

Probability

Random variables, distributions, expectation, and prediction

Last modified: 2026-10-25

```
# The Python chunks on this page import numpy, pandas and scipy; reticulate provisions them.
library(reticulate)
py_require(c("numpy", "pandas", "scipy"))
```

i Source and scope

This appendix fixes the probabilistic tools used across classification, loss functions, and noise models. It points to the worked calculations, interactive widgets, and exercises in the lab's shared prerequisite site: *Probability for Data Science*^a.

The joint probability table, chain rule, continuous density intuition, and conditional independence examples are adapted from Brian Hutchinson's Fall 2024 DATA 471 probability refresher [1].

Everything here is standard. Goodfellow et al. [2], chapter 3, covers probability and information theory at length and is freely readable online: random variables, probability distributions, marginal and conditional distributions, expectation, variance, covariance, and Bayes' rule. Bishop [3] is a second account of the same material, with probability in section 1.2.

^a<https://morrison-lab.github.io/pds/>

Probability provides the mathematical framework for quantifying uncertainty, modeling noise in data, and deriving optimal predictions under loss functions. This appendix summarizes the key concepts and links directly to their full definitions and worked exercises in *Probability for Data Science*¹.

1 Probability foundations and distributions

*Probability for Data Science*² covers the probabilistic foundations these notes assume:

- Random variables³, probability mass functions⁴ (PMFs), probability density functions⁵ (PDFs), and cumulative distribution functions⁶ (CDFs).
- Joint distributions⁷ and marginalization⁸.
- Continuous distributions and densities⁹.
- Conditional probability¹⁰, the chain rule¹¹, total probability¹², and Bayes' theorem¹³.
- Interactive widget: Base Rate Fallacy Explorer¹⁴.

¹<https://morrison-lab.github.io/pds/>

²<https://morrison-lab.github.io/pds/>

³<https://morrison-lab.github.io/pds/random-variables.html#def-random-variable>

⁴<https://morrison-lab.github.io/pds/random-variables.html#def-pmf>

⁵<https://morrison-lab.github.io/pds/random-variables.html#def-pdf>

⁶<https://morrison-lab.github.io/pds/random-variables.html#def-cdf>

⁷<https://morrison-lab.github.io/pds/random-variables.html#def-joint-distribution>

⁸<https://morrison-lab.github.io/pds/random-variables.html#def-marginal>

⁹<https://morrison-lab.github.io/pds/random-variables.html#sec-continuous-distributions>

¹⁰<https://morrison-lab.github.io/pds/probability-basics.html#def-conditional-prob>

¹¹<https://morrison-lab.github.io/pds/probability-basics.html#thm-law-conditional-prob>

¹²<https://morrison-lab.github.io/pds/probability-basics.html#thm-total-prob>

¹³<https://morrison-lab.github.io/pds/probability-basics.html#thm-bayes>

¹⁴<https://morrison-lab.github.io/pds/probability-basics.html#fig-base-rate>

- Statistical independence¹⁵ and conditional independence¹⁶, illustrated with Hutchinson’s umbrella and rain example [1].
- The multivariate normal distribution¹⁷ and mixture models¹⁸. Unsupervised Learning uses both to build Gaussian mixture models and fit them via expectation–maximization.

Worked exercises in Probability for Data Science:

- Deriving Bayes’ rule from conditional probability¹⁹
- Medical diagnosis and base rates²⁰

i Watch

- Sanderson builds Bayes’ rule from a picture^a, and generalizes it to a formula^b [4].
- Avati’s probability review covers Bayes’ theorem, from 26:37^c [5], Lecture 3.
- Hutchinson’s Probability Refresher^d (27 min) covers probability, including Bayes’ rule [6].
- Starmer explains conditional probability^e [7] and Bayes’ theorem^f [8].

^a<https://www.youtube.com/watch?v=HZGCoVF3YvM>

^b<https://www.youtube.com/watch?v=HZGCoVF3YvM&t=249s>

^c<https://www.youtube.com/watch?v=M8wnYc1m04&t=1597s>

^dhttps://facultyweb.cs.wvu.edu/~hutchib2/video_lectures/data371/#probability_refresher

^ehttps://www.youtube.com/watch?v=_IgyaD7vOOA

^f<https://www.youtube.com/watch?v=9wCnvr7Xw4E>

i Read

GeeksforGeeks explains:

- conditional probability^a [9]
- Bayes’ theorem^b [10]

^a<https://www.geeksforgeeks.org/maths/conditional-probability/>

^b<https://www.geeksforgeeks.org/maths/bayes-theorem/>

i Watch

5MinuteAI’s Probability & Statistics^a playlist [11] follows the University of Washington’s CSE 312. Its five-minute videos cover:

- introduction to discrete probability^b
- conditional probability^c
- independence^d

^a<https://www.youtube.com/playlist?list=PLB45KifGiuHesi4PALNZSYZFhViVGQJK>

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

^c<https://www.youtube.com/watch?v=c-yB3q7zYIU>

^d<https://www.youtube.com/watch?v=9t7u6pfgw0k>

i Read

The University of Washington’s CSE 446 slides open with a probability review^a [12], Lecture 1.

^a<https://courses.cs.washington.edu/courses/cse446/26sp/schedule/lectures/01-intro,%20probability%20review.pdf>

¹⁵<https://morrison-lab.github.io/pds/independence.html#def-indpt>

¹⁶<https://morrison-lab.github.io/pds/independence.html#def-cind>

¹⁷<https://morrison-lab.github.io/pds/distributions.html#def-mvn>

¹⁸<https://morrison-lab.github.io/pds/distributions.html#def-mixture>

¹⁹<https://morrison-lab.github.io/pds/probability-basics.html#exr-bayes>

²⁰<https://morrison-lab.github.io/pds/probability-basics.html#exr-base-rate>

2 Factorizing a joint distribution

Theorem 2.1 (The chain rule of probability). *For random variables $X_1, \dots, X_N$, in any order,*

$$\Pr(X_1, \dots, X_N) = \prod_{i=1}^N \Pr(X_i \mid X_1, \dots, X_{i-1}).$$

Each factor conditions on all the variables that precede it in the chosen order [13].

i Proof

Proof. For $N = 2$, this is the law of conditional probability^a, $\Pr(X_1, X_2) = \Pr(X_1) \Pr(X_2 \mid X_1)$. Suppose it holds for $N - 1$ variables. Apply the $N = 2$ case to the pair $(X_1, \dots, X_{N-1})$ and X_N :

$$\Pr(X_1, \dots, X_N) = \Pr(X_1, \dots, X_{N-1}) \Pr(X_N \mid X_1, \dots, X_{N-1}).$$

Then expand the first factor by the induction hypothesis. □

^a<https://morrison-lab.github.io/pds/probability-basics.html#thm-law-conditional-prob>

Example 2.1 (Two factorizations of the same spam probabilities). In the Spambase data [14], let Y be the class (spam or not), X_1 be “the email contains the word *free*”, and X_2 be “the email contains the word *money*”. Take the proportions of the 4601 emails as probabilities. Compute $\Pr(X_1 = 1, X_2 = 1, Y = 1)$ directly, and as the product of Theorem 2.1 in two different orders:

2.0.1 Python

```
import pandas as pd

sb_cr = pd.read_csv("data/spambase.csv")
y_cr = sb_cr["Class"] == 1
x1_cr = sb_cr["word_freq_free"] > 0
x2_cr = sb_cr["word_freq_money"] > 0

def pr_cr(event):
    """The proportion of emails for which the event holds."""
    return event.mean()

direct_cr = pr_cr(x1_cr & x2_cr & y_cr)

# Order (Y, X1, X2): Pr(y) Pr(x1 | y) Pr(x2 | x1, y).
order_a_cr = pr_cr(y_cr) * (pr_cr(x1_cr & y_cr) / pr_cr(y_cr)) * (pr_cr(x1_cr & x2_cr & y_cr) / pr_cr(x1_cr & y_cr))
# Order (X2, Y, X1): Pr(x2) Pr(y | x2) Pr(x1 | y, x2).
order_b_cr = pr_cr(x2_cr) * (pr_cr(y_cr & x2_cr) / pr_cr(x2_cr)) * (pr_cr(x1_cr & y_cr & x2_cr) / pr_cr(y_cr & x2_cr))
print(direct_cr, order_a_cr, order_b_cr)

0.09497935231471419 0.09497935231471419 0.09497935231471419
```

2.0.2 R

```

sb_cr <- read.csv("data/spambase.csv")
y_cr <- sb_cr$Class == 1
x1_cr <- sb_cr$word_freq_free > 0
x2_cr <- sb_cr$word_freq_money > 0

# The proportion of emails for which the event holds.
pr_cr <- function(event) mean(event)

direct_cr <- pr_cr(x1_cr & x2_cr & y_cr)

# Order (Y, X1, X2): Pr(y) Pr(x1 | y) Pr(x2 | x1, y).
order_a_cr <- pr_cr(y_cr) * (pr_cr(x1_cr & y_cr) / pr_cr(y_cr)) * (pr_cr(x1_cr & x2_cr & y_cr) / pr_cr(x1_cr & y_cr))
# Order (X2, Y, X1): Pr(x2) Pr(y | x2) Pr(x1 | y, x2).
order_b_cr <- pr_cr(x2_cr) * (pr_cr(y_cr & x2_cr) / pr_cr(x2_cr)) * (pr_cr(x1_cr & y_cr & x2_cr) / pr_cr(y_cr & x2_cr))
c(direct = direct_cr, order_a = order_a_cr, order_b = order_b_cr)

      direct      order_a      order_b
0.09497935 0.09497935 0.09497935
All three equal 0.0950: about 9.5% of emails are spam that contains both words. The two orders give
different factors, such as  $\Pr(x_1 | y)$  in one and  $\Pr(y | x_2)$  in the other, but the same product.

```

For three variables, this says $\Pr(x, y, z) = \Pr(x) \Pr(y | x) \Pr(z | x, y)$. Reordering the variables gives a different factorization of the same joint distribution.

The chain rule is the backbone of autoregressive sequence models, which predict each element of a sequence from the elements before it, and of the directed graphical models that generalize them.

The chain rule is exact, and it has too many parameters to use as it stands: the last factor, $\Pr(X_N | X_1, \dots, X_{N-1})$, is a separate distribution for every combination of the earlier variables. A modeling assumption cuts that down. Conditional independence²¹ says that, once some variable is known, other variables carry no further information about each other.

Definition 2.1 (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 [15]:

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

A *naive Bayes classifier* combines this with Bayes' rule, $\Pr(Y | X_1, \dots, X_p) \propto \Pr(Y) \prod_{j=1}^p \Pr(X_j | Y)$ [16].

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

Example 2.2 (Do *free* and *money* appear independently in spam?). Continue with the Spambase events of the [chain rule example](#), restricted to the spam emails ($Y = 1$). Compare the joint probability $\Pr(X_1 = 1, X_2 = 1 | Y = 1)$ with the product $\Pr(X_1 = 1 | Y = 1) \Pr(X_2 = 1 | Y = 1)$ that the naive Bayes assumption would give:

2.0.3 Python

²¹<https://morrison-lab.github.io/pds/independence.html#def-cind>

```

spam_nb = sb_cr[y_cr] # 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

2.0.4 R

```

spam_nb <- sb_cr[y_cr, ] # 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 independence would predict 20.5% with both. The data show 24.1%. The assumption is wrong here, since the two words tend to appear together, which is why it is called naive. A naive Bayes classifier uses the product anyway, and often classifies well despite the error.

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

Example 2.3 (Counting parameters). Suppose Y takes K values and each of p features is binary. Without the naive Bayes assumption, $\Pr(X_1, \dots, X_p \mid Y = k)$ is an arbitrary distribution^a over 2^p outcomes, 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/pds/random-variables.html#def-cdf>

3 Expectation, variance, and covariance

Probability for Data Science covers moments, variance properties, and noise models:

- Expectation²² and linearity of expectation²³.
- Variance²⁴ and the simplified expression for variance²⁵: $\text{Var}(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$.
- Covariance²⁶ and correlation²⁷.
- The mean-zero noise model²⁸ in regression: $\mathbb{E}[\varepsilon] = 0$ and $\mathbb{E}[\varepsilon^2] = \sigma^2$.

Worked exercises in Probability for Data Science:

- Using linearity of expectation²⁹
- Second moment of mean-zero noise³⁰

i Watch

- Avati's probability review covers expectations, from 11:33^a, and expectations of two random variables, from 34:06^b [5], Lecture 3.
- Starmer explains expected values^c [17].

^a<https://www.youtube.com/watch?v=Mt8wnYc1m04&t=693s>

^b<https://www.youtube.com/watch?v=Mt8wnYc1m04&t=2046s>

^chttps://www.youtube.com/watch?v=KLs_7b7SKi4

i Read

GeeksforGeeks explains:

- expected value^a [18]
- variance^b [19]

^a<https://www.geeksforgeeks.org/maths/expected-value/>

^b<https://www.geeksforgeeks.org/maths/variance/>

i Watch

5MinuteAI explains expectation^a, variance^b and covariance and correlation^c [11].

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

^b<https://www.youtube.com/watch?v=5CT1DXPp2HY>

^c<https://www.youtube.com/watch?v=hiim8HII81Y>

4 The best constant prediction

Exercise 4.1 (Predicting with one number). A city must predict the crime rate Y of a randomly chosen neighbourhood, using no input at all: it predicts the same number c for every neighbourhood. Write $\mu \stackrel{\text{def}}{=} \mathbb{E}[Y]$.

Write the expected^a squared error of the prediction c , as a function of c .

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

²²<https://morrison-lab.github.io/pds/expectation.html#def-expectation>

²³<https://morrison-lab.github.io/pds/expectation.html#thm-linearity-expectation>

²⁴<https://morrison-lab.github.io/pds/variance-covariance.html#def-variance>

²⁵<https://morrison-lab.github.io/pds/variance-covariance.html#thm-variance>

²⁶<https://morrison-lab.github.io/pds/variance-covariance.html#def-cov>

²⁷<https://morrison-lab.github.io/pds/variance-covariance.html#def-correlation>

²⁸<https://morrison-lab.github.io/pds/variance-covariance.html#sec-noise-model>

²⁹<https://morrison-lab.github.io/pds/expectation.html#exr-linearity-use>

³⁰<https://morrison-lab.github.io/pds/variance-covariance.html#exr-noise-second-moment>

Solution

Solution 4.1. The squared error of the prediction is $(Y - c)^2$, so its expectation^a is $\mathbb{E}[(Y - c)^2]$.

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

Exercise 4.2 (Predicting with one number (part 2)). Use the setup of Exercise 4.1. Show that this expected^a squared error equals $\text{Var}(Y) + (\mu - c)^2$.

Hint

Inside the square, add and subtract μ .

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

Solution

Solution 4.2. Add and subtract μ inside the square:

$$\mathbb{E}[(Y - c)^2] = \mathbb{E}[((Y - \mu) + (\mu - c))^2].$$

Expand the square, using the square of a sum^a:

$$= \mathbb{E}[(Y - \mu)^2 + 2(Y - \mu)(\mu - c) + (\mu - c)^2].$$

Apply linearity of expectation^b. The numbers μ and c are constants^c, so $2(\mu - c)$ and $(\mu - c)^2$ come outside the expectation:

$$= \mathbb{E}[(Y - \mu)^2] + 2(\mu - c) \mathbb{E}[Y - \mu] + (\mu - c)^2.$$

The middle expectation is zero, because

$$\begin{aligned}\mathbb{E}[Y - \mu] &= \mathbb{E}[Y] - \mu \\ &= \mu - \mu \\ &= 0,\end{aligned}$$

so the risk reduces to:

$$= \mathbb{E}[(Y - \mu)^2] + (\mu - c)^2.$$

The first term is the variance of Y ^d:

$$= \text{Var}(Y) + (\mu - c)^2.$$

The probability notes state this result as a theorem^e.

^a<https://morrison-lab.github.io/mds/algebra-sums.html#thm-square-of-a-sum>

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

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

^d<https://morrison-lab.github.io/pds/variance-covariance.html#thm-variance-expanded>

^e<https://morrison-lab.github.io/pds/variance-covariance.html#thm-constant-prediction-decompose>

Example 4.1 (Predicting fuel efficiency with a constant). In the `Auto` data [20], treat the `mpg` values of the cars as the whole distribution of Y , each car equally likely. For the constants $c = 15, 20, 25$, compute the expected squared error $\mathbb{E}[(Y - c)^2]$ directly and as $\text{Var}(Y) + (\mathbb{E}[Y] - c)^2$:

4.0.1 Python

```
import numpy as np
import pandas as pd

y_cp = pd.read_csv("data/auto.csv")["mpg"].to_numpy(float)
mean_cp, var_cp = y_cp.mean(), y_cp.var() # the variance divides by n, as the distribution requires
for c in (15, 20, 25):
    direct = np.mean((y_cp - c) ** 2)
    decomposed = var_cp + (mean_cp - c) ** 2
    print(c, round(direct, 3), round(decomposed, 3))

15 132.096 132.096
20 72.637 72.637
25 63.178 63.178
```

4.0.2 R

```
y_cp <- read.csv("data/auto.csv")$mpg
mean_cp <- mean(y_cp)
var_cp <- mean((y_cp - mean_cp)^2) # the variance divides by n, as the distribution requires
cs_cp <- c(15, 20, 25)
data.frame(
  c = cs_cp,
  direct = sapply(cs_cp, function(c) mean((y_cp - c)^2)),
  decomposed = var_cp + (mean_cp - cs_cp)^2
)

# A tibble: 3 x 3
      c direct decomposed
  <dbl> <dbl>      <dbl>
1    15  132.      132.
2    20   72.6     72.6
3    25   63.2     63.2
```

The two columns agree for every c . The mean of `mpg` is 23.45 and its variance is 60.76, so the constant $c = 20$ has expected squared error equal to the variance plus 11.87 for being 3.45 away from the mean. No constant can have a smaller expected squared error than the variance, as the next exercise shows.

Exercise 4.3 (The best single number). Use the decomposition of the expected squared error of a constant prediction^a.

Which number c minimizes $\mathbb{E}[(Y - c)^2]$?

^a<https://morrison-lab.github.io/pds/variance-covariance.html#thm-constant-prediction-decompose>

Solution

Solution 4.3. In the decomposition of the expected squared error of a constant prediction^a, the first term, $\text{Var}(Y)$, does not depend on c . The second term, $(\mu - c)^2$, is a square, so it is never negative, and it is zero exactly when $c = \mu$. So

$$\begin{aligned} c &= \mu \\ &= \mathbb{E}[Y] \end{aligned}$$

minimizes the expected^b squared error, and no other number does. In the notation of the argmin^c, the set of minimizers has the single element μ .

^a<https://morrison-lab.github.io/pds/variance-covariance.html#thm-constant-prediction-decompose>

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

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

Exercise 4.4 (The best single number (part 2)). Use the setup of Exercise 4.3. What is the smallest expected^a squared error that any constant^b prediction can achieve?

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

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

Solution

Solution 4.4. At $c = \mu$, the second term is zero, so the smallest expected^a squared error is $\text{Var}(Y)$. Under squared error, the best prediction made without any input is the mean^b, and the error that remains is the variance^c. A model that uses an input x can only improve on $\text{Var}(Y)$ by learning how the mean of Y changes with x .

The probability notes state the result as a corollary^d.

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

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

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

^d<https://morrison-lab.github.io/pds/variance-covariance.html#cor-best-constant>

Example 4.2 (Searching for the best constant). For the mpg values of the Auto data [20], find the constant c that minimizes the average squared error by numerical search, without using the mean:

4.0.3 Python

```
from scipy.optimize import minimize_scalar

best_bc = minimize_scalar(lambda c: np.mean((y_cp - c) ** 2))
print(best_bc.x, best_bc.fun)

23.445918367347897 60.76273844231571

print(mean_cp, var_cp)

23.445918367346938 60.76273844231571
```

4.0.4 R

```
best_bc <- optimize(function(c) mean((y_cp - c)^2), interval = range(y_cp))
c(minimizer = best_bc$minimum, minimum = best_bc$objective, mean = mean_cp, variance = var_cp)

minimizer    minimum      mean  variance
 23.44592  60.76274  23.44592  60.76274
```

The search ends at 23.446, the mean of mpg, and the smallest error it finds is 60.763, the variance. This matches the best constant prediction^a: the best constant under squared error is the mean, and the best error is the variance.

^a<https://morrison-lab.github.io/pds/variance-covariance.html#cor-best-constant>

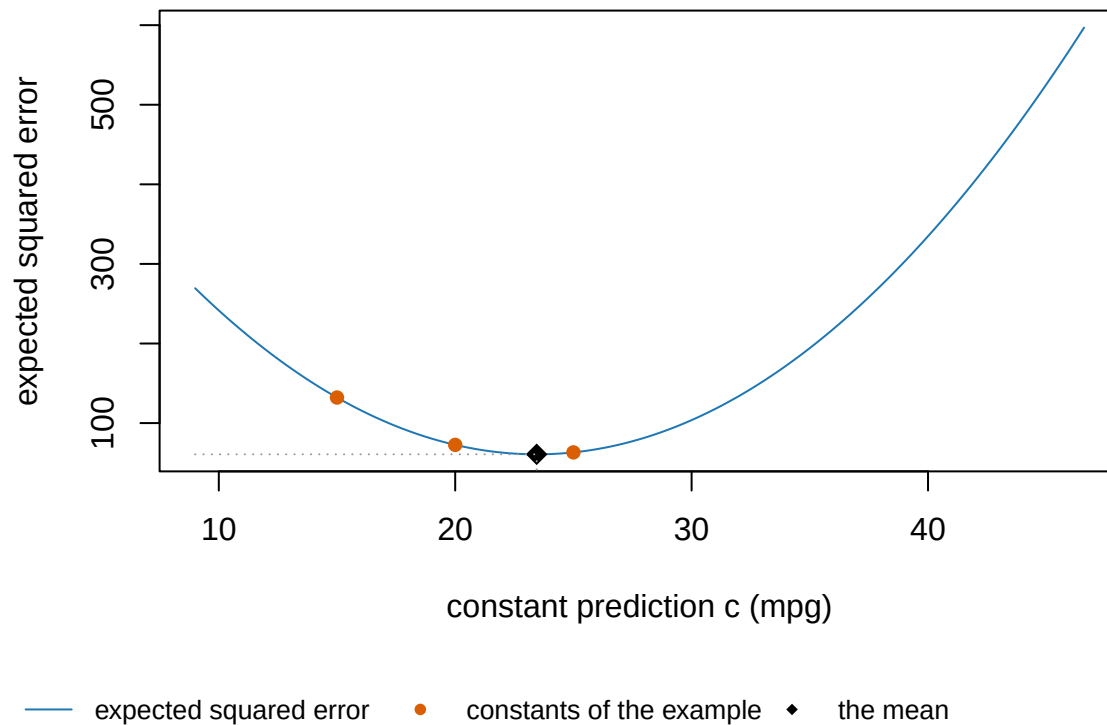


Figure 1: The expected squared error of a constant prediction c of `mpg`, for the `Auto` data of Example 4.2. The curve is a parabola, as the decomposition of the expected squared error³¹ says: its lowest point is at the mean of `mpg`, and its height there is the variance. The dots are the three constants of Example 4.1.

Exercise 4.5 (The best constant for the five neighbourhoods). Take the five neighbourhoods of A First Model, and treat them as the whole population, each with probability $1/5$. Then $\mathbb{E}[(Y - c)^2]$ is the empirical risk of the model $h(x) = c$, which ignores the income x . Use the best constant prediction^a to find the constant that minimizes this empirical risk, and the empirical risk it achieves. Compute both numbers.

^a<https://morrison-lab.github.io/pds/variance-covariance.html#cor-best-constant>

Solution

Solution 4.5. By the best constant prediction^a, the best constant is the mean crime rate, and its empirical risk is the variance of the five crime rates, computed with divisor 5 rather than 4, because the five neighbourhoods are the whole population here.

4.0.5 R

```
x <- c(0.49, 0.18, 0.31, 0.40, 0.24)
y <- c(0.09, 0.45, 0.23, 0.19, 0.48)

c_best <- mean(y)
risk_constant <- mean((y - c_best)^2)

risk_line <- mean(residuals(lm(y ~ x))^2)
print(round(c(c_best, risk_constant, risk_line), 6))

[1] 0.288000 0.023056 0.002721
```

4.0.6 Python

```
import numpy as np

x = np.array([0.49, 0.18, 0.31, 0.40, 0.24])
y = np.array([0.09, 0.45, 0.23, 0.19, 0.48])

c = np.mean(y)
risk_constant = np.mean((y - c) ** 2)

m, b = np.polyfit(x, y, 1)
risk_line = np.mean((m * x + b - y) ** 2)
print(round(c, 6), round(risk_constant, 6), round(risk_line, 6))
```

0.288 0.023056 0.002721

The best constant is $c = 0.2880$, with empirical risk 0.023056.

^a<https://morrison-lab.github.io/pds/variance-covariance.html#cor-best-constant>

Exercise 4.6 (The best constant for the five neighbourhoods (part 2)). Use the setup of Exercise 4.5. Compare that empirical risk with the empirical risk of the fitted line of A First Model.

Solution

Solution 4.6. The fitted line reaches an empirical risk of 0.002721, which is 88% lower.

Exercise 4.7 (The best constant for the five neighbourhoods (part 3)). Use the setup of Exercise 4.5. What does the comparison say about the income feature?

Solution

Solution 4.7. On these five training points, knowing the income removes 88% of the squared error. Both numbers are computed on the training points, so the comparison is not a promise about new neighbourhoods, for the reason A First Model gives.

5 Further reading

- Goodfellow et al. [2], Chapter 3, on probability and information theory
- Starmer [21], on basic probability and probability distributions

References

- [1] B. Hutchinson, “DATA 471/571: Machine learning.” Western Washington University, 2024.
- [2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, 2016. Available: <https://www.deeplearningbook.org/>
- [3] C. M. Bishop, *Pattern recognition and machine learning*. in Information science and statistics. New York: Springer, 2006.
- [4] G. Sanderson, “Bayes theorem, the geometry of changing beliefs.” 3Blue1Brown, video on YouTube, 2019. Available: <https://www.youtube.com/watch?v=HZGCoVF3YvM>
- [5] A. Avati, “CS229: Machine learning, summer 2019.” Stanford Online, lecture videos on YouTube, 2019. Available: https://www.youtube.com/playlist?list=PLoROMvovd4rNH7qL6-efu_q2_bPuy0adh

- [6] B. Hutchinson, “DATA 471/571 (machine learning) and CSCI 481/581 (deep learning) video lectures.” Western Washington University. Accessed: Sep. 28, 2026. [Online]. Available: https://facultyweb.cs.wvu.edu/~hutchib2/video_lectures/data371/
- [7] J. Starmer, “Conditional probabilities, clearly explained.” StatQuest, video on YouTube, 2021. Available: https://www.youtube.com/watch?v=_IgyaD7vOOA
- [8] J. Starmer, “Bayes’ theorem, clearly explained.” StatQuest, video on YouTube, 2021. Available: <https://www.youtube.com/watch?v=9wCnvr7Xw4E>
- [9] GeeksforGeeks, “Conditional probability.” GeeksforGeeks, tutorial article, 2026. Available: <https://www.geeksforgeeks.org/maths/conditional-probability/>
- [10] GeeksforGeeks, “Bayes’ theorem.” GeeksforGeeks, tutorial article, 2026. Available: <https://www.geeksforgeeks.org/maths/bayes-theorem/>
- [11] 5MinuteAI, “Probability & statistics.” Video playlist on YouTube. Accessed: Oct. 12, 2026. [Online]. Available: <https://www.youtube.com/playlist?list=PLeB45KifGiuHesi4PALNZSYZFhViVGQJK>
- [12] N. Jaques, “CSE 446/546: Machine learning, lecture slides.” University of Washington, 2026. Accessed: Oct. 12, 2026. [Online]. Available: <https://courses.cs.washington.edu/courses/cse446/26sp/schedule/lectures/>
- [13] Wikipedia contributors, “Chain rule (probability) — Wikipedia, the free encyclopedia.” 2026. Available: [https://en.wikipedia.org/wiki/Chain_rule_\(probability\)](https://en.wikipedia.org/wiki/Chain_rule_(probability))
- [14] M. Hopkins, E. Reeber, G. Forman, and J. Suermondt, “Spambase.” UCI Machine Learning Repository, 1999. doi: 10.24432/C53G6X³².
- [15] Wikipedia contributors, “Conditional independence — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Conditional_independence
- [16] Wikipedia contributors, “Naive bayes classifier — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Naive_Bayes_classifier
- [17] J. Starmer, “Expected values, main ideas.” StatQuest, video on YouTube, 2021. Available: https://www.youtube.com/watch?v=KLs_7b7SKi4
- [18] GeeksforGeeks, “Expected value.” GeeksforGeeks, tutorial article, 2026. Available: <https://www.geeksforgeeks.org/maths/expected-value/>
- [19] GeeksforGeeks, “Variance.” GeeksforGeeks, tutorial article, 2026. Available: <https://www.geeksforgeeks.org/maths/variance/>
- [20] 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/>
- [21] J. Starmer, *The StatQuest illustrated guide to statistics: With hands-on examples in Python and R*. 2026. Available: <https://statquest.org/statquest-store/>

³²<https://doi.org/10.24432/C53G6X>