compare the resulting posterior probabilities with a library's.

2.0.3 R

```
n_k <- table(da_train$Direction)
pooled <- sum((n_k - 1) * class_sd^2) / (sum(n_k) - 2)
delta <- x0 * class_mean / pooled - class_mean^2 / (2 * pooled) + log(prior)
exp_delta <- exp(delta)
by_hand <- exp_delta / sum(exp_delta)

lda_fit <- MASS::lda(Direction ~ Lag1, data = da_train)
lda_day <- tibble::tibble(Lag1 = x0)
library_post <- predict(lda_fit, lda_day)$posterior
pooled_shown <- round(pooled, 4)
cat("pooled variance:", pooled_shown, "\n")
```

```
pooled variance: 1.5119
```

```
# nolint start: undesirable_function_linter. A matrix.
post_table <- rbind(by_hand = by_hand, library = library_post[1, ])
# nolint end
round(post_table, 4)
```

```
          Down     Up
by_hand 0.5056 0.4944
library 0.5056 0.4944
```

2.0.4 Python

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

n_k_py = by_dir_py.size()
pooled_py = ((n_k_py - 1) * sd_py**2).sum() / (n_k_py.sum() - 2)
delta_py = x0_py * mean_py / pooled_py - mean_py**2 / (2 * pooled_py) + np.log(prior_py)
by_hand_py = np.exp(delta_py) / np.exp(delta_py).sum()

lda_py = LinearDiscriminantAnalysis().fit(da_train_py[["Lag1"]], da_train_py["Direction"])
library_py = lda_py.predict_proba(pd.DataFrame({"Lag1": [x0_py]}))[0]
print("pooled variance:", round(float(pooled_py), 4))
```

```
pooled variance: 1.5119
```

```
print("by hand:", by_hand_py.round(4).to_dict())
```

```
by hand: {'Down': 0.5056, 'Up': 0.4944}
```

```
print("library:", dict(zip(lda_py.classes_.tolist(), library_py.round(4).tolist())))
```

```
library: {'Down': 0.5056, 'Up': 0.4944}
```

The pooled variance is 1.512, between the two class variances 1.507 and 1.517. The two discriminant functions differ by only 0.022 at x_0. The hand computation and the library give the same posteriors to four decimal places in both languages. With almost a thousand training days, dividing the pooled sum by n instead of $n - K$ changes them only in the fifth decimal place. The posterior probability of Up is 0.4944, close to the 0.4941 of Example 1.3. The small difference from that example's posterior comes from replacing the two class variances by their pool.

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

Remark (The name "discriminant analysis")

Remark 2.1 (The name "discriminant analysis"). Despite its name, discriminant analysis is a generative method, not a discriminative one. It models the distribution[a] of the features within each class[b], $f_k(\underset{\sim}{x})$,

and the class proportions[c] π_k, and classifies by Bayes' rule[d]. Logistic regression, in contrast, models $\Pr[Y = k \mid \underset{\sim}{X} = \underset{\sim}{x}]$ directly. The book places discriminant analysis in its section on generative models for classification [5, Sec. 4.4]. The "discriminant" in the name refers to the discriminant functions[e] $\delta_k(\underset{\sim}{x})$ of Theorem 2.1, which the classifier[f] compares to choose a class, a use of the word that goes back to Fisher's linear discriminant.

[a] https://morrison-lab.github.io/pds/random-variables.html#def-cdf
[b] https://morrison-lab.github.io/sds/classification.html#def-class
[c] https://morrison-lab.github.io/sds/classification.html#def-prevalence
[d] https://morrison-lab.github.io/pds/probability-basics.html#thm-bayes
[e] https://morrison-lab.github.io/mcs/functions.html#def-function
[f] https://morrison-lab.github.io/sds/classification.html#def-classification-rule

Remark (The divisor of the pooled covariance matrix)

Remark 2.2 (The divisor of the pooled covariance matrix). The divisor $n - K$ in Definition 2.2 makes $\hat{\Sigma}$ an unbiased estimator[a] of Σ. Within class[b] k, the sum of the outer products[c] of the deviations[d] from $\hat{\underset{\sim}{\mu}}_k$ has expectation[e] $(n_k - 1)\Sigma$, because estimating the class mean[f] uses up one degree of freedom. Summing over the K classes[g] gives expectation $(n - K)\Sigma$. The maximum likelihood estimator[h] divides the same sum by n instead. Software differs: R's `MASS::lda()` divides by $n - K$ [7, Ch. 12]. scikit-learn's `LinearDiscriminantAnalysis` depends on its solver and version. Its `lsqr` and `eigen` solvers divide by n in versions 1.8.0 and 1.9.1. Its default `svd` solver divides by $n - K$ in version 1.8.0 and by n in version 1.9.1. With hundreds of training points the two choices give nearly the same classifier.

[a] https://morrison-lab.github.io/sds/estimation.html#def-estimator
[b] https://morrison-lab.github.io/sds/classification.html#def-class
[c] https://morrison-lab.github.io/mds/linear-algebra-inner-products.html#def-outer-product
[d] https://morrison-lab.github.io/pds/variance-covariance.html#def-deviation
[e] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[f] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean
[g] https://morrison-lab.github.io/sds/classification.html#def-class
[h] https://morrison-lab.github.io/sds/intro-MLEs.html#def-mle

Remark (Fisher's linear discriminant)

Remark 2.3 (Fisher's linear discriminant). "Fisher's linear discriminant" names a closely related method that finds the direction that best separates the class means[a], relative to the spread within the classes[b]. In the two-class case its direction is proportional to the LDA coefficient vector $(\Sigma)^{-1}(\vec{\mu}_1 - \vec{\mu}_0)$ [8, Sec. 4.3.3].

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

Exercise 2.4 (The log odds of LDA)**.** Consider two classes, $Y \in \{0, 1\}$, under the LDA model. Compare with the log-odds exercise, where the log odds were *assumed* linear[a].
Express $\log \frac{\Pr[Y=1|\underset{\sim}{X}=\underset{\sim}{x}]}{\Pr[Y=0|\underset{\sim}{X}=\underset{\sim}{x}]}$ in terms of the linear discriminant functions δ_1 and δ_0.

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

Solution

Solution 2.4. Both posteriors share the denominator $\sum_l \pi_l f_l(\underset{\sim}{x})$ of Theorem 1.1, which cancels in their ratio. By Theorem 2.1, $\log(\pi_k f_k(\underset{\sim}{x})) = \delta_k(\underset{\sim}{x}) + c(\underset{\sim}{x})$, where $c(\underset{\sim}{x})$ collects the terms that the theorem drops and is the same for both classes[a]. So

$$
\begin{aligned}
\log \frac{\Pr[Y=1 \mid \underset{\sim}{X}=\underset{\sim}{x}]}{\Pr[Y=0 \mid \underset{\sim}{X}=\underset{\sim}{x}]} &= \log \frac{\pi_1 f_1(\underset{\sim}{x})}{\pi_0 f_0(\underset{\sim}{x})} && \text{(the denominators cancel)} \\
&= \log(\pi_1 f_1(\underset{\sim}{x})) - \log(\pi_0 f_0(\underset{\sim}{x})) && \text{(log of a quotient)} \\
&= (\delta_1(\underset{\sim}{x}) + c(\underset{\sim}{x})) - (\delta_0(\underset{\sim}{x}) + c(\underset{\sim}{x})) && \text{(the theorem, for each class)} \\
&= \delta_1(\underset{\sim}{x}) - \delta_0(\underset{\sim}{x}) && (c(\underset{\sim}{x}) \text{ cancels}).
\end{aligned}
$$

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

Exercise 2.5 (The log odds of LDA (part 2))**.** Use the setup of Exercise 2.4.
Show that the log odds[a] equal $\beta_0 + \vec{\beta} \cdot \underset{\sim}{x}$, and give β_0 and $\vec{\beta}$ in terms of the parameters of the LDA model.

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

Solution

Solution 2.5. Subtract the two discriminant functions of Theorem 2.1:

$$
\begin{aligned}
\delta_1(\underset{\sim}{x}) - \delta_0(\underset{\sim}{x}) &= \left(\underset{\sim}{x}^\top (\Sigma)^{-1}\vec{\mu}_1 - \frac{1}{2}\vec{\mu}_1^\top (\Sigma)^{-1}\vec{\mu}_1 + \log \pi_1\right) \\
&\quad - \left(\underset{\sim}{x}^\top (\Sigma)^{-1}\vec{\mu}_0 - \frac{1}{2}\vec{\mu}_0^\top (\Sigma)^{-1}\vec{\mu}_0 + \log \pi_0\right) && \text{(definition)} \\
&= \underset{\sim}{x}^\top (\Sigma)^{-1}\vec{\mu}_1 - \frac{1}{2}\vec{\mu}_1^\top (\Sigma)^{-1}\vec{\mu}_1 + \log \pi_1 \\
&\quad - \underset{\sim}{x}^\top (\Sigma)^{-1}\vec{\mu}_0 + \frac{1}{2}\vec{\mu}_0^\top (\Sigma)^{-1}\vec{\mu}_0 - \log \pi_0 && \text{(distribute } -\text{)} \\
&= \underset{\sim}{x}^\top (\Sigma)^{-1}\vec{\mu}_1 - \underset{\sim}{x}^\top (\Sigma)^{-1}\vec{\mu}_0 - \frac{1}{2}\left(\vec{\mu}_1^\top (\Sigma)^{-1}\vec{\mu}_1 - \vec{\mu}_0^\top (\Sigma)^{-1}\vec{\mu}_0\right) + \log \pi_1 - \log \pi_0 && \text{(regroup)} \\
&= \underset{\sim}{x}^\top (\Sigma)^{-1}(\vec{\mu}_1 - \vec{\mu}_0) - \frac{1}{2}\left(\vec{\mu}_1^\top (\Sigma)^{-1}\vec{\mu}_1 - \vec{\mu}_0^\top (\Sigma)^{-1}\vec{\mu}_0\right) + \log \pi_1 - \log \pi_0 && \text{(factor out } \underset{\sim}{x}^\top (\Sigma)^{-1}\text{)} \\
&= \underset{\sim}{x}^\top (\Sigma)^{-1}(\vec{\mu}_1 - \vec{\mu}_0) - \frac{1}{2}\left(\vec{\mu}_1^\top (\Sigma)^{-1}\vec{\mu}_1 - \vec{\mu}_0^\top (\Sigma)^{-1}\vec{\mu}_0\right) + \log \frac{\pi_1}{\pi_0} && \text{(log of a quotient)} \\
&= \underset{\sim}{x}^\top (\Sigma)^{-1}(\vec{\mu}_1 - \vec{\mu}_0) - \frac{1}{2}(\vec{\mu}_1 + \vec{\mu}_0)^\top (\Sigma)^{-1}(\vec{\mu}_1 - \vec{\mu}_0) + \log \frac{\pi_1}{\pi_0} && \text{(symmetry of } (\Sigma)^{-1}\text{)}.
\end{aligned}
$$

The last step uses the symmetry of $(\Sigma)^{-1}$: expanding $(\vec{\mu}_1 + \vec{\mu}_0)^\top (\Sigma)^{-1}(\vec{\mu}_1 - \vec{\mu}_0)$ gives four terms, and the cross terms $-\vec{\mu}_1^\top (\Sigma)^{-1}\vec{\mu}_0$ and $\vec{\mu}_0^\top (\Sigma)^{-1}\vec{\mu}_1$ are equal and opposite, so they cancel. Because $(\Sigma)^{-1}$ is symmetric[a], $\underset{\sim}{x}^\top (\Sigma)^{-1}(\vec{\mu}_1 - \vec{\mu}_0)$ is the dot product[b] of $\underset{\sim}{x}$ with $(\Sigma)^{-1}(\vec{\mu}_1 - \vec{\mu}_0)$. So the log odds[c] have the form $\beta_0 + \vec{\beta} \cdot \underset{\sim}{x}$, with

$$
\vec{\beta} = (\Sigma)^{-1}(\vec{\mu}_1 - \vec{\mu}_0), \qquad \beta_0 = \log \frac{\pi_1}{\pi_0} - \frac{1}{2}(\vec{\mu}_1 + \vec{\mu}_0)^\top \vec{\beta}.
$$

[a] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-symmetric-matrix
[b] https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-dot-product
[c] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds

Exercise 2.6 (The log odds of LDA (part 3))**.** Use the setup of Exercise 2.4.
What form does $\Pr[Y=1 \mid \underset{\sim}{X}=\underset{\sim}{x}]$ take under the LDA model?

Solution

Solution 2.6. The sigmoid-from-log-odds theorem turns a linear[a] log odds[b] into the probability, which gives

$$\Pr[Y = 1 \mid \underset{\sim}{X} = \underset{\sim}{x}] = \operatorname{expit}\big(\beta_0 + \vec{\beta} \cdot \underset{\sim}{x}\big).$$

This equation is the logistic regression model, with these particular coefficients.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-linear-function
[b] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds

Theorem 2.2 (LDA implies a logistic model)**.** *Under the two-class LDA model, with* Σ *positive definite*[a] *and* $\pi_0, \pi_1 > 0$*,*

$$\Pr[Y = 1 \mid \underset{\sim}{X} = \underset{\sim}{x}] = \operatorname{expit}\big(\beta_0 + \vec{\beta} \cdot \underset{\sim}{x}\big), \qquad \vec{\beta} = (\Sigma)^{-1}(\vec{\mu}_1 - \vec{\mu}_0), \quad \beta_0 = \log\frac{\pi_1}{\pi_0} - \frac{1}{2}(\vec{\mu}_1 + \vec{\mu}_0)^\top \vec{\beta}.$$

The decision boundary at the threshold $1/2$*, the set*[b] *where* $\beta_0 + \vec{\beta} \cdot \underset{\sim}{x} = 0$*, is a hyperplane with normal vector*[c] $\vec{\beta}$ *whenever* $\vec{\mu}_1 \neq \vec{\mu}_0$*, by the decision-boundary theorem.*

[a] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-positive-definite
[b] https://morrison-lab.github.io/mcs/sets.html#def-set
[c] https://morrison-lab.github.io/mds/linear-algebra-matrices.html#def-normal-vector

Proof

Proof. By Solution 2.4, Solution 2.5 and Solution 2.6. □

Example 2.4 (LDA and logistic regression on the same days)**.** For the one-predictor LDA fit of Example 2.3, compute β_0 and β_1 from Theorem 2.2, and compare them with the coefficients of a logistic regression of Up on `Lag1` fitted to the same days.

2.0.5 R

```
b1 <- (class_mean[["Up"]] - class_mean[["Down"]]) / pooled
mid <- (class_mean[["Up"]] + class_mean[["Down"]]) / 2
b0 <- log(prior[["Up"]] / prior[["Down"]]) - mid * b1

logit_fit <- glm(
  I(Direction == "Up") ~ Lag1,
  family = binomial, data = da_train
)
lda_coef <- c(b0, b1)
logit_coef <- coef(logit_fit)
# nolint start: undesirable_function_linter. A matrix.
rbind(LDA = lda_coef, logistic = logit_coef) |> round(4)
```

```
         (Intercept)    Lag1
LDA           0.0322 -0.0545
logistic      0.0322 -0.0546
```

```
# nolint end
```

2.0.6 Python

```
import statsmodels.formula.api as smf
import statsmodels.api as sm

b1_py = (mean_py["Up"] - mean_py["Down"]) / pooled_py
mid_py = (mean_py["Up"] + mean_py["Down"]) / 2
b0_py = np.log(prior_py["Up"] / prior_py["Down"]) - mid_py * b1_py

da_train_py = da_train_py.assign(up=(da_train_py["Direction"] == "Up").astype(int))
logit_py = smf.glm("up ~ Lag1", data=da_train_py, family=sm.families.Binomial()).fit()
print("LDA:     ", round(float(b0_py), 4), round(float(b1_py), 4))
```

```
LDA:      0.0322 -0.0545
```

```
print("logistic:", logit_py.params.round(4).tolist())
```

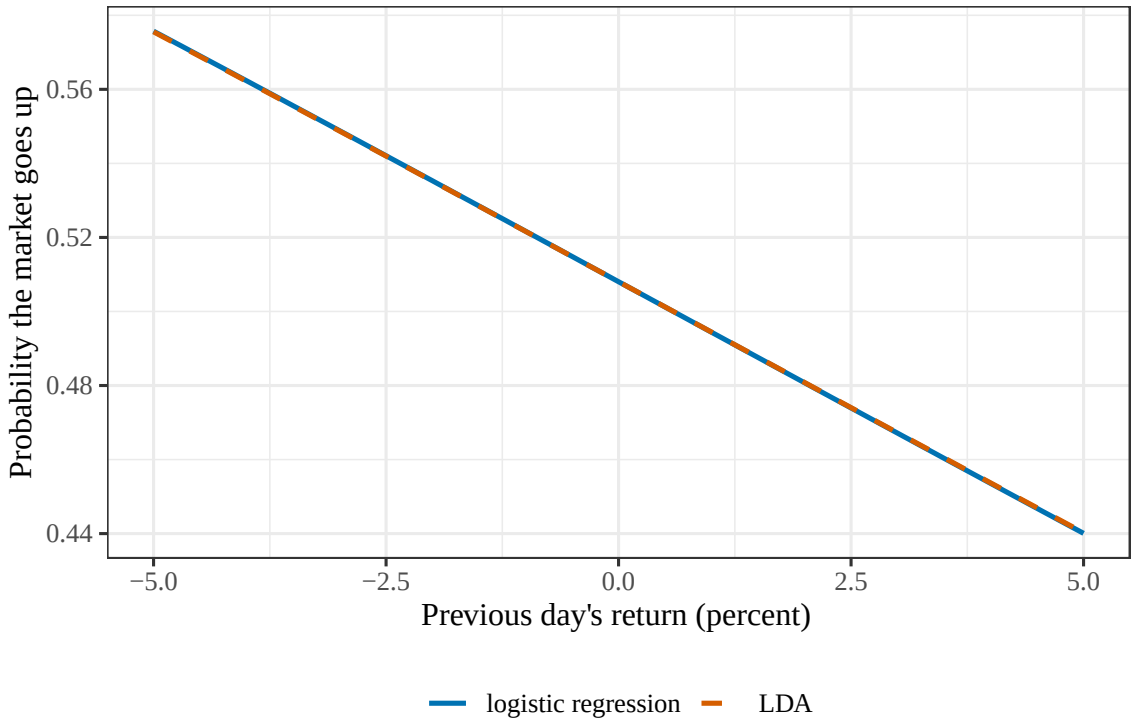
```
logistic: [0.0322, -0.0546]
```

The two fitted log-odds[a] lines nearly coincide. LDA gives a slope of -0.0545 and logistic regression -0.0546, a difference of 0.3% of the logistic slope, and the intercepts are 0.0322 and 0.0322. Both slopes[b] are negative and small: a rise of one percentage point in the previous day's return lowers the log odds of an up day by about 0.05. When the class[c] densities are normal[d] with a shared variance[e], which is the model LDA assumes, the two methods' estimates[f] converge to the same coefficients as the sample[g] grows [8, Sec. 4.4.5]. Their estimates can differ more when that model fails, for example for a strongly skewed predictor.

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds
[b] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope
[c] https://morrison-lab.github.io/sds/classification.html#def-class
[d] https://morrison-lab.github.io/pds/random-variables.html#def-normal
[e] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance
[f] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[g] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample

2.0.7 R

```r
library(ggplot2)
llo_grid <- seq(-5, 5, length.out = 201)
llo_curves <- data.frame(
  lag1 = rep(llo_grid, 2),
  prob = c(plogis(b0 + b1 * llo_grid), plogis(coef(logit_fit)[[1]] + coef(logit_fit)[[2]] * llo_grid)),
  model = rep(c("LDA", "logistic regression"), each = length(llo_grid))
)
llo_curves$model <- factor(llo_curves$model, levels = c("logistic regression", "LDA"))
ggplot(llo_curves) +
  aes(lag1, prob, colour = model, linetype = model) +
  geom_line(linewidth = 0.9) +
  scale_colour_manual(values = c("logistic regression" = "#0072B2", LDA = "#D55E00")) +
  scale_linetype_manual(values = c("logistic regression" = "solid", LDA = "dashed")) +
  labs(
    x = "Previous day's return (percent)", y = "Probability the market goes up",
    colour = NULL, linetype = NULL
  )
```



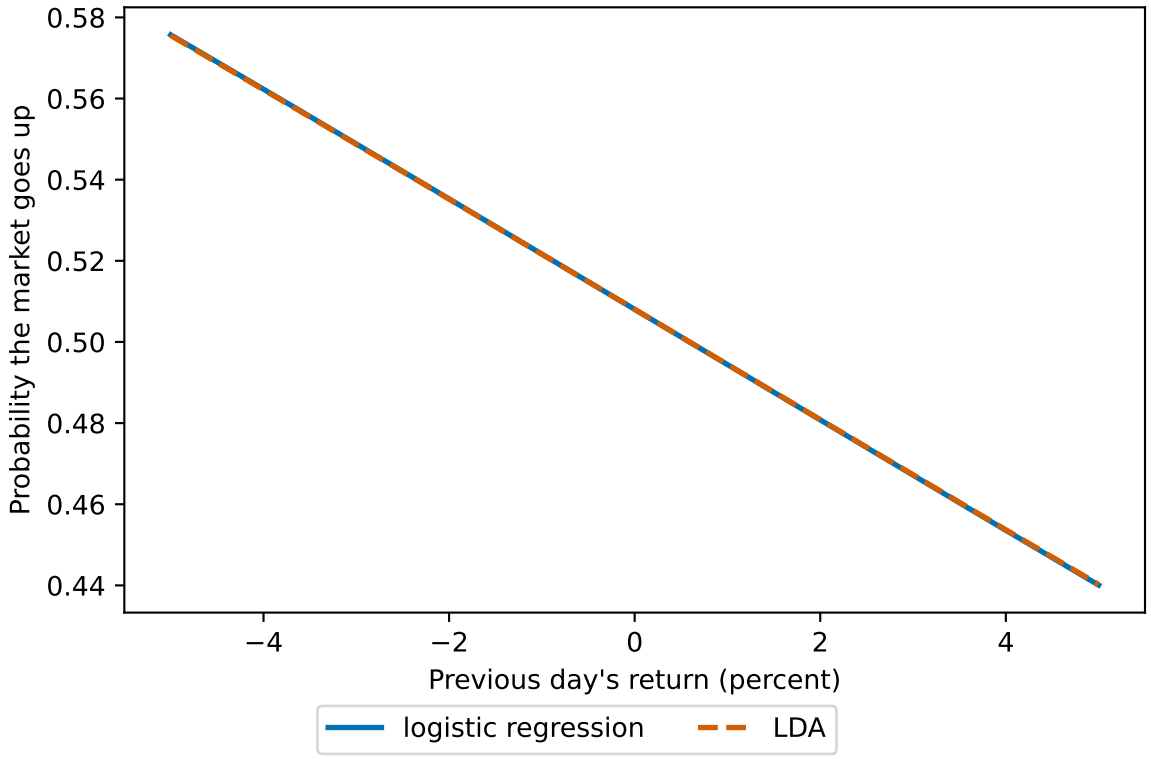
(a)

2.0.8 Python

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

grid_llo = np.linspace(-5, 5, 201)
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.plot(grid_llo, expit(logit_py.params.iloc[0] + logit_py.params.iloc[1] * grid_llo),
        color="#0072B2", linestyle="-", linewidth=2, label="logistic regression");
ax.plot(grid_llo, expit(b0_py + b1_py * grid_llo),
```

```
        color="#D55E00", linestyle="--", linewidth=2, label="LDA");
ax.set_xlabel("Previous day's return (percent)");
ax.set_ylabel("Probability the market goes up");
fig.legend(loc="outside lower center", ncol=2);
plt.show()
```



(b)

Figure 5: The fitted probability that the market goes up, from the two fits of Example 2.4. The two models give almost the same curve.

Simulated records of 10000 credit-card holders, each saying whether the card holder defaulted on their debt, make up the `Default` data [2, Sec. 4.1]. The records describe no actual card holders. They are in `data/default.csv`[9], one line per card holder, copied from `ISLR2`[10], the R package of *An Introduction to Statistical Learning* [2]. Each line gives:

- `default`: the outcome[11], `Yes` for a card holder who defaulted and `No` otherwise.
- `student`: `Yes` for a student and `No` otherwise.
- `balance`: the average[12] balance left on the card after the monthly payment, in dollars.
- `income`: the card holder's income, in dollars.

Dataset 1: The credit-card default (`Default`) data

2.0.9 R

```
default_preview <- readr::read_csv("data/default.csv", show_col_types = FALSE)
default_preview
```

```
# A tibble: 10,000 x 4
   default student balance income
   <chr>   <chr>     <dbl>  <dbl>
 1 No      No         730. 44362.
 2 No      Yes        817. 12106.
```

```
 3 No      No        1074. 31767.
 4 No      No         529. 35704.
 5 No      No         786. 38463.
 6 No      Yes        920.  7492.
 7 No      No         826. 24905.
 8 No      Yes        809. 17600.
 9 No      No        1161. 37469.
10 No      No           0  29275.
# i 9,990 more rows
```

```
dim(default_preview)
```

```
[1] 10000     4
```

```
skimr::skim_without_charts(default_preview)
```

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
default	0	1	2	3	0	2	0
student	0	1	2	3	0	2	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
balance	0	1	835.37	483.71	0.00	481.73	823.64	1166.31	2654.32
income	0	1	33516.98	13336.64	771.97	21340.46	34552.64	43807.73	73554.23

2.0.10 Python

```
import pandas as pd

pd.set_option("display.max_columns", None)
pd.set_option("display.width", 120)

default_preview = pd.read_csv("data/default.csv")
default_preview
```

```
     default student      balance        income
0          No      No   729.526495  44361.625074
1          No     Yes   817.180407  12106.134700
2          No      No  1073.549164  31767.138947
3          No      No   529.250605  35704.493935
4          No      No   785.655883  38463.495879
...       ...     ...          ...           ...
9995       No      No   711.555020  52992.378914
9996       No      No   757.962918  19660.721768
9997       No      No   845.411989  58636.156984
9998       No      No  1569.009053  36669.112365
9999       No     Yes   200.922183  16862.952321

[10000 rows x 4 columns]
```

```
default_preview.info()
```

```
<class 'pandas.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   default  10000 non-null  str
 1   student  10000 non-null  str
 2   balance  10000 non-null  float64
 3   income   10000 non-null  float64
dtypes: float64(2), str(2)
memory usage: 312.6 KB
```

```
print(default_preview.describe())
```

```
            balance        income
count  10000.000000  10000.000000
mean     835.374886  33516.981876
std      483.714985  13336.639563
min        0.000000    771.967729
25%      481.731105  21340.462903
50%      823.636973  34552.644802
75%     1166.308386  43807.729272
max     2654.322576  73554.233495
```

```
print(default_preview.isna().sum())
```

```
default    0
student    0
balance    0
income     0
dtype: int64
```

Example 2.5 (From class densities to a logistic curve)**.** Fit the one-predictor LDA model to the `Default` data, with `balance` as the predictor and a default (`Yes`) as class[a] 1. Compute each class's prior[b] times its normal density[c] over a grid of balances, the posterior[d] probability[e] of default from them by Theorem 1.1, and the logistic curve of Theorem 2.2.

2.0.11 R

```r
dflt <- readr::read_csv("data/default.csv", show_col_types = FALSE)
dflt_stats <- dflt |>
  dplyr::group_by(default) |>
  dplyr::summarise(n = dplyr::n(), mean = mean(balance), var = var(balance))
dflt_n <- sum(dflt_stats$n)
dflt_prior <- dflt_stats$n / dflt_n
names(dflt_prior) <- dflt_stats$default
dflt_mean <- dflt_stats$mean
names(dflt_mean) <- dflt_stats$default
dflt_class_sd <- sqrt(dflt_stats$var)
names(dflt_class_sd) <- dflt_stats$default
dflt_ss <- (dflt_stats$n - 1) * dflt_stats$var
dflt_pooled <- sum(dflt_ss) / (dflt_n - 2)
dflt_sd <- sqrt(dflt_pooled)

dflt_b1 <- (dflt_mean[["Yes"]] - dflt_mean[["No"]]) / dflt_pooled
dflt_mid <- (dflt_mean[["Yes"]] + dflt_mean[["No"]]) / 2
dflt_log_prior_ratio <- log(dflt_prior[["Yes"]] / dflt_prior[["No"]])
dflt_b0 <- dflt_log_prior_ratio - dflt_mid * dflt_b1
dflt_boundary <- -dflt_b0 / dflt_b1

dflt_grid <- seq(0, 2700, by = 10)
# Prior times density, on the log scale, for each class.
dflt_log_yes <- log(dflt_prior[["Yes"]]) +
  dnorm(dflt_grid, dflt_mean[["Yes"]], dflt_sd, log = TRUE)
dflt_log_no <- log(dflt_prior[["No"]]) +
  dnorm(dflt_grid, dflt_mean[["No"]], dflt_sd, log = TRUE)
# Bayes' rule: class 1's share of the two products.
dflt_bayes <- 1 / (1 + exp(dflt_log_no - dflt_log_yes))
dflt_sigmoid <- plogis(dflt_b0 + dflt_b1 * dflt_grid)

dflt_glm <- glm(I(default == "Yes") ~ balance, family = binomial, data = dflt)
dflt_logit <- coef(dflt_glm)
dflt_abs_gap <- abs(dflt_bayes - dflt_sigmoid)
dflt_gap <- max(dflt_abs_gap)
dflt_coef <- c(LDA = dflt_b1, logistic = dflt_logit[["balance"]])
signif(dflt_coef, 3)
```

```
     LDA logistic
  0.0046   0.0055
```

2.0.12 Python

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import statsmodels.formula.api as smf
from scipy.stats import norm
from scipy.special import expit

dflt_py = pd.read_csv("data/default.csv")
by_class_py = dflt_py.groupby("default")["balance"]
n_dflt_py = by_class_py.size()
prior_dflt_py = n_dflt_py / n_dflt_py.sum()
mean_dflt_py = by_class_py.mean()
pooled_dflt_py = ((n_dflt_py - 1) * by_class_py.var()).sum() / (n_dflt_py.sum() - 2)
sd_dflt_py = np.sqrt(pooled_dflt_py)

b1_dflt_py = (mean_dflt_py["Yes"] - mean_dflt_py["No"]) / pooled_dflt_py
mid_dflt_py = (mean_dflt_py["Yes"] + mean_dflt_py["No"]) / 2
b0_dflt_py = np.log(prior_dflt_py["Yes"] / prior_dflt_py["No"]) - mid_dflt_py * b1_dflt_py

grid_dflt_py = np.arange(0, 2701, 10)
# Prior times density, on the log scale, for each class.
log_yes_py = np.log(prior_dflt_py["Yes"]) + norm.logpdf(grid_dflt_py, mean_dflt_py["Yes"], sd_dflt_py)
log_no_py = np.log(prior_dflt_py["No"]) + norm.logpdf(grid_dflt_py, mean_dflt_py["No"], sd_dflt_py)
# Bayes' rule: class 1's share of the two products.
bayes_dflt_py = 1 / (1 + np.exp(log_no_py - log_yes_py))
sigmoid_dflt_py = expit(b0_dflt_py + b1_dflt_py * grid_dflt_py)

dflt_py = dflt_py.assign(yes=(dflt_py["default"] == "Yes").astype(int))
glm_dflt_py = smf.glm("yes ~ balance", data=dflt_py, family=sm.families.Binomial()).fit()
print("LDA slope:     ", float(np.round(b1_dflt_py, 6)))
```

```
LDA slope:      0.004597
```

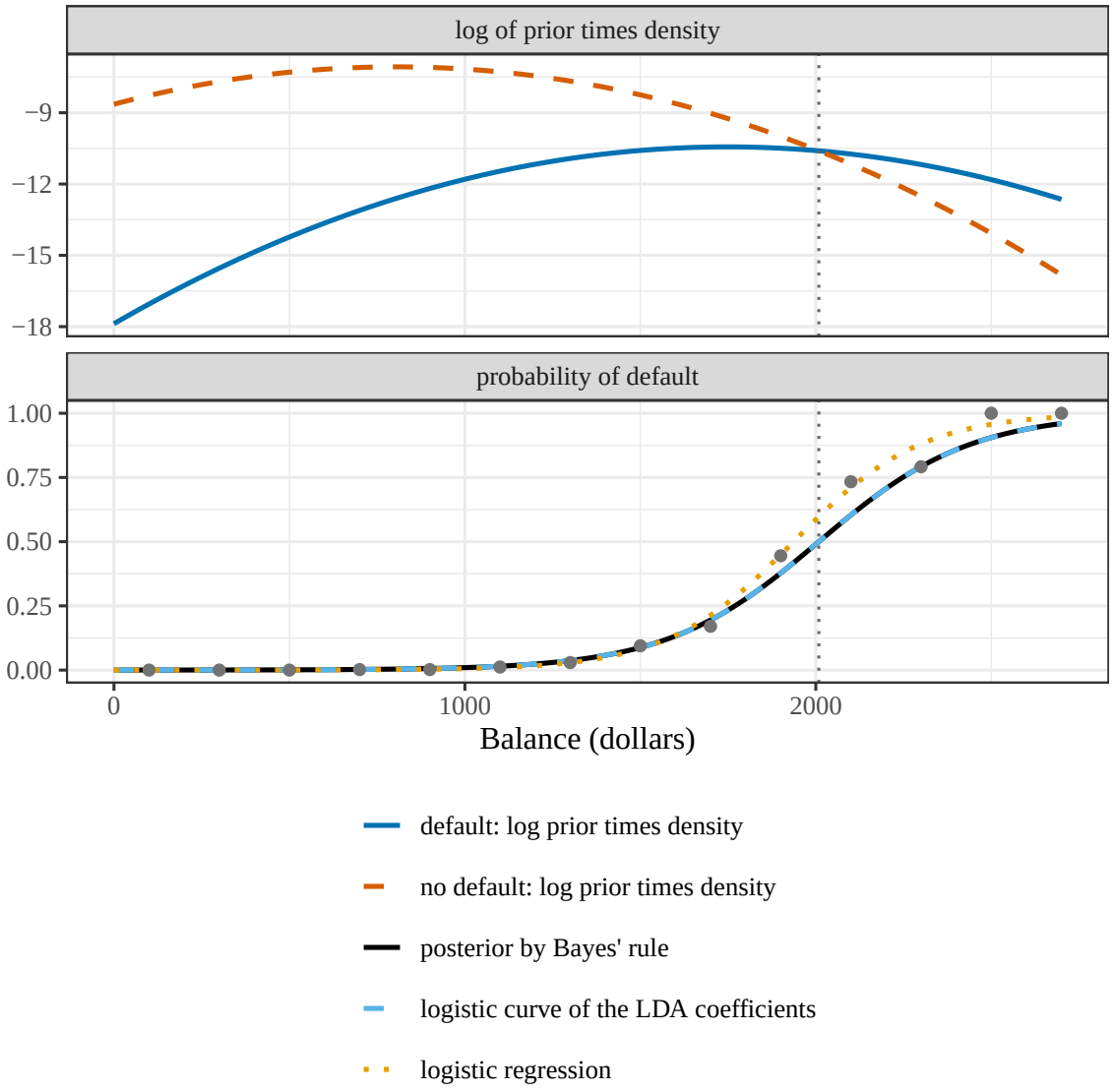
```
print("logistic slope:", float(np.round(glm_dflt_py.params["balance"], 6)))
```

```
logistic slope: 0.005499
```

In Figure 6, the logarithm[f] of the prior[g] of each class[h] times its density[i] is a parabola[j] in the balance x. The two parabolas have the same x^2 term, $-x^2/(2\Sigma)$, because the two classes share one variance[k] Σ, the single entry of the shared covariance matrix[l] of Definition 2.2, which is 1×1 here. Their difference, the log odds[m] of default, is therefore an affine function of the balance, so the posterior[n] from Bayes' rule is the logistic curve of Theorem 2.2. On the grid the two curves differ by at most 9.4e-16, the size of floating-point rounding. The LDA coefficient of balance is 0.0046 per dollar, and the posterior crosses one half at a balance of 2009 dollars. The small prior of default, 0.033, moves that crossing well above 1276 dollars, the midpoint of the two class means[o], where the two densities alone would cross. A logistic regression fitted to the same card holders has the coefficient 0.0055, so its curve is steeper than LDA's. The two fits differ because the balances do not follow LDA's model: the sample standard deviations[p] of balance are 341 dollars for the card holders who defaulted and 456 dollars for the others, not one shared value.

2.0.13 R

```r
dflt_series <- c(
  "default: log prior times density",
  "no default: log prior times density",
  "posterior by Bayes' rule",
  "logistic curve of the LDA coefficients",
  "logistic regression"
)
dflt_panels <- c("log of prior times density", "probability of default")
dflt_logit_eta <- dflt_logit[[1]] + dflt_logit[[2]] * dflt_grid
dflt_logit_curve <- plogis(dflt_logit_eta)
dflt_n_grid <- length(dflt_grid)
dflt_values <- c(
  dflt_log_yes, dflt_log_no, dflt_bayes, dflt_sigmoid, dflt_logit_curve
)
dflt_panel_of <- dflt_panels[c(1, 1, 2, 2, 2)]
dflt_balance_col <- rep(dflt_grid, 5)
dflt_series_col <- rep(dflt_series, each = dflt_n_grid)
dflt_panel_col <- rep(dflt_panel_of, each = dflt_n_grid)
dflt_curves <- tibble::tibble(
  balance = dflt_balance_col,
  value = dflt_values,
  series = dflt_series_col,
  panel = dflt_panel_col
)
dflt_curves$series <- factor(dflt_curves$series, levels = dflt_series)
dflt_bins <- dflt |>
  dplyr::mutate(band = floor(balance / 200) * 200 + 100) |>
  dplyr::group_by(band) |>
  dplyr::summarise(value = mean(default == "Yes"), .groups = "drop") |>
  dplyr::mutate(panel = dflt_panels[[2]])
dflt_colours <- c("#0072B2", "#D55E00", "black", "#56B4E9", "#E69F00")
names(dflt_colours) <- dflt_series
dflt_types <- c("solid", "dashed", "solid", "dashed", "dotted")
names(dflt_types) <- dflt_series
dflt_bin_aes <- ggplot2::aes(x = band, y = value)
dflt_facets <- ggplot2::vars(panel)
dflt_legend <- ggplot2::guide_legend(ncol = 1)
dflt_curves |>
  ggplot2::ggplot() +
  ggplot2::aes(
    x = balance, y = value, colour = series, linetype = series
  ) +
  ggplot2::geom_line(linewidth = 0.9) +
  ggplot2::geom_point(
    data = dflt_bins,
    mapping = dflt_bin_aes,
    inherit.aes = FALSE,
    colour = "grey45"
  ) +
  ggplot2::geom_vline(
    xintercept = dflt_boundary, linetype = "dotted", colour = "grey40"
  ) +
  ggplot2::facet_wrap(dflt_facets, ncol = 1, scales = "free_y") +
  ggplot2::scale_colour_manual(values = dflt_colours) +
  ggplot2::scale_linetype_manual(values = dflt_types) +
  ggplot2::guides(colour = dflt_legend, linetype = dflt_legend) +
  ggplot2::labs(
    x = "Balance (dollars)", y = NULL, colour = NULL, linetype = NULL
  ) +
  ggplot2::theme(legend.position = "bottom")
```


log of prior times density
−9
−12
−15
−18
probability of default
1.00
0.75
0.50
0.25
0.00
0
1000
2000
Balance (dollars)
default: log prior times density
no default: log prior times density
posterior by Bayes' rule
logistic curve of the LDA coefficients
logistic regression

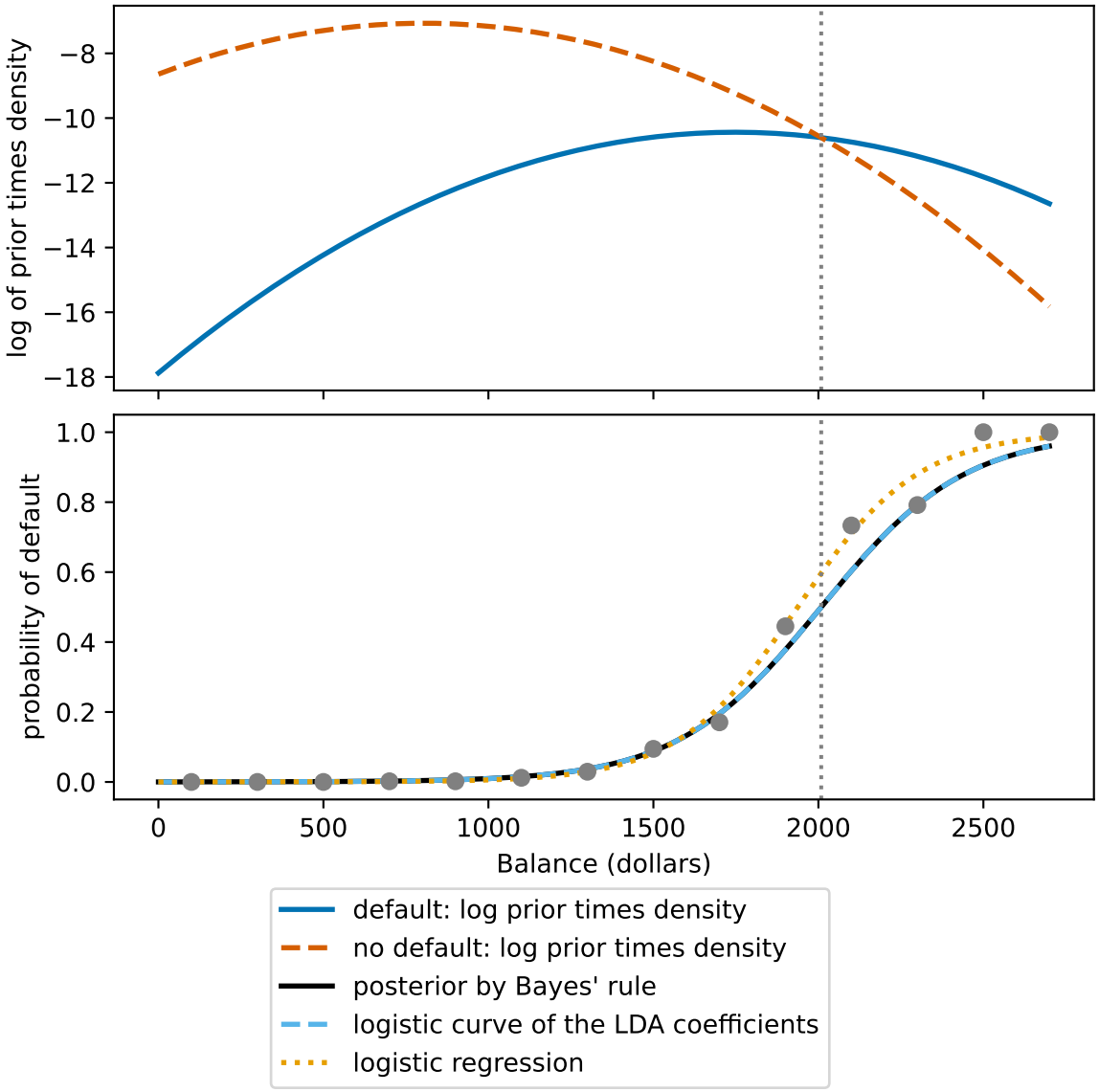
(a)

2.0.14 Python

```
import matplotlib.pyplot as plt

boundary_dflt_py = -b0_dflt_py / b1_dflt_py
band_py = np.floor(dflt_py["balance"] / 200) * 200 + 100
rate_py = dflt_py.groupby(band_py)["yes"].mean()
logit_curve_py = expit(glm_dflt_py.params["Intercept"] + glm_dflt_py.params["balance"] * grid_dflt_py)

fig, (top, bottom) = plt.subplots(2, 1, figsize=(6, 6), sharex=True, layout="constrained")
top.plot(grid_dflt_py, log_yes_py, color="#0072B2", linestyle="-", linewidth=2,
         label="default: log prior times density");
top.plot(grid_dflt_py, log_no_py, color="#D55E00", linestyle="--", linewidth=2,
         label="no default: log prior times density");
top.set_ylabel("log of prior times density");
bottom.plot(grid_dflt_py, bayes_dflt_py, color="black", linestyle="-", linewidth=2,
            label="posterior by Bayes' rule");
bottom.plot(grid_dflt_py, sigmoid_dflt_py, color="#56B4E9", linestyle="--", linewidth=2,
            label="logistic curve of the LDA coefficients");
bottom.plot(grid_dflt_py, logit_curve_py, color="#E69F00", linestyle=":", linewidth=2,
            label="logistic regression");
bottom.scatter(rate_py.index, rate_py.values, color="grey", zorder=3);
bottom.set_ylabel("probability of default");
bottom.set_xlabel("Balance (dollars)");
for ax in (top, bottom):
    ax.axvline(boundary_dflt_py, color="grey", linestyle=":");
fig.legend(loc="outside lower center", ncol=1);
plt.show()
```



(b)

Figure 6: LDA fitted to the `Default` data in Example 2.5. Top: the logarithm[q] of the prior[r] of each class[s] times its normal density[t], with the pooled variance[u]. Bottom: the posterior[v] probability[w] of default from Bayes' rule, the logistic curve of Theorem 2.2, a fitted logistic regression, and the share of defaults in each 200-dollar band of balance (gray dots). The dotted vertical line marks where the posterior crosses one half.

[a] https://morrison-lab.github.io/sds/classification.html#def-class
[b] https://morrison-lab.github.io/sds/bayesian-inference.html#def-prior
[c] https://morrison-lab.github.io/pds/random-variables.html#def-normal
[d] https://morrison-lab.github.io/sds/bayesian-inference.html#def-posterior
[e] https://morrison-lab.github.io/pds/probability-basics.html#def-probability
[f] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-logarithm
[g] https://morrison-lab.github.io/sds/bayesian-inference.html#def-prior
[h] https://morrison-lab.github.io/sds/classification.html#def-class
[i] https://morrison-lab.github.io/pds/random-variables.html#def-pdf
[j] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-quadratic-cubic
[k] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance
[l] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x
[m] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds

[n] https://morrison-lab.github.io/sds/bayesian-inference.html#def-posterior
[o] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean
[p] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-sd
[q] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-logarithm
[r] https://morrison-lab.github.io/sds/bayesian-inference.html#def-prior
[s] https://morrison-lab.github.io/sds/classification.html#def-class
[t] https://morrison-lab.github.io/pds/random-variables.html#def-normal
[u] https://morrison-lab.github.io/sds/basic-statistical-methods.html#def-pooled-variance
[v] https://morrison-lab.github.io/sds/bayesian-inference.html#def-posterior
[w] https://morrison-lab.github.io/pds/probability-basics.html#def-probability

Exercise 2.7 (Same form, different fitting criterion)**.** By Theorem 2.2, two-class LDA and logistic regression give a posterior probability of the same functional form.
Each method chooses its parameters to maximize a likelihood[a]. State the quantity LDA maximizes and the quantity logistic regression maximizes, in terms of π_k, f_k and $\Pr[Y = 1 \mid \underset{\sim}{X} = \underset{\sim}{x}]$.

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

Solution

Solution 2.7. LDA maximizes the *joint* likelihood of the labels and the features,

$$\prod_{i=1}^{n} \pi_{y_i} f_{y_i}(\underset{\sim}{x}_i),$$

and logistic regression maximizes the *conditional* likelihood of the labels given the features,

$$\prod_{i=1}^{n} \Pr[Y_i = y_i \mid \underset{\sim}{X}_i = \underset{\sim}{x}_i].$$

The LDA maximizers have closed forms: the class[a] proportions[b], class means[c], and pooled covariance matrix[d] of Definition 2.2, with the divisor n in place of $n - K$ (Remark 2.2). The logistic estimates[e] need the iterative methods[f] of the logistic regression page, such as the Newton iteration of the Newton's method exercise. Let $\mathbb{p}(\underset{\sim}{x}) = \sum_l \pi_l f_l(\underset{\sim}{x})$ be the marginal density[g] of the features. By Theorem 1.1, $\pi_{y_i} f_{y_i}(\underset{\sim}{x}_i) = \Pr[Y_i = y_i \mid \underset{\sim}{X}_i = \underset{\sim}{x}_i]\, \mathbb{p}(\underset{\sim}{x}_i)$, so the joint likelihood is the conditional likelihood times $\prod_i \mathbb{p}(\underset{\sim}{x}_i)$. LDA therefore also fits the marginal distribution[h] of the features, which logistic regression does not model.

[a] https://morrison-lab.github.io/sds/classification.html#def-class
[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/variance-covariance.html#def-cov-vec-x
[e] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[f] https://morrison-lab.github.io/mds/optimization-newton.html#def-iterative-method
[g] https://morrison-lab.github.io/pds/random-variables.html#def-marginal
[h] https://morrison-lab.github.io/pds/random-variables.html#def-marginal

Exercise 2.8 (Same form, different fitting criterion (part 2))**.** Use the setup of Exercise 2.7.
Does a logistic model with log odds[a] affine in $\underset{\sim}{x}$ imply that the classes[b] are Gaussian with a common covariance matrix[c]? Give a counterexample or a proof.

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

Solution

Solution 2.8. No: logistic log odds[a] that are an affine function of $\underset{\sim}{x}$ do not imply Gaussian[b] class[c]es. Consider a single binary feature $X \in \{0,1\}$ with $\Pr[X = 1 \mid Y = k] = q_k$, where $0 < q_k < 1$. The class densities[d] are $f_k(x) = q_k^x(1-q_k)^{1-x}$ for $x \in \{0,1\}$, with logarithms[e] $\log f_k(x) = x \log q_k + (1-x)\log(1-q_k)$. By Theorem 1.1, the log odds of $Y = 1$ are

$$\begin{aligned}
\log \frac{\pi_1 f_1(x)}{\pi_0 f_0(x)} &= \log \frac{\pi_1}{\pi_0} + \log f_1(x) - \log f_0(x) && \text{(log of a quotient of products)} \\
&= \log \frac{\pi_1}{\pi_0} + x \log q_1 + (1-x)\log(1-q_1) - x \log q_0 - (1-x)\log(1-q_0) && \text{(substitute } \log f_k(x)\text{)} \\
&= \log \frac{\pi_1}{\pi_0} + x \log \frac{q_1}{q_0} + (1-x)\log \frac{1-q_1}{1-q_0} && \text{(log of a quotient)} \\
&= \log \frac{\pi_1}{\pi_0} + \log \frac{1-q_1}{1-q_0} + x\left(\log \frac{q_1}{q_0} - \log \frac{1-q_1}{1-q_0}\right) && \text{(collect the terms in } x\text{)},
\end{aligned}$$

which is an affine function of x, although X is not Gaussian. Many other families of class-conditional densities also give log odds that are affine in $\underset{\sim}{x}$, so logistic regression makes weaker distributional assumptions than LDA [8, Sec. 4.4.5].

[a] https://morrison-lab.github.io/pds/probability-basics.html#def-log-odds
[b] https://morrison-lab.github.io/pds/random-variables.html#def-normal
[c] https://morrison-lab.github.io/sds/classification.html#def-class
[d] https://morrison-lab.github.io/pds/random-variables.html#def-pdf
[e] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-logarithm

Exercise 2.9 (Same form, different fitting criterion (part 3))**.** Use the setup of Exercise 2.7. Which method is more affected when the features are not Gaussian, and which uses the data more efficiently when they are?

Solution

Solution 2.9. LDA is more affected when the features are not Gaussian. Its estimates[a] are then fitted to a misspecified density, and its boundary can have a higher error rate than logistic regression's. LDA uses the data more efficiently when the features are Gaussian: it is then asymptotically more efficient, because it also fits the marginal distribution[b] of the features, which carries information about the parameters. In practice the two methods often give similar boundaries [8, Sec. 4.4.5].

[a] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[b] https://morrison-lab.github.io/pds/random-variables.html#def-marginal

Exercise 2.10 (Same form, different fitting criterion (part 4))**.** Use the setup of Exercise 2.7. the separation exercise showed that the logistic regression coefficients have no maximum likelihood estimate[a] when the two classes[b] are perfectly separated. What happens to the LDA parameters in that case?

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

Solution

Solution 2.10. The LDA parameter estimates[a] remain finite. The class[b] proportions[c], class means[d], and pooled covariance matrix[e] of Definition 2.2 are averages[f] of the data, so they are finite for any data. When the pooled covariance matrix is invertible[g], the coefficients $\vec{\beta}$ and β_0 of Theorem 2.2 are finite too. So LDA gives a finite decision boundary when the classes[h] are linearly separable, and the

logistic regression coefficients diverge (the separation exercise).

[a] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[b] https://morrison-lab.github.io/sds/classification.html#def-class
[c] https://morrison-lab.github.io/sds/classification.html#def-prevalence
[d] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean
[e] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x
[f] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean
[g] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-matrix-inverse
[h] https://morrison-lab.github.io/sds/classification.html#def-class

3 Quadratic discriminant analysis

Exercise 3.1 (Gaussian classes with their own covariance matrices)**.** Repeat Exercise 2.1, but let each class have its own covariance matrix: $\underset{\sim}{X} \mid Y = k \sim \mathrm{N}(\vec{\mu}_k, \Sigma_k)$.
Which terms of $\log(\pi_k f_k(\underset{\sim}{x}))$ can still be dropped because they are the same for all classes?

Solution

Solution 3.1. Only $-\frac{p}{2}\log(2\pi)$.
Assume that every Σ_k is positive definite[a] and every $\pi_k > 0$. Then each $(\Sigma_k)^{-1}$ exists, because a positive definite matrix is invertible[b], each $\log|\Sigma_k|$ and $\log \pi_k$ is defined, and every product $\pi_k f_k(\underset{\sim}{x})$ is positive. The derivation of Solution 2.1 then goes through with Σ_k in place of Σ. The term $-\frac{1}{2}\log|\Sigma_k|$ now depends on k, and so does the quadratic form[c] $\vec{\mathrm{d}}_k^{\top}(\Sigma_k)^{-1}\vec{\mathrm{d}}_k$, where $\vec{\mathrm{d}}_k \overset{\text{def}}{=} (\underset{\sim}{x} - \vec{\mu}_k)$, because Σ_k is inside it. In particular, $-\frac{1}{2}\underset{\sim}{x}^{\top}(\Sigma_k)^{-1}\underset{\sim}{x}$ differs between classes[d], so it cannot be dropped.

[a] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-positive-definite
[b] https://morrison-lab.github.io/mds/linear-algebra-diagonalization.html#thm-pd-invertible
[c] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-quadratic-form
[d] https://morrison-lab.github.io/sds/classification.html#def-class

Exercise 3.2 (Gaussian classes with their own covariance matrices (part 2))**.** Use the setup of Exercise 3.1.
Give the remaining discriminant function $\delta_k(\underset{\sim}{x})$.

Solution

Solution 3.2. With the constant[a] dropped,

$$\delta_k(\underset{\sim}{x}) = -\frac{1}{2}\log|\Sigma_k| - \frac{1}{2}\vec{\mathrm{d}}_k^{\top}(\Sigma_k)^{-1}\vec{\mathrm{d}}_k + \log \pi_k,$$

where $\vec{\mathrm{d}}_k \overset{\text{def}}{=} (\underset{\sim}{x} - \vec{\mu}_k)$ is the deviation[b] of $\underset{\sim}{x}$ from the class[c] mean[d].

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function
[b] https://morrison-lab.github.io/pds/variance-covariance.html#def-deviation
[c] https://morrison-lab.github.io/sds/classification.html#def-class
[d] https://morrison-lab.github.io/pds/expectation.html#def-expectation

Exercise 3.3 (Gaussian classes with their own covariance matrices (part 3))**.** Use the setup of Exercise 3.1.
Is $\delta_k(\underset{\sim}{x})$ an affine function of $\underset{\sim}{x}$?

Solution

Solution 3.3. No: $\delta_k(\underset{\sim}{x})$ is not an affine function of $\underset{\sim}{x}$. Expanding the quadratic form as in Solution 2.1 shows that $\delta_k(\underset{\sim}{x})$ contains $-\frac{1}{2}\underset{\sim}{x}^\top(\Sigma_k)^{-1}\underset{\sim}{x}$, a quadratic form in $\underset{\sim}{x}$ that is not zero for any $\underset{\sim}{x} \neq \vec{0}$, because $(\Sigma_k)^{-1}$ is positive definite[a].

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

Definition 3.1 (Quadric surface, quadric)**.** A **quadric surface**, or **quadric**, in $\mathbb{R}^p$ is a set[a] of the form

$$\{\underset{\sim}{x} \in \mathbb{R}^p : \underset{\sim}{x}^\top \mathbf{A}\underset{\sim}{x} + \underset{\sim}{a} \cdot \underset{\sim}{x} + c = 0\},$$

where $\mathbf{A}$ is a nonzero $p \times p$ symmetric matrix[b], $\underset{\sim}{a} \in \mathbb{R}^p$, and $c \in \mathbb{R}$ [9].

[a] https://morrison-lab.github.io/mcs/sets.html#def-set
[b] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-symmetric-matrix

Example 3.1 (Quadric surfaces in two dimensions and in one)**.**

- With $p = 2$, $\mathbf{A} = \mathrm{diag}(1, 1)$, $\underset{\sim}{a} = \vec{0}$, and $c = -1$, the equation is $x_1^2 + x_2^2 - 1 = 0$, so the quadric surface is the unit circle[a].
- With $p = 1$, $\mathbf{A}$ is a nonzero number a_2, $\underset{\sim}{a}$ is a number a_1, and the equation is $a_2x^2 + a_1x + c = 0$, so the quadric surface is the set[b] of real[c] roots of a quadratic[d] equation: at most two numbers.

[a] https://morrison-lab.github.io/mds/algebra-trigonometry.html#def-unit-circle
[b] https://morrison-lab.github.io/mcs/sets.html#def-set
[c] https://morrison-lab.github.io/mcs/number-systems.html#def-real-numbers
[d] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-quadratic-cubic

Example 3.2 (Three points on a line are not a quadric surface)**.** With $p = 1$, the set[a] $\{-1, 0, 1\}$ is not a quadric surface. With $p = 1$, Definition 3.1 needs a nonzero number a_2 with $a_2x^2 + a_1x + c = 0$ at each of $x = -1$, $x = 0$, and $x = 1$, and a quadratic[b] equation has at most two real[c] roots.

[a] https://morrison-lab.github.io/mcs/sets.html#def-set
[b] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-quadratic-cubic
[c] https://morrison-lab.github.io/mcs/number-systems.html#def-real-numbers

Exercise 3.4 (Gaussian classes with their own covariance matrices (part 4))**.** Use the setup of Exercise 3.1.
What geometric surface is the set[a] $\{\underset{\sim}{x} : \delta_k(\underset{\sim}{x}) = \delta_l(\underset{\sim}{x})\}$ where the discriminant functions[b] of two classes[c] agree?

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

Solution

Solution 3.4. The difference $\delta_k(\underset{\sim}{x}) - \delta_l(\underset{\sim}{x})$ is a polynomial[a] of degree[b] at most two in the coordinates of $\underset{\sim}{x}$, with quadratic[c] part $-\frac{1}{2}\underset{\sim}{x}^\top\left((\Sigma_k)^{-1} - (\Sigma_l)^{-1}\right)\underset{\sim}{x}$. The matrix[d] $-\frac{1}{2}\left((\Sigma_k)^{-1} - (\Sigma_l)^{-1}\right)$ is symmetric[e], and it is nonzero when $\Sigma_k \neq \Sigma_l$, because $(\Sigma_k)^{-1} = (\Sigma_l)^{-1}$ would force $\Sigma_k = \Sigma_l$. So when $\Sigma_k \neq \Sigma_l$, the boundary $\{\underset{\sim}{x} : \delta_k(\underset{\sim}{x}) = \delta_l(\underset{\sim}{x})\}$ is a quadric surface. When $p = 2$, it is a conic section: an ellipse, a parabola, a hyperbola, or a degenerate case such as a pair of lines. When $\Sigma_k = \Sigma_l$, the quadratic terms and the log-determinant terms cancel, and the boundary is a hyperplane, as for LDA.

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

[b] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-polynomial-degree
[c] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-quadratic-cubic
[d] https://morrison-lab.github.io/mds/linear-algebra-matrices.html#def-matrix
[e] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-symmetric-matrix

Exercise 3.5 (Gaussian classes with their own covariance matrices (part 5))**.** Use the setup of Exercise 3.1.
How many points can the boundary between two classes[a] contain when $p = 1$ and $\Sigma_k \neq \Sigma_l$?

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

Solution

Solution 3.5. When $p = 1$ and the two variances[a] differ, the condition $\delta_k(x) = \delta_l(x)$ is a quadratic equation in x with a nonzero coefficient of x^2, so the boundary consists of at most two points.

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

Definition 3.2 (Quadratic discriminant function)**.** For class[a] k with prior π_k, mean[b] $\vec{\mu}_k$ and covariance matrix[c] Σ_k, the **quadratic discriminant function** is

$$\delta_k(\underset{\sim}{x}) = -\frac{1}{2}\log|\Sigma_k| - \frac{1}{2}\vec{\mathrm{d}}_k^{\top}(\Sigma_k)^{-1}\vec{\mathrm{d}}_k + \log\pi_k,$$

where $\vec{\mathrm{d}}_k \overset{\text{def}}{=} (\underset{\sim}{x} - \vec{\mu}_k)$ is the deviation[d] of $\underset{\sim}{x}$ from the class mean.

[a] https://morrison-lab.github.io/sds/classification.html#def-class
[b] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[c] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x
[d] https://morrison-lab.github.io/pds/variance-covariance.html#def-deviation

Example 3.3 (The quadratic discriminant function with one feature)**.** Let $p = 1$, with two classes[a] that have the same mean[b] $\vec{\mu}_0 = \vec{\mu}_1 = 0$, variances[c] $\Sigma_0 = 1$ and $\Sigma_1 = 4$, and equal priors. Evaluate the quadratic discriminant function of each class at $\underset{\sim}{x} = 0.5$.

```
qdf_var <- c(1, 4)
qdf_x <- 0.5
qdf_delta <- -log(qdf_var) / 2 - qdf_x^2 / (2 * qdf_var) + log(0.5)
qdf_delta
```

```
[1] -0.8181472 -1.4175444
```

With one feature and $\vec{\mu}_k = 0$, $\delta_k(\underset{\sim}{x}) = -\frac{1}{2}\log\Sigma_k - \underset{\sim}{x}^2/(2\Sigma_k) + \log\pi_k$, so $\delta_0(0.5) =$ -0.818 and $\delta_1(0.5) =$ -1.418.

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

Theorem 3.1 (The quadratic discriminant function)**.** *Suppose that within class[a] k the features are multivariate Gaussian[b], $\underset{\sim}{X} \mid Y = k \sim \mathrm{N}(\vec{\mu}_k, \Sigma_k)$, with its own positive definite[c] covariance matrix[d] Σ_k, and prior $\pi_k > 0$ for every class. Then the marginal density[e] of $\underset{\sim}{X}$ is positive everywhere:*

$$\mathbb{p}(\underset{\sim}{x}) > 0 \quad \textit{for every } \underset{\sim}{x} \in \mathbb{R}^p. \tag{3}$$

The class that maximizes $\pi_k f_k(\underset{\sim}{x})$ also maximizes the quadratic discriminant function $\delta_k(\underset{\sim}{x})$:

$$\arg\max_k \pi_k f_k(\underset{\sim}{x}) = \arg\max_k \delta_k(\underset{\sim}{x}). \tag{4}$$

[a] https://morrison-lab.github.io/sds/classification.html#def-class
[b] https://morrison-lab.github.io/pds/distributions.html#def-mvn
[c] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-positive-definite
[d] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x
[e] https://morrison-lab.github.io/pds/random-variables.html#def-marginal

Proof

Proof. By Solution 3.1 and Solution 3.2. □

Example 3.4 (The quadratic discriminant with one feature)**.** Let $p = 1$, with two classes[a] that have the same mean[b] $\vec{\mu}_0 = 0$ and $\vec{\mu}_1 = 0$, variances[c] $\Sigma_0 = 1$ and $\Sigma_1 = 4$, and equal priors. Classify $\underset{\sim}{x} = 0.5$ and $\underset{\sim}{x} = 3$.

```
qd_var <- c(1, 4)
qd_delta <- function(x) -log(qd_var) / 2 - x^2 / (2 * qd_var) + log(0.5)
qd_x <- c(0.5, 3)
qd_sd <- sqrt(qd_var)
qd_one <- numeric(1)
qd_class_by_delta <- vapply(qd_x, \(x) {
  qd_d <- qd_delta(x)
  which.max(qd_d) - 1
}, qd_one)
qd_class_by_density <- vapply(qd_x, \(x) {
  qd_f <- dnorm(x, 0, qd_sd)
  which.max(0.5 * qd_f) - 1
}, qd_one)
# delta_0(x) = delta_1(x) where x^2 (1/2 - 1/8) = log(4) / 2
qd_root <- sqrt(log(4) / 2 / (1 / 2 - 1 / 8))
rbind(
  x = qd_x,
  class_by_delta = qd_class_by_delta,
  class_by_density = qd_class_by_density
)
```

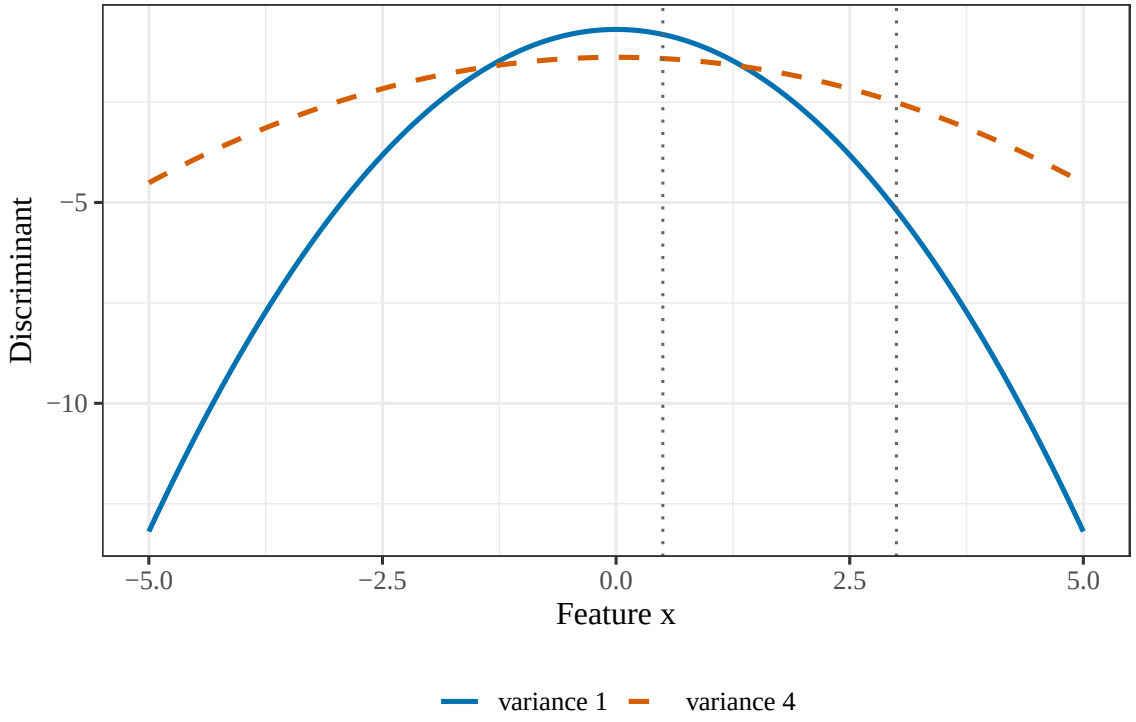
```
                [,1] [,2]
x                0.5    3
class_by_delta   0.0    1
class_by_density 0.0    1
```

By Theorem 3.1 the discriminants and the densities give the same classes: class[d] 0 for $\underset{\sim}{x} = 0.5$ and class 1 for $\underset{\sim}{x} = 3$. The narrower class has the larger discriminant near the shared mean, and the wider class has the larger discriminant far from it. So $\delta_0 = \delta_1$ at two points, $x = \pm\ 1.36$, the roots of a quadratic equation in x. The rule assigns class 0 to the interval between the roots and class 1 to both tails. No linear discriminant can give this rule: with one feature, an affine discriminant difference changes sign at most once, so it splits the line into two pieces, not three.

[a] https://morrison-lab.github.io/sds/classification.html#def-class
[b] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[c] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance
[d] https://morrison-lab.github.io/sds/classification.html#def-class

3.0.1 R

```r
library(ggplot2)
qdf_grid <- seq(-5, 5, length.out = 241)
qdf_curves <- data.frame(
  x = rep(qdf_grid, 2),
  delta = c(sapply(qdf_grid, \(x) qd_delta(x)[1]), sapply(qdf_grid, \(x) qd_delta(x)[2])),
  class = rep(c("variance 1", "variance 4"), each = length(qdf_grid))
)
ggplot(qdf_curves) +
  aes(x, delta, colour = class, linetype = class) +
  geom_line(linewidth = 0.9) +
  geom_vline(xintercept = qd_x, linetype = "dotted", colour = "grey40") +
  scale_colour_manual(values = c("variance 1" = "#0072B2", "variance 4" = "#D55E00")) +
  scale_linetype_manual(values = c("variance 1" = "solid", "variance 4" = "dashed")) +
  labs(x = "Feature x", y = "Discriminant", colour = NULL, linetype = NULL)
```



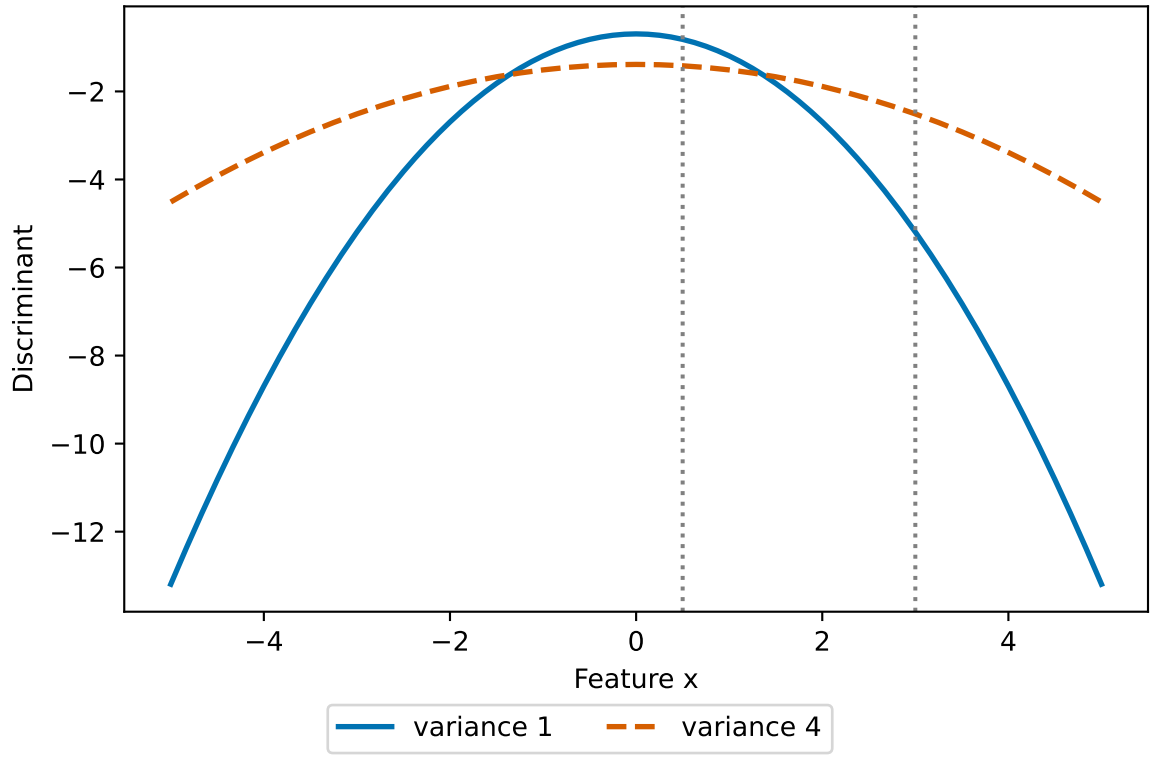
(a)

3.0.2 Python

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

var_qdf = np.asarray(r.qd_var)
grid_qdf = np.linspace(-5, 5, 241)
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
for k, colour, style in [(0, "#0072B2", "-"), (1, "#D55E00", "--")]:
    delta_qdf = -np.log(var_qdf[k]) / 2 - grid_qdf ** 2 / (2 * var_qdf[k]) + np.log(0.5)
    ax.plot(grid_qdf, delta_qdf, color=colour, linestyle=style, linewidth=2,
            label=f"variance {var_qdf[k]:.0f}");
for x_qdf in np.atleast_1d(r.qd_x):
    ax.axvline(x_qdf, color="grey", linestyle=":");
```

```
ax.set_xlabel("Feature x");
ax.set_ylabel("Discriminant");
fig.legend(loc="outside lower center", ncol=2);
plt.show()
```



(b)

Figure 7: The two quadratic discriminant functions of Example 3.4, whose classes[13] have the same mean[14]. The point $x = 0.5$ (dotted vertical) is between the two crossings, where the class[15] of variance[16] 1 has the higher discriminant, and the point $x = 3$ is outside them, where the class of variance 4 does, so the two points go to different classes.

Corollary 3.1 (Quadratic discriminant boundaries are quadric surfaces)**.** *Under the assumptions of Theorem 3.1, let k and l be two classes[a] with $\Sigma_k \neq \Sigma_l$. Then the set[b] on which their quadratic discriminant functions agree,*

$$\{\underset{\sim}{x} \in \mathbb{R}^p : \delta_k(\underset{\sim}{x}) = \delta_l(\underset{\sim}{x})\},$$

is a quadric surface.

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

Proof

Proof. By Solution 3.4. □

[13] https://morrison-lab.github.io/sds/classification.html#def-class
[14] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[15] https://morrison-lab.github.io/sds/classification.html#def-class
[16] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance

Example 3.5 (A circular quadratic discriminant boundary)**.** Let $p = 2$, with two classes[a] that have the same mean[b] $\vec{\mu}_0 = \vec{\mu}_1 = \vec{0}$ and the same prior $\pi_0 = \pi_1 = \frac{1}{2}$. Their covariance matrices[c] are $\Sigma_0 = \text{diag}(1, 1)$ and $\Sigma_1 = \text{diag}(4, 4)$. Then $|\Sigma_0| = 1$ and $|\Sigma_1| = 16$, so the quadratic discriminant functions are

$$\delta_0(\underset{\sim}{x}) = -\frac{1}{2}\|\underset{\sim}{x}\|^2 + \log\frac{1}{2}$$

and

$$\delta_1(\underset{\sim}{x}) = -\frac{1}{2}\log 16 - \frac{1}{8}\|\underset{\sim}{x}\|^2 + \log\frac{1}{2}.$$

```
qb_radius_sq <- log(16) / 2 / (1 / 2 - 1 / 8)
qb_radius_sq
```

```
[1] 3.696785
```

So $\delta_0(\underset{\sim}{x}) = \delta_1(\underset{\sim}{x})$ where $\frac{3}{8}\|\underset{\sim}{x}\|^2 - \frac{1}{2}\log 16 = 0$, which is the circle[d] $\|\underset{\sim}{x}\|^2 = 3.7$ around the origin. By Corollary 3.1 this circle is a quadric surface, with $\mathbf{A} = \frac{3}{8}\,\text{diag}(1, 1)$, $\underset{\sim}{a} = \vec{0}$, and $c = -\frac{1}{2}\log 16$.

[a] https://morrison-lab.github.io/sds/classification.html#def-class
[b] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[c] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x
[d] https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-circle

Definition 3.3 (Quadratic discriminant analysis (QDA))**. Quadratic discriminant analysis** is the generative classifier with a multivariate normal[a] density in each class[b] and a separate covariance matrix[c] Σ_k for each class [10]. It predicts the class with the largest quadratic discriminant function of Theorem 3.1, with each parameter replaced by its estimate[d]. The priors and means are estimated as in Definition 2.2. Each $\widehat{\Sigma}_k$ is the sample[e] covariance matrix of the training points in class k alone, with divisor $n_k - 1$. R's `MASS::qda()` uses that divisor. scikit-learn's `QuadraticDiscriminantAnalysis` uses it in version 1.8.0, and it divides by n_k in version 1.9.1.
The LDA model is the QDA model with the constraint that $\Sigma_k = \Sigma$ for every class k.

[a] https://morrison-lab.github.io/pds/distributions.html#def-mvn
[b] https://morrison-lab.github.io/sds/classification.html#def-class
[c] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x
[d] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[e] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample

Example 3.6 (The quadratic boundary for one predictor)**.** Give the two directions of the `Smarket` days before 2005 their own variances[a], as in Example 1.3, and find where the Up and Down discriminant functions of QDA are equal. Setting $\delta_{\text{Up}}(x) = \delta_{\text{Down}}(x)$ and collecting terms gives a quadratic equation in x.

3.0.3 R

```
delta_qda <- function(x, k, mean, sd, prior) {
  -log(sd[[k]]) - (x - mean[[k]])^2 / (2 * sd[[k]]^2) + log(prior[[k]])
}
delta_up <- function(x) delta_qda(x, "Up", class_mean, class_sd, prior)
delta_down <- function(x) delta_qda(x, "Down", class_mean, class_sd, prior)

# delta_Up - delta_Down is quadratic in x; fit its three coefficients exactly
grid <- c(-1, 0, 1)
gap <- delta_up(grid) - delta_down(grid)
roots <- sort(Re(polyroot(coef(lm(gap ~ grid + I(grid^2))))))
cat("roots:", round(roots, 3), "\n")
```

```
roots: 0.539 23.441
```

```r
for (x in c(-1, 1, 30)) {
  pred_class <- if (delta_up(x) > delta_down(x)) "Up" else "Down"
  cat(x, "->", pred_class, "\n")
}
```

```
-1 -> Up
1 -> Down
30 -> Up
```

3.0.4 Python

```python
def delta_qda_py(x, k):
    return (-np.log(sd_py[k]) - (x - mean_py[k]) ** 2 / (2 * sd_py[k] ** 2)
            + np.log(prior_py[k]))

# delta_Up - delta_Down is quadratic in x; fit its three coefficients exactly
grid_py = np.array([-1.0, 0.0, 1.0])
coef_py = np.polyfit(grid_py, delta_qda_py(grid_py, "Up") - delta_qda_py(grid_py, "Down"), 2)
roots_py = np.sort(np.roots(coef_py))
print("roots:", roots_py.round(3))
```

```
roots: [ 0.539 23.441]
```

```python
for x in [-1.0, 1.0, 30.0]:
    up = delta_qda_py(x, "Up") > delta_qda_py(x, "Down")
    print(x, "->", "Up" if up else "Down")
```

```
-1.0 -> Up
1.0 -> Down
30.0 -> Up
```

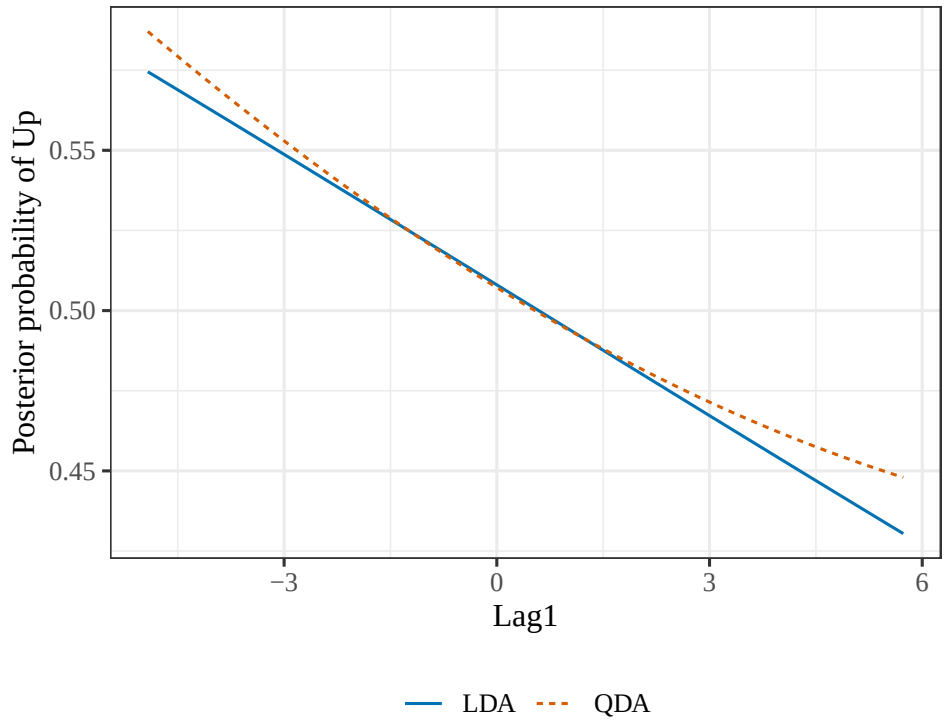
The two discriminant functions are equal at $x = 0.54$ and $x = 23.44$. The first root is in the middle of the data, where the returns range from -4.9 to 5.7. The second root is far outside that range, so within the range of the data the boundary is a single point, as for LDA. The boundary is curved only because the Up variance[b], 1.517, differs slightly from the Down variance, 1.507. With the variances[c] this close, the coefficient of x^2 is small, and QDA gives almost the same classifier as LDA.

[a] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance
[b] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance
[c] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance

3.0.5 R

```r
library(ggplot2)
lq_x <- seq(min(da_train$Lag1), max(da_train$Lag1), length.out = 300)
lq_post <- function(delta_fun) {
  up <- delta_fun(lq_x, "Up")
  down <- delta_fun(lq_x, "Down")
  exp(up) / (exp(up) + exp(down))
}
lq_lda <- function(x, k) x * class_mean[[k]] / pooled - class_mean[[k]]^2 / (2 * pooled) + log(prior[[k]
lq_qda <- function(x, k) delta_qda(x, k, class_mean, class_sd, prior)
lq <- data.frame(
  lag1 = rep(lq_x, 2),
  posterior = c(lq_post(lq_lda), lq_post(lq_qda)),
  method = rep(c("LDA", "QDA"), each = length(lq_x))
)
```

```
ggplot(lq) +
  aes(lag1, posterior, colour = method, linetype = method) +
  geom_line() +
  scale_colour_manual(values = c(LDA = "#0072B2", QDA = "#D55E00")) +
  labs(x = "Lag1", y = "Posterior probability of Up", colour = NULL, linetype = NULL)
```



(a)

3.0.6 Python

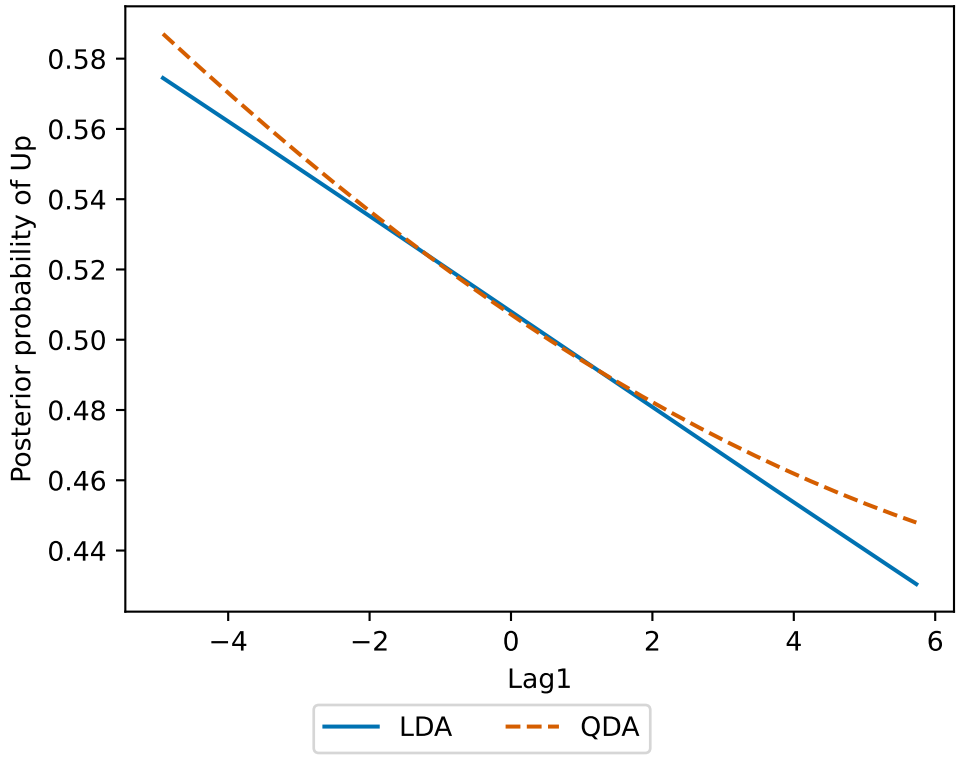
```
import matplotlib.pyplot as plt
import numpy as np

x_lq = np.linspace(da_train_py["Lag1"].min(), da_train_py["Lag1"].max(), 300)

def post_lq(delta_up, delta_down):
    return np.exp(delta_up) / (np.exp(delta_up) + np.exp(delta_down))

def delta_lda_py(x, k):
    return x * mean_py[k] / pooled_py - mean_py[k] ** 2 / (2 * pooled_py) + np.log(prior_py[k])

fig, ax = plt.subplots(figsize=(5, 4), layout="constrained")
ax.plot(x_lq, post_lq(delta_lda_py(x_lq, "Up"), delta_lda_py(x_lq, "Down")),
        color="#0072B2", linestyle="-", label="LDA");
ax.plot(x_lq, post_lq(delta_qda_py(x_lq, "Up"), delta_qda_py(x_lq, "Down")),
        color="#D55E00", linestyle="--", label="QDA");
ax.set_xlabel("Lag1");
ax.set_ylabel("Posterior probability of Up");
fig.legend(loc="outside lower center", ncol=2);
plt.show()
```



(b)

Figure 8: The posterior probability of Up given `Lag1` under LDA (solid) and QDA (dashed) for the `Smarket` days before 2005, as in Example 2.3 and Example 3.6. The two curves nearly coincide, because the two directions have nearly equal spread.

4 Naive Bayes and nearest neighbors

The chain rule of probability holds for every joint distribution[17], and it has too many parameters to use as it stands: the last factor, $\Pr(X_N \mid X_1, \dots, X_{N-1})$, is a separate distribution for every combination of the earlier variables. A modeling assumption cuts that down. Conditional independence[18] says that, once some variable is known, other variables carry no further information about each other.

Definition 4.1 (The naive Bayes assumption)**.** Let Y be a class label and $X_1, \dots, X_p$ the input[a] features. The *naive Bayes assumption* is that the features are conditionally independent given the class [11]:

$$\Pr(X_1, \dots, X_p \mid Y) = \prod_{j=1}^{p} \Pr(X_j \mid Y).$$

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

Example 4.1 (Do *free* and *money* appear independently in spam?)**.** In the Spambase data [12], let Y be the class[a] (spam or not), X_1 be "the email contains the word *free*", and X_2 be "the email contains the word *money*", as in the chain rule example. Restrict to the spam emails ($Y = 1$). Compare the joint[b] probability[c] $\Pr(X_1 = 1, X_2 = 1 \mid Y = 1)$ with the value $\Pr(X_1 = 1 \mid Y = 1)\Pr(X_2 = 1 \mid Y = 1)$ that the naive Bayes assumption would give:

4.0.1 Python

```
import pandas as pd

spambase_nb = pd.read_csv("data/spambase.csv")
spam_nb = spambase_nb[spambase_nb["Class"] == 1]   # keep only the spam emails
p1_nb = (spam_nb["word_freq_free"] > 0).mean()
p2_nb = (spam_nb["word_freq_money"] > 0).mean()
joint_nb = ((spam_nb["word_freq_free"] > 0) & (spam_nb["word_freq_money"] > 0)).mean()
print(joint_nb, p1_nb * p2_nb)
```

```
0.2410369553226696 0.20490275387446608
```

4.0.2 R

```
spambase_nb <- readr::read_csv("data/spambase.csv", show_col_types = FALSE)
spam_nb <- dplyr::filter(spambase_nb, Class == 1) # keep only the spam emails
p1_nb <- mean(spam_nb$word_freq_free > 0)
p2_nb <- mean(spam_nb$word_freq_money > 0)
joint_nb <- mean(spam_nb$word_freq_free > 0 & spam_nb$word_freq_money > 0)
c(joint = joint_nb, product = p1_nb * p2_nb)
```

```
    joint   product
0.2410370 0.2049028
```

Among spam, 54.6% of emails contain *free* and 37.6% contain *money*, so under the assumption 20.5% would contain both. The data show 24.1%. The assumption is wrong here, since the two words tend to appear together. A naive Bayes classifier uses the product anyway, and it still often classifies well.

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

The assumption is rarely exactly true, which is why it is called naive. Its value is in how few parameters it needs.

[17] https://morrison-lab.github.io/pds/random-variables.html#def-joint-distribution
[18] https://morrison-lab.github.io/pds/independence.html#def-cind

Example 4.2 (Counting parameters)**.**
Suppose the class[a] Y takes K values and each of p features is binary.
Without the naive Bayes assumption, $\Pr(X_1, \ldots, X_p \mid Y = k)$ is an arbitrary distribution[b] over 2^p possible values of the binary features, for each class k. It needs $2^p - 1$ free numbers per class, since the probabilities must sum to one. With the assumption, each $\Pr(X_j \mid Y = k)$ is a single Bernoulli probability, so each class needs only p numbers.

```
# The number of free parameters in the class-conditional distributions
# of K classes with D binary features, with and without the assumption.
k_classes <- 2
d_features <- c(5, 20, 30)
full <- k_classes * (2^d_features - 1)
naive <- k_classes * d_features
data.frame(
  D = d_features,
  full = format(full, scientific = FALSE),
  naive = naive
)
```

```
# A tibble: 3 x 3
      D full         naive
  <dbl> <chr>        <dbl>
1     5 "        62"    10
2    20 "   2097150"    40
3    30 "2147483646"    60
```

With 20 binary features and 2 classes, the full model has about 2,097,150 parameters, more than most datasets have observations. The naive Bayes model has 40. The number grows exponentially in p without the assumption, and linearly in p with it.

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

Definition 4.2 (Naive Bayes classifier)**.** A **naive Bayes classifier** is the generative classifier whose class-conditional density[a] has one factor for each feature, by the naive Bayes assumption,

$$f_k(\underset{\sim}{x}) = \prod_{j=1}^{p} f_{kj}(x_j), \tag{5}$$

with each one-feature density f_{kj} estimated[b] from the training points of class[c] k [5, Sec. 4.4.4], [13, Sec. 10.1], [14]. The naive Bayes classifier predicts the class with the largest $\hat{\pi}_k \prod_{j=1}^{p} \hat{f}_{kj}(x_j)$.

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

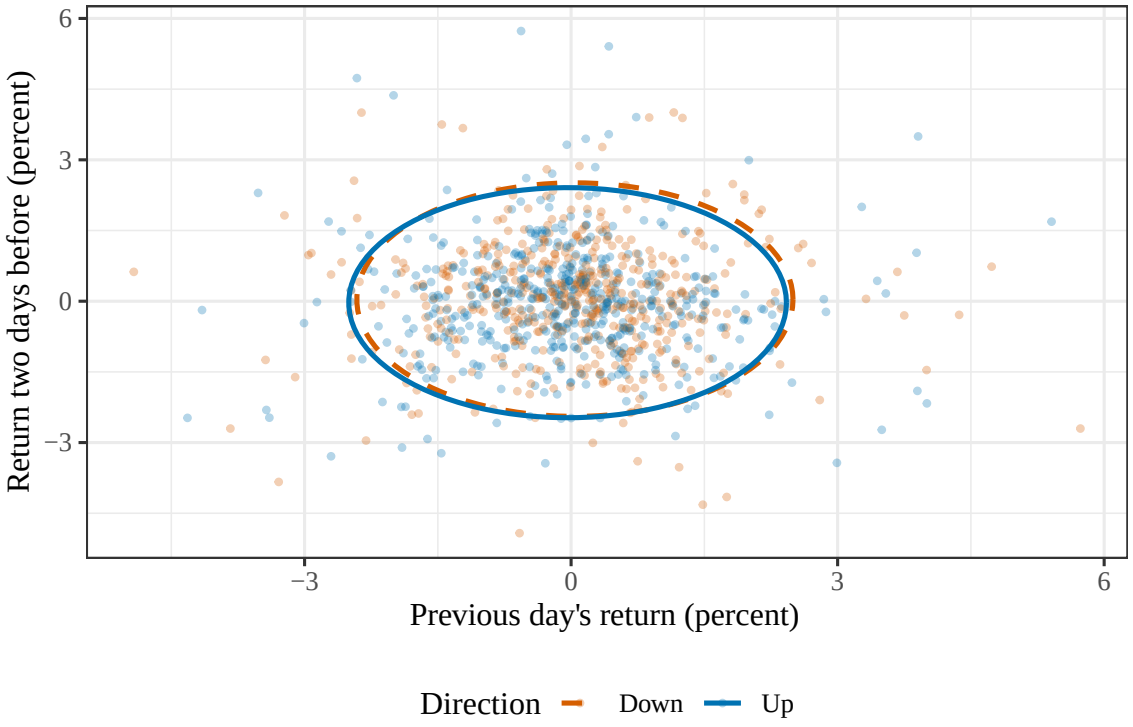
Definition 4.3 (Gaussian naive Bayes classifier)**.** A **Gaussian naive Bayes classifier** is the naive Bayes classifier that takes each f_{kj} to be a Gaussian density[a] with its own mean[b] μ_{kj} and variance[c] σ_{kj}^2, estimated[d] by the class-k sample mean[e] and sample variance[f] of feature j.

[a] https://morrison-lab.github.io/pds/random-variables.html#def-normal
[b] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[c] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance
[d] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[e] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean
[f] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-variance

Example 4.3 (Gaussian naive Bayes is QDA with diagonal covariance matrices)**.** Consider a Gaussian naive Bayes classifier with p features. Its class-conditional density[a] multiplies p normal densities[b] with means[c] μ_{kj} and variances[d] σ_{kj}^2. That multiplication gives the multivariate normal[e] density with mean[f] vector $\vec{\mu}_k = (\mu_{k1}, \ldots, \mu_{kp})$ and diagonal[g] covariance matrix[h] $\Sigma_k = \text{diag}\left(\sigma_{k1}^2, \ldots, \sigma_{kp}^2\right)$. So the classifier[i] is QDA with each Σ_k restricted to be diagonal.
For the `Smarket` days before 2005, with features `Lag1` and `Lag2`, the ellipses of Figure 9 have axes parallel to the coordinate axes, because each Σ_k is diagonal.

4.0.3 R

```
library(dplyr) # nolint: undesirable_function_linter
library(ggplot2) # nolint: undesirable_function_linter
angle <- seq(0, 2 * pi, length.out = 100)
nbc_angles <- tibble(angle = angle)
nbc_class_stats <- da_train |>
  summarize(
    .by = Direction,
    Lag1_mean = mean(Lag1),
    Lag1_sd = sd(Lag1),
    Lag2_mean = mean(Lag2),
    Lag2_sd = sd(Lag2)
  )
nbc_ellipses <- nbc_class_stats |>
  cross_join(nbc_angles) |>
  mutate(
    Lag1 = Lag1_mean + 2 * Lag1_sd * cos(angle),
    Lag2 = Lag2_mean + 2 * Lag2_sd * sin(angle)
  )
nbc_colours <- c(Up = "#0072B2", Down = "#D55E00")
nbc_linetypes <- c(Up = "solid", Down = "dashed")
da_train |>
  ggplot() +
  aes(Lag1, Lag2, colour = Direction, linetype = Direction) +
  geom_point(alpha = 0.3, size = 0.8) +
  geom_path(data = nbc_ellipses, linewidth = 0.9) +
  scale_colour_manual(values = nbc_colours) +
  scale_linetype_manual(values = nbc_linetypes) +
  labs(
    x = "Previous day's return (percent)",
    y = "Return two days before (percent)"
  )
```

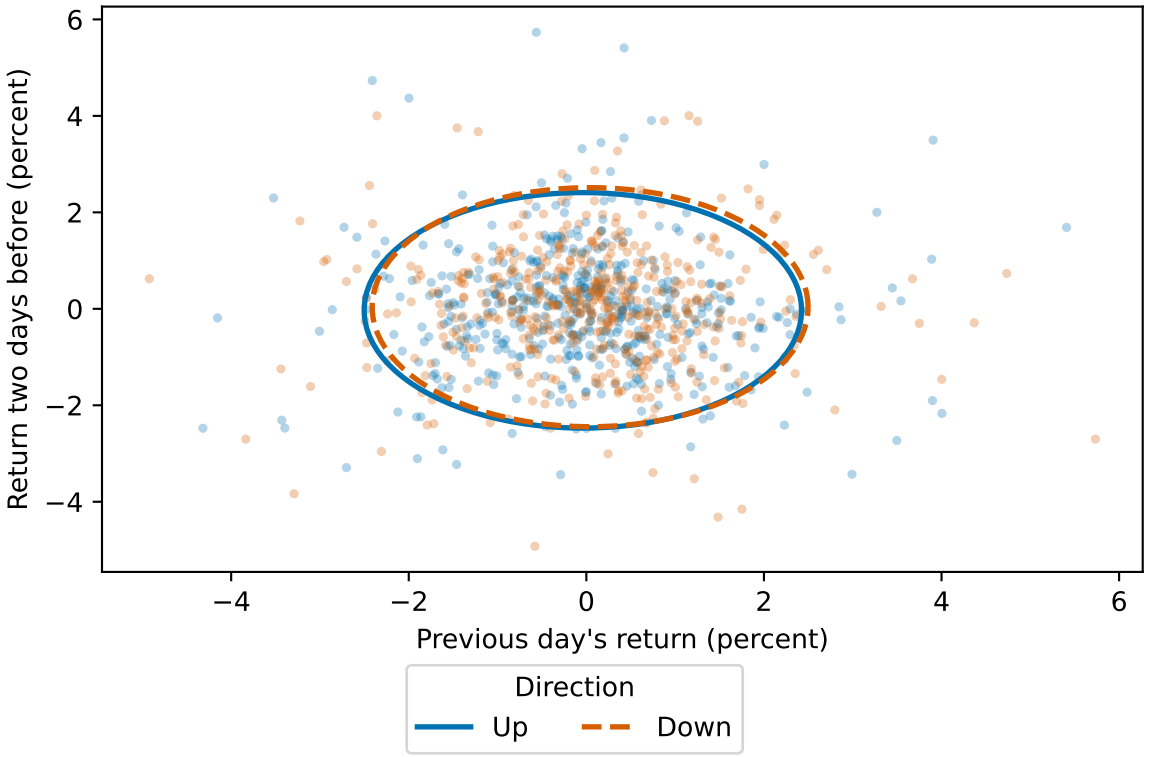

Return two days before (percent)
6
3
0
−3
−3
0
3
6
Previous day's return (percent)
Direction
Down
Up

(a)

4.0.4 Python

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

angle_nbc = np.linspace(0, 2 * np.pi, 100)
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
for direction, colour, style in [("Up", "#0072B2", "-"), ("Down", "#D55E00", "--")]:
    grp = da_train_py[da_train_py["Direction"] == direction]
    ax.scatter(grp["Lag1"], grp["Lag2"], s=6, alpha=0.3, color=colour);
    ax.plot(grp["Lag1"].mean() + 2 * grp["Lag1"].std() * np.cos(angle_nbc),
            grp["Lag2"].mean() + 2 * grp["Lag2"].std() * np.sin(angle_nbc),
            color=colour, linestyle=style, linewidth=2, label=direction);
ax.set_xlabel("Previous day's return (percent)");
ax.set_ylabel("Return two days before (percent)");
fig.legend(title="Direction", loc="outside lower center", ncol=2);
plt.show()
```



(b)

Figure 9: The class-k Gaussian naive Bayes density[j] contour two standard deviations[k] from the mean[l], for each of the two classes[m].

[a] https://morrison-lab.github.io/pds/random-variables.html#def-pdf
[b] https://morrison-lab.github.io/pds/random-variables.html#def-normal
[c] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[d] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance
[e] https://morrison-lab.github.io/pds/distributions.html#def-mvn
[f] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[g] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-diagonal-matrix
[h] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x
[i] https://morrison-lab.github.io/sds/classification.html#def-classification
[j] https://morrison-lab.github.io/pds/random-variables.html#def-pdf
[k] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-sd

[l] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean
[m] https://morrison-lab.github.io/sds/classification.html#def-class

Definition 4.4 (K-nearest neighbors classifier)**.** For a positive integer[a] K of neighbors and a feature vector $\underset{\sim}{x}$, the **K-nearest neighbors classifier** finds the K training points closest to $\underset{\sim}{x}$ in Euclidean distance[b], and predicts the most frequent class[c] among them [5, Sec. 4.7.6]. The K-nearest neighbors classifier estimates[d] $\Pr(Y = k \mid \underset{\sim}{X} = \underset{\sim}{x})$ by the fraction[e] of those K points in class k, so K-nearest neighbors makes no assumption about the form of f_k. The nonparametric models chapter studies K-nearest neighbors further.

[a] https://morrison-lab.github.io/mcs/number-systems.html#def-positive-integer
[b] https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-euclidean-distance
[c] https://morrison-lab.github.io/sds/classification.html#def-class
[d] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[e] https://morrison-lab.github.io/mds/algebra-sums.html#def-fraction

Example 4.4 (The nearest neighbors of one Smarket day)**.**
Consider a day on which `Lag1` was 0.5 and `Lag2` was −0.5. For $K = 5$, the five `Smarket` training days closest to that point in Euclidean distance[a] are the neighbors the K-nearest neighbors classifier counts.

4.0.5 R

```
knn_query <- c(Lag1 = 0.5, Lag2 = -0.5)
knn_diff1 <- da_train$Lag1 - knn_query[["Lag1"]]
knn_diff2 <- da_train$Lag2 - knn_query[["Lag2"]]
knn_dist <- sqrt(knn_diff1^2 + knn_diff2^2)
knn_nearest <- da_train$Direction[order(knn_dist)[1:5]]
table(knn_nearest)
```

```
knn_nearest
Down   Up
   3    2
```

4.0.6 Python

```
import numpy as np

knn_dist_py = np.hypot(r.da_train["Lag1"] - 0.5, r.da_train["Lag2"] + 0.5)
knn_nearest_py = r.da_train["Direction"].iloc[np.argsort(knn_dist_py)[:5]]
knn_nearest_py.value_counts()
```

```
Direction
Down    3
Up      2
Name: count, dtype: int64
```

Of the five neighbors, 2 were up days and 3 were down days, so the classifier[b] estimates[c] $\Pr(Y = \text{Up} \mid \underset{\sim}{X} = \underset{\sim}{x})$ as 0.4 and predicts Down.

[a] https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-euclidean-distance
[b] https://morrison-lab.github.io/sds/classification.html#def-classification
[c] https://morrison-lab.github.io/sds/estimation.html#def-estimate

5 Comparing the methods

Exercise 5.1 (Counting parameters)**.** Consider a classification problem with K classes and p features. A symmetric[a] $p \times p$ matrix has $p(p+1)/2$ free entries. Count the free parameters of LDA (Definition 2.2).

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

Solution

Solution 5.1. LDA has $K-1$ free priors, because the priors sum to one, and Kp entries in the K class[a] means[b]. Its one covariance matrix[c] is symmetric[d], so it adds $p(p+1)/2$ free entries. The total is $(K-1) + Kp + p(p+1)/2$ parameters.

[a] https://morrison-lab.github.io/sds/classification.html#def-class
[b] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[c] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x
[d] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-symmetric-matrix

Exercise 5.2 (Counting parameters (part 2))**.** Use the setup of Exercise 5.1.
Count the free parameters of QDA (Definition 3.3).

Solution

Solution 5.2. QDA has the same $K-1$ priors and Kp mean[a] entries as LDA, and K covariance matrices[b] instead of one: $(K-1) + Kp + K\,p(p+1)/2$ parameters.

[a] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[b] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x

Exercise 5.3 (Counting parameters (part 3))**.** Use the setup of Exercise 5.1.
Count the free parameters of the naive Bayes assumption with a Gaussian density[a] for each feature in each class[b]. This assumption makes each class covariance matrix[c] diagonal[d].

[a] https://morrison-lab.github.io/pds/random-variables.html#def-normal
[b] https://morrison-lab.github.io/sds/classification.html#def-class
[c] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x
[d] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-diagonal-matrix

Solution

Solution 5.3. Gaussian naive Bayes has the same $K-1$ priors and Kp mean[a] entries, and K diagonal[b] covariance matrices[c], which hold Kp variances[d]: $(K-1) + 2Kp$ parameters.

[a] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[b] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-diagonal-matrix
[c] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x
[d] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance

Exercise 5.4 (Counting parameters (part 4))**.** Use the setup of Exercise 5.1.
Count the free parameters of multinomial logistic regression with a reference class[a], as in the second softmax exercise, so that it has $K-1$ coefficient vectors, each with an intercept[b].

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

Solution

Solution 5.4. Each of the $K-1$ coefficient vectors has an intercept[a] and p slopes[b], so multinomial logistic regression has $(K-1)(p+1)$ parameters.

[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 5.5 (Counting parameters (part 5))**.** Use the setup of Exercise 5.1.
Which counts grow fastest in p?

Solution

Solution 5.5. As p grows, the LDA count grows like $p^2/2$, and the QDA count like $Kp^2/2$. The naive Bayes and logistic counts grow only linearly in p. Each QDA covariance matrix[a] is estimated[b] from only the n_k points of its own class[c], so when the classes[d] are small, QDA's estimates[e] vary more from one training set to another than those of any of the other models.

[a] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x
[b] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[c] https://morrison-lab.github.io/sds/classification.html#def-class
[d] https://morrison-lab.github.io/sds/classification.html#def-class
[e] https://morrison-lab.github.io/sds/estimation.html#def-estimate

Theorem 5.1 (Parameter counts of the classifiers)**.** *With K classes and p features, the number of free parameters is*

Table 1: The number of free parameters of each method.

Model	*Parameters*
LDA	$(K-1)+Kp+p(p+1)/2$
QDA	$(K-1)+Kp+K\,p(p+1)/2$
Gaussian naive Bayes	$(K-1)+2Kp$
Multinomial logistic regression	$(K-1)(p+1)$

Proof

Proof. By Solution 5.1, Solution 5.2, Solution 5.3, and Solution 5.4. □

Example 5.1 (Parameters for two classes)**.**
Count the parameters of Theorem 5.1 for $K=2$ classes[a] and several numbers of features p, including $p=2$, for two lagged returns, and the $p=6$ of the Smarket fit exercise.

5.0.1 R

```r
k_classes <- 2
p_features <- c(2, 6, 50, 100)
counts <- data.frame(
  p = p_features,
  LDA = (k_classes - 1) + k_classes * p_features +
    p_features * (p_features + 1) / 2,
  QDA = (k_classes - 1) + k_classes * p_features +
    k_classes * p_features * (p_features + 1) / 2,
  `Gaussian naive Bayes` = (k_classes - 1) + 2 * k_classes * p_features,
  logistic = (k_classes - 1) * (p_features + 1),
  check.names = FALSE
)
counts
```

```
# A tibble: 4 x 5
      p   LDA   QDA `Gaussian naive Bayes` logistic
  <dbl> <dbl> <dbl>                  <dbl>    <dbl>
1     2     8    11                      9        3
2     6    34    55                     25        7
3    50  1376  2651                    201       51
4   100  5251 10301                    401      101
```

5.0.2 Python

```python
K_py = 2
p_py = np.array([2, 6, 50, 100])
counts_py = pd.DataFrame({
    "p": p_py,
    "LDA": (K_py - 1) + K_py * p_py + p_py * (p_py + 1) // 2,
    "QDA": (K_py - 1) + K_py * p_py + K_py * p_py * (p_py + 1) // 2,
    "Gaussian naive Bayes": (K_py - 1) + 2 * K_py * p_py,
    "logistic": (K_py - 1) * (p_py + 1),
})
print(counts_py.to_string(index=False))
```

```
  p  LDA   QDA  Gaussian naive Bayes  logistic
  2    8    11                     9         3
  6   34    55                    25         7
 50 1376  2651                   201        51
100 5251 10301                   401       101
```

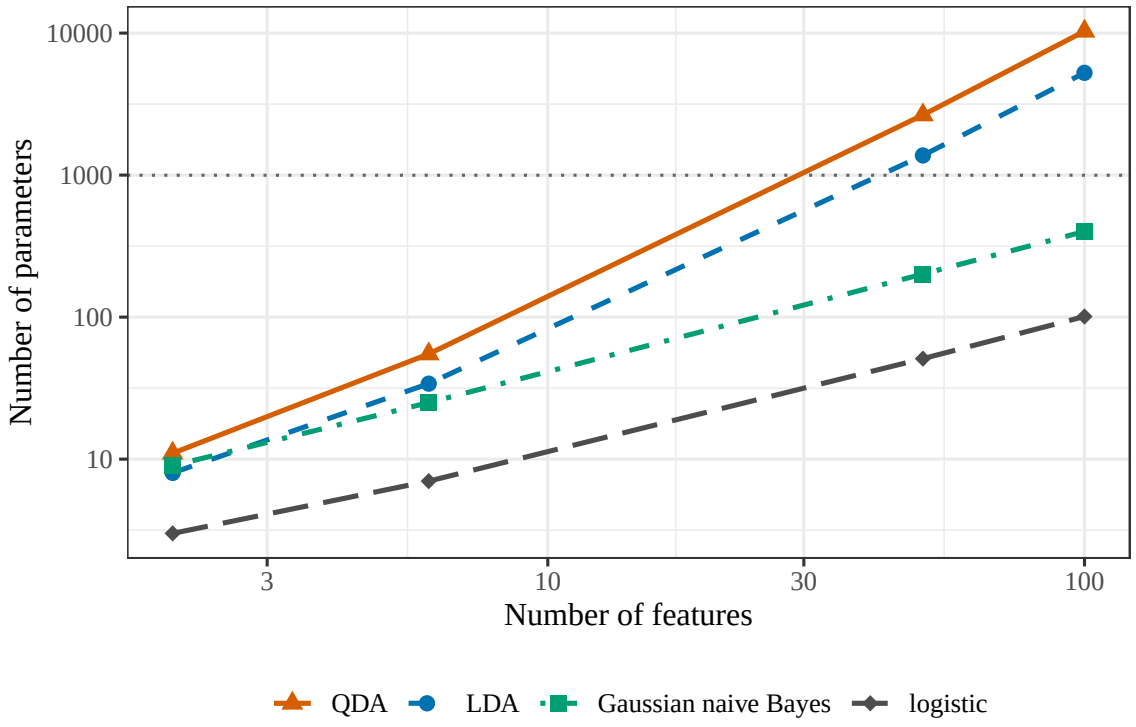
At $p = 50$ the QDA model has 2,651 parameters and LDA has 1,376, against 51 for logistic regression. The `Smarket` training set has 998 days, which is 0.38 observations per QDA parameter at $p = 50$. At $p = 100$ QDA has 10,301 parameters, more than the 998 training days.

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

5.0.3 R

```r
library(ggplot2)
dpf_long <- reshape(
  counts, direction = "long", idvar = "p",
  varying = list(c("LDA", "QDA", "Gaussian naive Bayes", "logistic")),
  v.names = "parameters", timevar = "method",
  times = c("LDA", "QDA", "Gaussian naive Bayes", "logistic")
)
dpf_long$method <- factor(dpf_long$method, levels = c("QDA", "LDA", "Gaussian naive Bayes", "logistic"))
```

```r
ggplot(dpf_long) +
  aes(p, parameters, colour = method, linetype = method, shape = method) +
  geom_hline(yintercept = nrow(da_train), linetype = "dotted", colour = "grey40") +
  geom_line(linewidth = 0.9) +
  geom_point(size = 2.5) +
  scale_x_log10() +
  scale_y_log10() +
  scale_colour_manual(values = c(
    QDA = "#D55E00", LDA = "#0072B2", `Gaussian naive Bayes` = "#009E73", logistic = "grey30"
  )) +
  scale_linetype_manual(values = c(
    QDA = "solid", LDA = "dashed", `Gaussian naive Bayes` = "dotdash", logistic = "longdash"
  )) +
  scale_shape_manual(values = c(QDA = 17, LDA = 16, `Gaussian naive Bayes` = 15, logistic = 18)) +
  labs(x = "Number of features", y = "Number of parameters", colour = NULL, linetype = NULL, shape = NUL
```



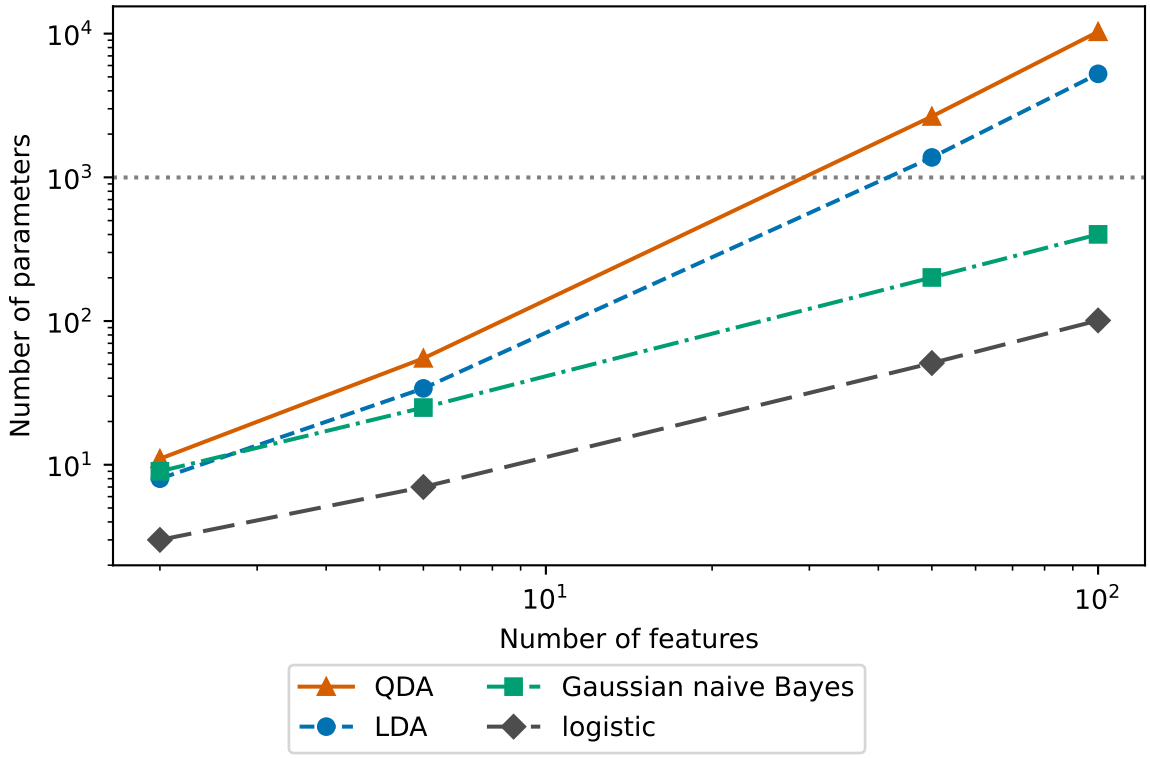
(a)

5.0.4 Python

```python
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.axhline(len(da_train_py), color="grey", linestyle=":");
for method, colour, style, marker in [
    ("QDA", "#D55E00", "-", "^"), ("LDA", "#0072B2", "--", "o"),
    ("Gaussian naive Bayes", "#009E73", "-.", "s"), ("logistic", "#4d4d4d", (0, (8, 3)), "D"),
]:
    ax.plot(counts_py["p"], counts_py[method], color=colour, linestyle=style, marker=marker,
            label=method);
ax.set_xscale("log");
ax.set_yscale("log");
```

```
ax.set_xlabel("Number of features");
ax.set_ylabel("Number of parameters");
fig.legend(loc="outside lower center", ncol=2);
plt.show()
```



(b)

Figure 10: The parameter counts of Example 5.1 for two classes, with the number of Smarket training days (dotted line). The quadratic model has more parameters than there are training days somewhere between 6 and 50 features.

Exercise 5.6 (Choosing among the classifiers)**.** LDA, QDA, Gaussian naive Bayes, and logistic regression are all linear[a] or quadratic[b] classifier[c]s fitted to the same data. For each situation in this exercise and the next three, state which you would try first, and why.
There are $n = 60$ observations in two classes[d] and $p = 40$ features.

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

Solution

Solution 5.6. Gaussian naive Bayes or a regularized version of logistic regression, not QDA, and LDA only with caution.

- With about 30 points per class[a], if the classes[b] are about equal in size, each QDA class covariance matrix[c] is singular, because a sample[d] covariance matrix from no more than $p = 40$ points is singular (corollary in the mathematics notes[e]).
- The pooled covariance matrix of LDA is invertible[f], since it has 58 degrees of freedom of within-class variation, more than 40, but it estimates[g] 820 free entries from them, so its estimate[h] is noisy.

- The diagonal[i] covariance matrices[j] of naive Bayes have only p variances[k] per class, so their estimates are the most stable of the three.

[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/variance-covariance.html#def-cov-vec-x
[d] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample
[e] https://morrison-lab.github.io/mds/linear-algebra-rank-nullity.html#cor-sample-covariance-singular
[f] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-matrix-inverse
[g] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[h] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[i] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-diagonal-matrix
[j] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x
[k] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance

Exercise 5.7 (Choosing among the classifiers (part 2))**.** Use the setup of Exercise 5.6. There are $n = 100{,}000$ observations and $p = 3$ features, and a plot shows that the within-class spread of the features differs a lot between classes.

Solution

Solution 5.7. QDA. With 3 features and two classes[a], QDA has 19 parameters and LDA 13, by Theorem 5.1, and they are estimated[b] from 100,000 points. The unequal spreads are structure that LDA's single covariance matrix[c] cannot represent, so LDA is biased here, and QDA can represent it.

[a] https://morrison-lab.github.io/sds/classification.html#def-class
[b] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[c] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x

Exercise 5.8 (Choosing among the classifiers (part 3))**.** Use the setup of Exercise 5.6. Some features are binary indicators, such as whether a word appears in an email.

Solution

Solution 5.8. Not LDA or QDA. They model the features with a Gaussian density, which is a poor model of a binary feature. Logistic regression does not model the distribution[a] of the features, so binary features need no density, and naive Bayes can use a Bernoulli[b] density for each one, as in the parameter count for binary features.

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

Exercise 5.9 (Choosing among the classifiers (part 4))**.** Use the setup of Exercise 5.6. How is the choice a bias-variance trade-off?

Solution

Solution 5.9. Moving from LDA to QDA adds parameters. That lowers the bias[a] when the classes[b] really differ in their covariance matrices[c], and it raises the variance[d] of the estimates[e], because each class[f] covariance matrix[g] is estimated[h] from that class alone. LDA accepts bias from forcing one covariance matrix on all classes, in exchange for estimating only one matrix, from all n points. Gaussian naive Bayes restricts each class covariance matrix to be diagonal[i] instead, so it has the fewest parameters of the three generative models when p is large, and it is biased when the features are correlated within a class. QDA is the better choice when n is large relative to its number of parameters in Theorem 5.1, and LDA when n is small [2, Ch. 4]. Regularized discriminant analysis compromises between the two by shrinking each $\hat{\Sigma}_k$ toward the pooled $\hat{\Sigma}$ [8, Sec. 4.3.1].

[a] https://morrison-lab.github.io/sds/estimation.html#def-bias
[b] https://morrison-lab.github.io/sds/classification.html#def-class
[c] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x
[d] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance
[e] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[f] https://morrison-lab.github.io/sds/classification.html#def-class
[g] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x
[h] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[i] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-diagonal-matrix

6 Lab: discriminant analysis, naive Bayes and nearest neighbors on the stock market data

Exercise 6.1 (LDA and QDA on the stock market)**.** Use the `Smarket` days of the Smarket fit exercise. Fit LDA and QDA for the direction of the market with `Lag1` and `Lag2` as the only predictors, on the days before 2005. Predict each day of 2005 and tabulate the predictions against what happened. What are the fitted class proportions and class means[a]?

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

Solution

Solution 6.1.

6.0.1 R

```
da_test <- da[da$Year == 2005, ]
lda2 <- MASS::lda(Direction ~ Lag1 + Lag2, data = da_train)
qda2 <- MASS::qda(Direction ~ Lag1 + Lag2, data = da_train)
# The six-predictor logistic regression of the Smarket fit exercise, scored on the same days.
glm_fit <- glm(
  I(Direction == "Up") ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
  data = da_train,
  family = binomial
)
pred_glm <- ifelse(predict(glm_fit, da_test, type = "response") > 0.5, "Up", "Down")
acc_glm <- mean(pred_glm == da_test$Direction)

lda2$prior
```

```
    Down       Up
0.491984 0.508016
```

```
lda2$means
```

```
            Lag1        Lag2
Down  0.04279022  0.03389409
Up   -0.03954635 -0.03132544
```

```
pred_lda <- predict(lda2, da_test)$class
pred_qda <- predict(qda2, da_test)$class
table(LDA = pred_lda, actual = da_test$Direction)
```

```
      actual
LDA    Down  Up
  Down   35  35
  Up     76 106
```

```r
table(QDA = pred_qda, actual = da_test$Direction)
```

```
      actual
QDA    Down  Up
  Down   30  20
  Up     81 121
```

```r
acc <- c(
  LDA = mean(pred_lda == da_test$Direction),
  QDA = mean(pred_qda == da_test$Direction),
  always_up = mean(da_test$Direction == "Up")
)
round(acc, 3)
```

```
      LDA       QDA always_up
    0.560     0.599     0.560
```

```r
# 95% intervals for each accuracy, and for the paired differences between rules
# scored on the same days, as for the logistic regression of the Smarket fit exercise.
wald_ci <- function(p, n) {
  p + c(lower = -1, upper = 1) * qnorm(0.975) * sqrt(p * (1 - p) / n)
}
paired_ci <- function(a, b) {
  d <- as.integer(a) - as.integer(b)
  mean(d) + c(lower = -1, upper = 1) * qt(0.975, length(d) - 1) * sd(d) / sqrt(length(d))
}
ci_acc <- sapply(acc, wald_ci, n = nrow(da_test))
hit_lda <- pred_lda == da_test$Direction
hit_qda <- pred_qda == da_test$Direction
hit_up <- da_test$Direction == "Up"
ci_qda_up <- paired_ci(hit_qda, hit_up)
ci_qda_lda <- paired_ci(hit_qda, hit_lda)
```

6.0.2 Python

```python
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

features_py = ["Lag1", "Lag2"]
da_test_py = da_py[da_py["Year"] == 2005]
lda2_py = LinearDiscriminantAnalysis().fit(da_train_py[features_py], da_train_py["Direction"])
qda2_py = QuadraticDiscriminantAnalysis().fit(da_train_py[features_py], da_train_py["Direction"])

print("priors:", dict(zip(lda2_py.classes_.tolist(), lda2_py.priors_.round(3).tolist())))
```

```
priors: {'Down': 0.492, 'Up': 0.508}
```

```python
print(pd.DataFrame(lda2_py.means_, index=lda2_py.classes_, columns=features_py).round(3))
```

```
       Lag1   Lag2
Down  0.043  0.034
Up   -0.040 -0.031
```

```python
for name, model in [("LDA", lda2_py), ("QDA", qda2_py)]:
    pred = model.predict(da_test_py[features_py])
    acc_py = float((pred == da_test_py["Direction"]).mean())
    # A 95% interval for the accuracy: the proportion plus or minus 1.96 standard errors.
    half_py = 1.96 * np.sqrt(acc_py * (1 - acc_py) / len(da_test_py))
    print(name, "accuracy:", round(acc_py, 3), [round(float(acc_py - half_py), 3), round(float(acc_py
    print(pd.crosstab(pred, da_test_py["Direction"], rownames=["predicted"], colnames=["actual"]))
```

```
LDA accuracy: 0.56 [0.498, 0.621]
actual     Down   Up
predicted
Down         35   35
Up           76  106
QDA accuracy: 0.599 [0.539, 0.66]
actual     Down   Up
predicted
Down         30   20
Up           81  121
```

The classes are balanced, with 50.8% of the training days up. The means[a] of both lags are slightly negative on days that went up and slightly positive on days that went down: -0.040 and 0.043 for `Lag1`, and -0.031 and 0.034 for `Lag2`. The gap between the `Lag1` means, 0.08, is small next to the standard deviation[b] of `Lag1` within a class[c], about 1.23, so the classes overlap almost completely.

[a] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean
[b] https://morrison-lab.github.io/pds/variance-covariance.html#def-sd
[c] https://morrison-lab.github.io/sds/classification.html#def-class

Exercise 6.2 (LDA and QDA on the stock market (part 2))**.** Use the setup of Exercise 6.1.
How often is each model right on the 2005 days? Compare with the rule that always answers Up, and with the logistic regression of its solution.

Solution

Solution 6.2. LDA is right on 56% of the 252 days in 2005 (95% interval [49.8, 62.1]%), QDA on 59.9% ([53.9, 66]%), and always predicting Up on 56% ([49.8, 62.1]%). LDA does no better than the constant[a] rule that always predicts Up, and the logistic regression of its solution, with more predictors, was right on only 48%. QDA exceeds the accuracy of the constant rule by 4 percentage points.

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

Exercise 6.3 (LDA and QDA on the stock market (part 3))**.** Use the setup of Exercise 6.1.
How much trust does the accuracy difference between the two models deserve, given that it comes from one year of data?

Solution

Solution 6.3. Little. Each confidence interval[a] in Solution 6.2 is the accuracy plus or minus 1.96 standard errors[b], and the standard error[c] is about 3.1 percentage points for QDA on 252 days, so the intervals for the three rules overlap almost entirely.
The rules are scored on the same days, so the sharper comparison is day by day. QDA exceeds the accuracy of the constant[d] rule by 4 percentage points (95% interval [-1.5, 9.5]), an interval that includes 0. It exceeds the accuracy of LDA by 4 percentage points ([0.5, 7.4]), an interval that excludes 0.
These intervals reflect sampling variation across the test days in 2005, not across training sets, and they do not account for comparing several rules on the same days before reporting the best. Showing that QDA has higher out-of-sample accuracy would need more test days or repeated splits.

[a] https://morrison-lab.github.io/sds/inference.html#def-confidence-interval
[b] https://morrison-lab.github.io/sds/estimation.html#def-SE
[c] https://morrison-lab.github.io/sds/estimation.html#def-SE
[d] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function

```
library(ggplot2)
grid <- expand.grid(
```

```
  Lag1 = seq(-4, 4, length.out = 120),
  Lag2 = seq(-4, 4, length.out = 120)
)
grid_post <- rbind(
  transform(grid, model = "LDA", p_up = predict(lda2, grid)$posterior[, "Up"]),
  transform(grid, model = "QDA", p_up = predict(qda2, grid)$posterior[, "Up"])
)
ggplot(grid_post) +
  aes(Lag1, Lag2) +
  geom_raster(aes(fill = p_up)) +
  geom_contour(aes(z = p_up), breaks = 0.5, colour = "black") +
  geom_point(
    data = da_test,
    aes(shape = Direction),
    colour = "grey20", alpha = 0.7
  ) +
  scale_fill_gradient2(midpoint = 0.5, limits = c(0, 1), name = "P(Up)") +
  facet_wrap(~model) +
  coord_cartesian(xlim = c(-4, 4), ylim = c(-4, 4))
```

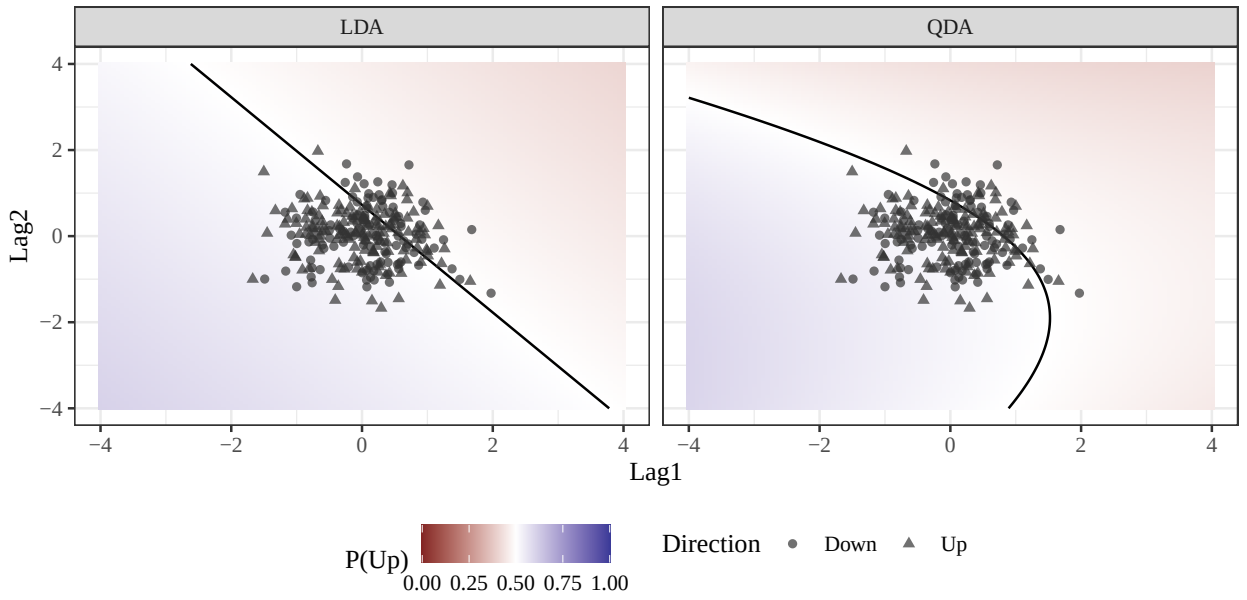


Figure 11: Predicted probability of an up day from the LDA and QDA fits of Solution 6.0.2, over the plane of the two lagged returns. The black line and curve are the decision boundaries, where the probability is one half. The points are the days of 2005.

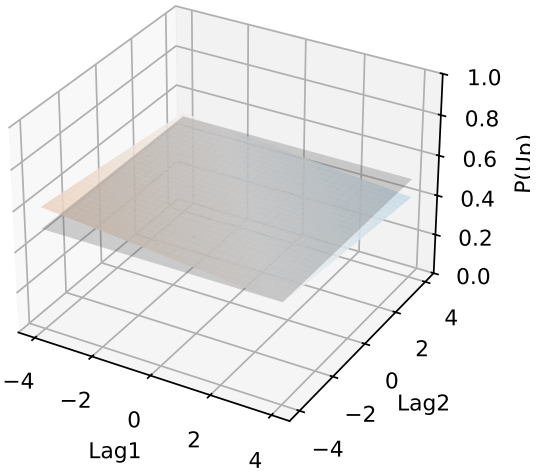
6.0.3 R

(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
from matplotlib.colors import LinearSegmented
from sklearn.discriminant_analysis import (
    LinearDiscriminantAnalysis,
    QuadraticDiscriminantAnalysis,
)

sm_train_3d = r.da_train
sm_x_3d = sm_train_3d[["Lag1", "Lag2"]].to_n
sm_y_3d = sm_train_3d["Direction"].astype(st
sm_axis_3d = np.linspace(-4, 4, 60)
sm_lag1_3d, sm_lag2_3d = np.meshgrid(sm_axis
sm_points_3d = np.column_stack([sm_lag1_3d.r
sm_cmap_3d = LinearSegmentedColormap.from_lis
    "sm", ["#0072B2", "#F2F2F2", "#D55E00"]
)
sm_lda_model = LinearDiscriminantAnalysis().f
# `classes_` is sorted, so the column of "Up"
sm_lda_up = sm_lda_model.predict_proba(sm_poi
sm_lda_z_3d = sm_lda_up.reshape(sm_lag1_3d.sh
fig = plt.figure(figsize=(4, 4))
ax = fig.add_subplot(1, 1, 1, projection="3d"
ax.plot_surface(sm_lag1_3d, sm_lag2_3d, sm_ld
                vmin=0, vmax=1, alpha=0.9);
ax.plot_surface(sm_lag1_3d, sm_lag2_3d, np.fu
                color="grey", alpha=0.3);
ax.set_xlabel("Lag1");
ax.set_ylabel("Lag2");
ax.set_zlabel("P(Up)");
ax.set_zlim(0, 1);
plt.show();
```



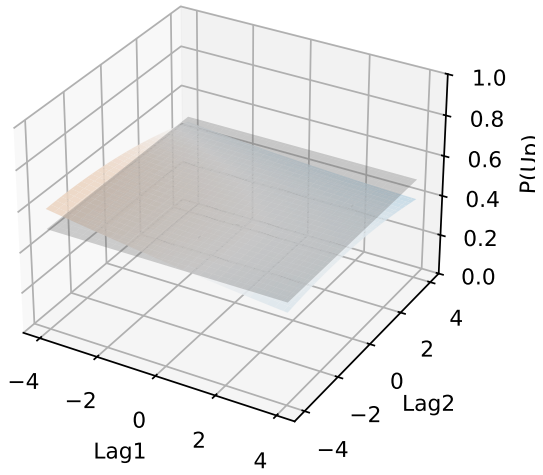
(a) The LDA surface.

6.0.5 R

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

6.0.6 Python

```
sm_qda_model = QuadraticDiscriminantAnalysis().fit(sm_x_3d,
sm_qda_up = sm_qda_model.predict_proba(sm_points_3d)[:, 1]
sm_qda_z_3d = sm_qda_up.reshape(sm_lag1_3d.shape)
fig = plt.figure(figsize=(4, 4))
ax = fig.add_subplot(1, 1, 1, projection="3d")
ax.plot_surface(sm_lag1_3d, sm_lag2_3d, sm_qda_z_3d, cmap=s
                vmin=0, vmax=1, alpha=0.9);
ax.plot_surface(sm_lag1_3d, sm_lag2_3d, np.full_like(sm_qda
                color="grey", alpha=0.3);
ax.set_xlabel("Lag1");
ax.set_ylabel("Lag2");
ax.set_zlabel("P(Up)");
ax.set_zlim(0, 1);
plt.show();
```

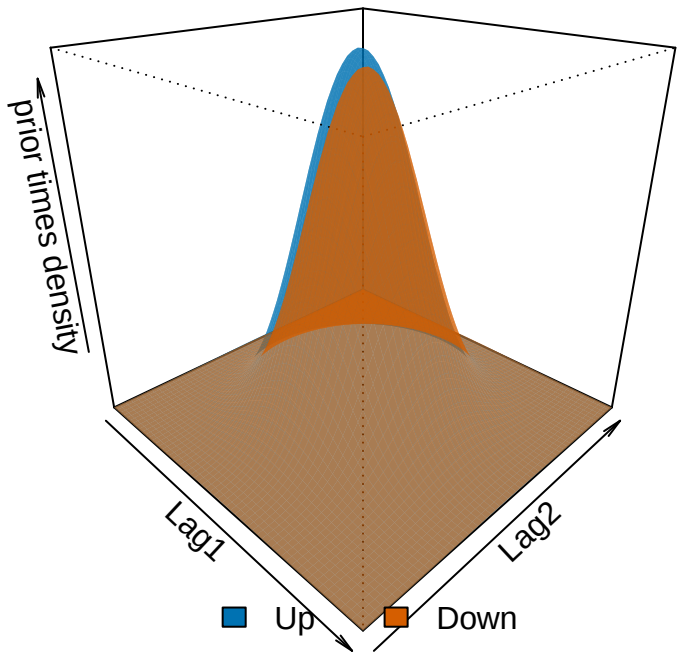


(b) The QDA surface.

Figure 12: The graph of the predicted probability[19] of an up day of Figure 11, as a surface over the plane of the two lagged returns. The gray plane is at probability one half, and the decision boundary is where it cuts the surface: a line for LDA and a curve for QDA.

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

6.0.7 R



6.0.8 Python

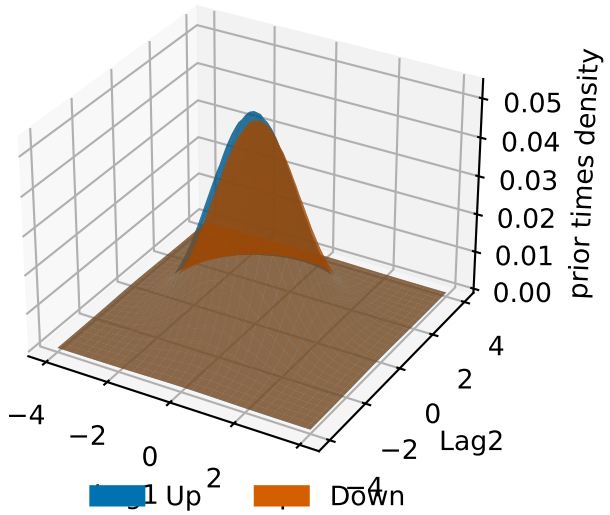


Figure 13: The generative view of the QDA fit of Figure 12b: each class[20]'s prior[21] times its Gaussian density[22] over the plane of the two lagged returns. By Bayes' rule[23], the predicted probability[24] of an up day is the height of the blue surface as a fraction[25] of the sum[26] of the two heights. The two surfaces almost coincide, so that fraction stays close to one half, as in Figure 12b.

Exercise 6.4 (Lab: split the Smarket data by year)**.** Split the `Smarket` data [5] into a training set of the days before 2005 and a test set of the days in 2005. Keep `Lag1` and `Lag2` as the predictors and `Direction` as the outcome.

Hint
A mask on a column keeps the matching rows, and a selection of columns gives the predictors and the outcome.

Solution

Solution 6.4.

6.0.9 Python

```
import pandas as pd

smarket = pd.read_csv("data/smarket.csv")
predictors = ["Lag1", "Lag2"]
is_train = smarket["Year"] < 2005
x_train = smarket.loc[is_train, predictors]
x_test = smarket.loc[~is_train, predictors]
y_train = smarket.loc[is_train, "Direction"]
y_test = smarket.loc[~is_train, "Direction"]
len(y_train), len(y_test)
```

```
(998, 252)
```

6.0.10 R

```
smarket <- readr::read_csv("data/smarket.csv", show_col_types = FALSE)
is_train <- smarket$Year < 2005
lag_cols <- c("Lag1", "Lag2")
x_train <- smarket[is_train, lag_cols]
x_test <- smarket[!is_train, lag_cols]
y_train <- smarket$Direction[is_train]
y_test <- smarket$Direction[!is_train]
n_train <- length(y_train)
n_test <- length(y_test)
c(n_train, n_test)
```

```
[1] 998 252
```

Exercise 6.5 (Lab: find the test error at K = 1)**.** Fit a K-nearest neighbors classifier with $K = 1$ to the training set of Solution 6.0.10, and find its test error.

Hint
Every estimator is fit with `.fit` on the training set and used with `.predict` on the test set. In R, `class::knn()` does both in one call. The test error is the fraction of test days on which the prediction differs from `Direction`.

Solution

Solution 6.5.

6.0.11 Python

```python
from sklearn.neighbors import KNeighborsClassifier

knn1_py = KNeighborsClassifier(n_neighbors=1).fit(x_train, y_train)
knn1_py_error = float((knn1_py.predict(x_test) != y_test).mean())
knn1_py_error
```

```
0.5
```

6.0.12 R

```r
set.seed(1)
knn1_pred <- class::knn(x_train, x_test, y_train, k = 1)
knn1_r_error <- mean(knn1_pred != y_test)
knn1_r_error
```

```
[1] 0.5
```

The test error at $K = 1$ is 0.50 in the R run and 0.50 in the Python run. Both are near 0.5, the error of a coin flip, which suggests that `Lag1` and `Lag2` carry little information about the direction of the market.

Exercise 6.6 (Lab: find the test error at K = 3)**.** Fit a K-nearest-neighbors classifier with $K = 3$ to the training set of Solution 6.0.10, and find its test error.

Hint

Every estimator is fit with `.fit` on the training set and used with `.predict` on the test set. In R, `class::knn()` does both in one call. Change only the number of neighbors from Exercise 6.5.

Solution

Solution 6.6.

6.0.13 Python

```python
knn3_py = KNeighborsClassifier(n_neighbors=3).fit(x_train, y_train)
knn3_py_error = float((knn3_py.predict(x_test) != y_test).mean())
knn3_py_error
```

```
0.46825396825396826
```

6.0.14 R

```r
knn3_pred <- class::knn(x_train, x_test, y_train, k = 3)
knn3_r_error <- mean(knn3_pred != y_test)
knn3_r_error
```

```
[1] 0.4642857
```

The test error at $K = 3$ is 0.46 in the R run and 0.47 in the Python run.

Exercise 6.7 (Lab: fit a naive Bayes classifier)**.** Fit a Gaussian naive Bayes classifier to the training set of Solution 6.0.10, and report the estimated[a] prior probability of each class[b].

Hint

The naive Bayes assumption lets you summarize each predictor separately within each class[c], and the estimator interface shows where an estimator stores what it learned.

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

Solution

Solution 6.7.

6.0.15 Python

```
from sklearn.naive_bayes import GaussianNB

nb_py = GaussianNB().fit(x_train, y_train)
dict(zip(nb_py.classes_.tolist(), nb_py.class_prior_.round(3).tolist()))
```

```
{'Down': 0.492, 'Up': 0.508}
```

6.0.16 R

```
library(discrim) # nolint: undesirable_function_linter
direction_train <- factor(y_train)
train_nb <- dplyr::bind_cols(x_train, Direction = direction_train)
nb_spec <- naive_Bayes() |>
  set_engine("naivebayes", usekernel = FALSE) |>
  set_mode("classification")
nb_r <- fit(nb_spec, Direction ~ Lag1 + Lag2, data = train_nb)
round(nb_r$fit$prior, 3)
```

```
 Down    Up
0.492 0.508
```

The priors are the fractions of days in the training set on which the market went down and up, so the two runs agree.

Exercise 6.8 (Lab: find the test error of naive Bayes)**.** Find the test error of the naive Bayes classifier from Solution 6.0.16 on the test set of Solution 6.0.10.

Hint

Use `.predict` in Python, as in the scikit-learn estimator interface, or `predict()` in R, then compare the predicted classes[a] with the actual ones on the test set.

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

Solution

Solution 6.8.

6.0.17 Python

```
nb_py_error = float((nb_py.predict(x_test) != y_test).mean())
nb_py_error
```

```
0.40476190476190477
```

6.0.18 R

```
nb_pred <- predict(nb_r, new_data = x_test)$.pred_class
nb_r_error <- mean(nb_pred != y_test)
nb_r_error
```

```
[1] 0.4087302
```

The test error is 0.41 in the R run and 0.40 in the Python run.

7 Further reading

> Read
>
> James et al. [2], Chapter 4, derives these classifiers and compares them with logistic regression. Hastie et al. [8], Section 4.3, covers discriminant analysis, including its regularized versions.

References

[1] B. Hutchinson, "DATA 471/571: Machine learning." Western Washington University, 2024.

[2] 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/

[3] Wikipedia contributors, "Bayes classifier — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Bayes_classifier

[4] Wikipedia contributors, "Generative model — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Generative_model

[5] G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor, *An introduction to statistical learning: With applications in Python*. Springer, 2023. doi: 10.1007/978-3-031-38747-0[27].

[6] Wikipedia contributors, "Linear discriminant analysis — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Linear_discriminant_analysis

[7] W. N. Venables and B. D. Ripley, *Modern applied statistics with S*, 4th ed. in Statistics and computing. Springer, 2002. doi: 10.1007/978-0-387-21706-2[28].

[8] 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/

[9] Wikipedia contributors, "Quadric — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Quadric

[10] Wikipedia contributors, "Quadratic classifier — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Quadratic_classifier

[11] Wikipedia contributors, "Conditional independence — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Conditional_independence

[12] M. Hopkins, E. Reeber, G. Forman, and J. Suermondt, "Spambase." UCI Machine Learning Repository, 1999. doi: 10.24432/C53G6X[29].

[13] D. Barber, *Bayesian reasoning and machine learning*. Cambridge: Cambridge University Press, 2012. doi: 10.1017/CBO9780511804779[30].

[14] Wikipedia contributors, "Naive bayes classifier — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Naive_Bayes_classifier

[27] https://doi.org/10.1007/978-3-031-38747-0
[28] https://doi.org/10.1007/978-0-387-21706-2
[29] https://doi.org/10.24432/C53G6X
[30] https://doi.org/10.1017/CBO9780511804779