

Kinds of Learning Problems and Features

What the output looks like, which models keep what, and how inputs become numbers

Last modified: 2026-10-27

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

i Source

This chapter was split from A First Model and Generalization, which keeps the first worked example and the rest of its topics. The two chapters share one set of figures and data.

1 Kinds of learning problems

1.1 Models as assumptions and constraints

Exercise 1.1 (What the data cannot decide). Many different functions pass through all five points of the crime data. Fit the straight line and the polynomial^a of degree^b 4 to the five neighbourhoods, and compare their training errors.

^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>

Solution

Solution 1.1.

1.1.1 R

```
# Fit each family by least squares to the five neighbourhoods of the crime data.
fits_r <- do.call(rbind, lapply(c(1, 4), function(d) {
  fit <- lm(y ~ poly(x, d, raw = TRUE), data = crime)
  data.frame(
    degree = d,
    training_mse = mean(residuals(fit)^2),
    pred_at_0.1 = predict(fit, data.frame(x = 0.1)),
    pred_at_0.9 = predict(fit, data.frame(x = 0.9))
  )
}))
print(fits_r, digits = 3, row.names = FALSE)
```

degree	training_mse	pred_at_0.1	pred_at_0.9
1	0.00272	0.576	-0.454
4	0.00000	-1.222	-95.851

1.1.2 Python

```
import numpy as np

# Fit each family by least squares to the five neighbourhoods of the crime data.
rows = []
for degree in (1, 4):
    coefs = np.polyfit(crime["x"], crime["y"], degree)
    train_mse = np.mean((np.polyval(coefs, crime["x"]) - crime["y"]) ** 2)
    rows.append([degree, train_mse, np.polyval(coefs, 0.1), np.polyval(coefs, 0.9)])

fits = pd.DataFrame(rows, columns=["degree", "training MSE", "prediction at 0.1", "prediction at 0.9"])
print(fits.round(3).to_string(index=False))
```

degree	training MSE	prediction at 0.1	prediction at 0.9
1	0.003	0.576	-0.454
4	0.000	-1.222	-95.851

The polynomial^a of degree^b 4 passes through all five points, so its training error is zero (to rounding), and the line's is 0.0027. On the training error alone, the polynomial wins.

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

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

Exercise 1.2 (What the data cannot decide (part 2)). Use the two fits of Exercise 1.1. Compare their predictions at incomes $x = 0.1$ and $x = 0.9$, both outside the incomes in the data. What does the comparison say about what a model has to supply besides the data?

Solution

Solution 1.2. Away from the data the two disagree sharply: at $x = 0.1$ the line predicts a crime rate of 0.58 and the polynomial^a predicts -1.22, which is impossible for a rate. At $x = 0.9$ both leave the possible range, but the line predicts -0.45 and the polynomial -95.9.

Five points do not say which of these functions to trust. The data fix the value of the function at five inputs^b and leave the rest open. A model closes the gap by supplying the rest: it restricts the functions we consider, so that the data can choose among the few that remain.

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

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

i Remark (A model is a set of assumptions, and each one constrains the learner)

Remark 1.1 (A model is a set of assumptions, and each one constrains the learner). Choosing a model family is an assumption about the true mean function, and the assumptions made in the course come in three kinds.

- **Functional form.** The family of the crime example assumes the true mean function has no curvature. It constrains training to the two numbers m and b . If the true mean function μ bends, the best member h^* of the family (best in the sense of the smallest squared gap to μ , averaged over the population^a of inputs^b) still differs from μ , and more data narrows the search to h^* without closing that gap.
- **Noise.** The noise has mean^c zero by definition of the true mean function (zero-mean-noise theorem). That fixes what the fitted model estimates^d, the mean of the outcome given the input^e, and it is why squared error is a sensible loss. The noise model adds the assumption that the variance^f of the noise is finite, which keeps the squared error well defined.
- **Independence^g.** Treating the neighbourhoods as independent^h draws from the population is what lets the error on a validation set of held-out neighbourhoods stand in for the error on a new

one. The error on the training set cannot stand in, whatever the dependence, because the fit has already seen those neighbourhoods.

The constraint is what makes learning possible, and it is also the cost: a conclusion drawn from a fitted model is only as good as the assumptions behind it. What the fitted model recovers from the data is h^* within the family, and the estimator^{*i*} is the training procedure that picks a member of the family from the data.

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

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

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

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

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

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

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

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

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

1.2 Supervised, unsupervised, and in between

Exercise 1.3 (What the training data contain). Course Overview sorted learning problems by what the training data contain. Name the three cases, and say which one the crime-rate data of A first model, end to end are.

Solution

Solution 1.3.

- Every point is a pair (x, y) : supervised learning.
- Every point is an x alone: unsupervised learning, whose goal is to find structure.
- A few points have labels and most do not: semi-supervised learning.

The crime-rate data are supervised: each neighbourhood comes with its crime rate, which is the y the model learns to predict. That is also the case the anatomy figure draws.

Exercise 1.4 (Which of these is unsupervised?).

1. Predicting a neighbourhood's crime rate from its per-capita income.
2. Grouping news articles so that each group is about one topic, without being told the topics in advance.
3. Deciding whether an email is spam, from a corpus of emails somebody has already marked.

Solution

Solution 1.4. The second. Nothing in its training data says which topic any article belongs to, so there is no y to compare a prediction against — which also means there is no obvious way to score the result, and that difficulty is characteristic of unsupervised learning rather than incidental to it.

The first and third both include the answer and are supervised: regression and classification respectively.

i Watch

Hutchinson covers machine learning and its main problem types, Machine Learning Overview^{*a*} (30 min) [1].

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

Watch

Starmer gives a gentle introduction to machine learning^a [2].

^ahttps://www.youtube.com/watch?v=Gv9_4yMHFhI

1.3 What the output looks like

Exercise 1.5 (Other kinds of output). In regression the output y is a quantity, and in classification it is one of a fixed set of categories. What other shapes could y take? For each one you think of, name a task that produces it.

Solution

Solution 1.5. A few possibilities:

- a sequence of labels, one per word of a sentence, such as each word's part of speech;
- a sequence of arbitrary length, such as captioning an image with words or transcribing speech to text;
- an ordering, such as a search engine's list of results;
- a structured object, such as the parse tree of a sentence.

The definitions that follow name the problem each one poses. The shape of y decides what kind of problem it is, and largely decides which algorithms are available.

Definition 1.1 (Sequence labelling). In **sequence labelling**, the input is a sequence and the output y is a sequence of labels of the same length, one label per input position [3].

Example 1.1 (Part-of-speech tagging). Tagging each word of a sentence with its part of speech is [sequence labelling](#).

Definition 1.2 (Sequence-to-sequence prediction (sequence-to-sequence decoding, seq2seq)). In **sequence-to-sequence prediction**, the output y is a sequence whose length is not fixed in advance and need not match the length of the input [4].
Other names: **sequence-to-sequence decoding**, **seq2seq**.

Example 1.2 (Captioning, transcribing and translating). Captioning an image, transcribing speech to text and translating a sentence are all [sequence-to-sequence prediction](#).

Definition 1.3 (Ranking). In **ranking**, the output y is an ordering of a set of candidates [5].

Example 1.3 (Ordering web pages for a search). A search engine orders web pages by relevance to a query, which is [ranking](#).

Definition 1.4 (Structured prediction). In **structured prediction**, the output y is an object with internal structure, such as a tree or a graph, whose parts constrain one another [6].

Example 1.4 (Parsing a sentence). Producing the parse tree of a sentence is [structured prediction](#).

Exercise 1.6 (Learning without an output). Clustering learns from inputs x alone, with no output y . What else could a model learn from inputs alone, and what task could it serve?

Solution

Solution 1.6. A model can learn how probable each input is: a probability distribution over the inputs. Such a model says which inputs are typical and which are improbable. The commonest use is to flag improbable inputs, such as an unusual credit card transaction or a sensor reading from a failing machine, for a person to inspect. These notes spend most of their time on regression and classification, and visit the other kinds of problem once the needed methods have been introduced.

Definition 1.5 (Density estimation). In **density estimation**, there is no output y : the model is a probability density or mass function $P(x)$ over the inputs [7].

Example 1.5 (A histogram density for horsepower). Ignore `mpg` and every other column of the `Auto` data [8]: the inputs are the horsepower of the 392 cars, and there is no output. Estimate a density $P(x)$ for horsepower with a histogram, using bins 20 horsepower wide from 40 to 240, scaled so that the total area is 1:

1.3.1 Python

```
import numpy as np
import pandas as pd

auto_de = pd.read_csv("data/auto.csv").dropna(subset=["horsepower"])
hp_de = auto_de["horsepower"].to_numpy(float)

# density=True divides each bin's count by n times the bin width.
edges_de = np.arange(40, 241, 20)
dens_de, _ = np.histogram(hp_de, bins=edges_de, density=True)
print(dens_de.round(4))

[0.0019 0.0124 0.0144 0.0078 0.0028 0.0057 0.0022 0.0014 0.0008 0.0006]

print(round(dens_de.sum() * 20, 3))

1.0

# The share of cars from 80 to 120 horsepower, with a normal-approximation 95% interval.
share_de = np.mean((hp_de >= 80) & (hp_de < 120))
share_se_de = np.sqrt(share_de * (1 - share_de) / len(hp_de))
print(round(share_de, 3), round(share_de - 1.96 * share_se_de, 3), round(share_de + 1.96 * share_se_de, 3))

0.444 0.395 0.493
```

1.3.2 R

```
auto_de <- read.csv("data/auto.csv")
auto_de <- na.omit(auto_de[, c("name", "horsepower", "origin")])

# right = FALSE makes each bin include its left edge, as the Python bins do.
edges_de <- seq(40, 240, 20)
hist_de <- hist(auto_de$horsepower, breaks = edges_de, right = FALSE, plot = FALSE)
dens_de <- hist_de$density
round(dens_de, 4)
```

```
[1] 0.0019 0.0124 0.0144 0.0078 0.0028 0.0057 0.0022 0.0014 0.0008 0.0006
```

```
round(sum(dens_de) * 20, 3)
```

```
[1] 1
```

```
# The share of cars from 80 to 120 horsepower, with a normal-approximation 95% interval.
```

```
share_de <- mean(auto_de$horsepower >= 80 & auto_de$horsepower < 120)
```

```
share_se_de <- sqrt(share_de * (1 - share_de) / nrow(auto_de))
```

```
round(c(share = share_de, lower = share_de - 1.96 * share_se_de, upper = share_de + 1.96 * share_se_de,
```

```
share lower upper
```

```
0.444 0.395 0.493
```

The model is the function that gives a car with horsepower in bin j the density `dens_de[j]`. The densest bin is 80 to 100 horsepower, at 0.0144, and the sparsest is 220 to 240, at 0.0006. Its area is 1.00, as a density's must be, and the area between 80 and 120 horsepower, 0.44, estimates the share of cars in that range, with a 95% confidence interval of 0.39 to 0.49 from the 392 cars. No label was used anywhere, so density estimation is unsupervised (Definition 1.5).

i Remark (Density estimation is unsupervised)

Remark 1.2 (Density estimation is unsupervised). [Density estimation](#) is unsupervised.

Definition 1.6 (Anomaly detection (novelty detection, outlier detection)). **Anomaly detection** flags inputs that are unlike the training inputs, such as those in regions where an estimated density $P(x)$ is low [9].

Other names: **novelty detection**, **outlier detection**.

Example 1.6 (Flagging the low-density cars). Use the histogram density of Example 1.5 and flag every car whose horsepower falls in a region where the estimated density is below 0.001:

1.3.3 Python

```
# Which bin each car falls in, and the density of that bin.
```

```
bin_ad = np.digitize(hp_de, edges_de) - 1
```

```
flag_ad = dens_de[bin_ad] < 0.001
```

```
flagged_ad = auto_de.loc[flag_ad, ["name", "horsepower", "origin"]]
```

```
print(flag_ad.sum(), flagged_ad["horsepower"].min())
```

```
11 200
```

```
print(flagged_ad.head(5).to_string(index=False))
```

	name	horsepower	origin
	chevrolet impala	220	1
	plymouth fury iii	215	1
	pontiac catalina	225	1
buick	estate wagon (sw)	225	1
	ford f250	215	1

1.3.4 R

```
# Which bin each car falls in, and the density of that bin.
bin_ad <- findInterval(auto_de$horsepower, edges_de)
flagged_ad <- auto_de[dens_de[bin_ad] < 0.001, ]

c(flagged = nrow(flagged_ad), lowest_hp = min(flagged_ad$horsepower))

  flagged lowest_hp
      11         200

head(flagged_ad, 5)

# A tibble: 5 x 3
  name                horsepower origin
<chr>                <int>   <int>
1 chevrolet impala      220       1
2 plymouth fury iii     215       1
3 pontiac catalina      225       1
4 buick estate wagon (sw) 225       1
5 ford f250             215       1

import matplotlib.pyplot as plt

# Draw each bin as a bar; the flagged bins are those below the threshold.
low_ad = dens_de < 0.001
fig, ax = plt.subplots(figsize=(6, 3), layout="constrained")
ax.bar(edges_de[:-1], dens_de, width=20, align="edge", edgecolor="white",
       color=np.where(low_ad, "firebrick", "steelblue"));
ax.axhline(0.001, linestyle="--", color="black");
ax.set_xlabel("horsepower");
ax.set_ylabel("estimated density");
plt.show()
```

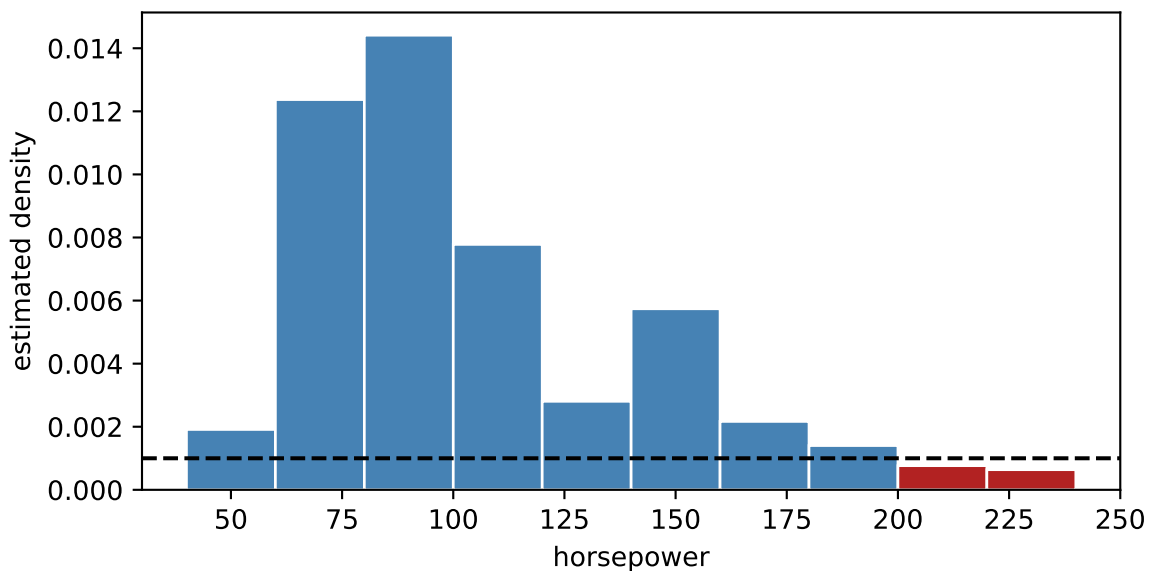


Figure 1: The histogram density of horsepower for the 392 Auto cars. Bins whose density falls below the dashed line at 0.001 are shaded red; the cars in them are flagged.

Figure 1 shows the density and the threshold. This flags 11 of the 392 cars, all with at least 200 horsepower. Among them, 11 have `origin` 1 (American), such as the chevrolet impala and the plymouth fury iii. Nobody labeled these cars as unusual. They are flagged only because few training cars look like

them, which is anomaly detection (Definition 1.6). Whether they are errors or just large-engined cars is a question the density cannot answer, and the threshold of 0.001 was a choice. It flags a bin only if it holds under 2% of the cars, since the bins are 20 horsepower wide.

Example 1.7 (A purchase unlike a customer's others). A bank's model of its customers' usual card transactions can flag a purchase whose amount and location look like none of them, without anyone having labeled that purchase as fraud.

1.4 Parametric and non-parametric models

Exercise 1.7 (What does a model keep?). A new neighbourhood reports its per-capita income, and the city wants a predicted crime rate. To predict with the fitted line of the crime example, what does the city need to have kept from training?

Solution

Solution 1.7. The line needs only its two parameters, $\hat{m}$ and $\hat{b}$. After training, the five data points can be thrown away.

Exercise 1.8 (What does a model keep? (part 2)). Use the setup of Exercise 1.7. A second rule predicts by averaging the crime rates of the k training neighbourhoods whose incomes are nearest the new one. What does the city need to have kept to use this rule?

Solution

Solution 1.8. The averaging rule needs every training pair $(x_{(i)}, y_{(i)})$, because any of them might be among the k nearest to the next input^a. It stores $2N$ numbers.

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

Exercise 1.9 (What does a model keep? (part 3)). Use the setup of Exercise 1.7. How does each answer change as the number of training points n grows?

Solution

Solution 1.9. In Exercise 1.9, the line keeps two numbers however large n gets. The averaging rule keeps $2N$ numbers, so its storage grows with the data.

The two rules also spend their computing time at different stages. The line does its work at training time, by solving the empirical-risk minimization, and then each prediction costs one multiplication and one addition. The averaging rule does nothing at training time, and each prediction searches all n stored points. This is the train-time-versus-run-time distinction of The anatomy of a learning system.

The averaging rule also becomes more flexible as n grows. With few points its predictions follow a coarse pattern; with many, its predictions can follow a curved relationship that no straight line can fit.

Definition 1.7 (Non-parametric model). A **non-parametric model** is a model family whose description grows with the training set, so that it is not a parametric model [10].

Example 1.8 (Nearest neighbors keep every car). Predict the mpg of a car with 100 horsepower by averaging the mpg of the five cars in the training set whose horsepower is closest to 100. This is the

averaging rule of Exercise 1.7. Train it on the first 20, the first 100 and then all 392 cars of the Auto data [8], and count the numbers it must keep to make a prediction:

1.4.1 Python

```
import numpy as np
import pandas as pd

auto_np = pd.read_csv("data/auto.csv").dropna(subset=["horsepower"])
hp_np = auto_np["horsepower"].to_numpy(float)
mpg_np = auto_np["mpg"].to_numpy(float)

for n_np in (20, 100, 392):
    # The prediction needs every stored pair, to find the five nearest in horsepower.
    nearest_np = np.argsort(np.abs(hp_np[:n_np] - 100), kind="stable")[:5]
    print(n_np, 2 * n_np, round(mpg_np[:n_np][nearest_np].mean(), 2))
```

20 40 22.4
100 200 17.8
392 784 17.8

1.4.2 R

```
auto_np <- na.omit(read.csv("data/auto.csv")[, c("mpg", "horsepower")])

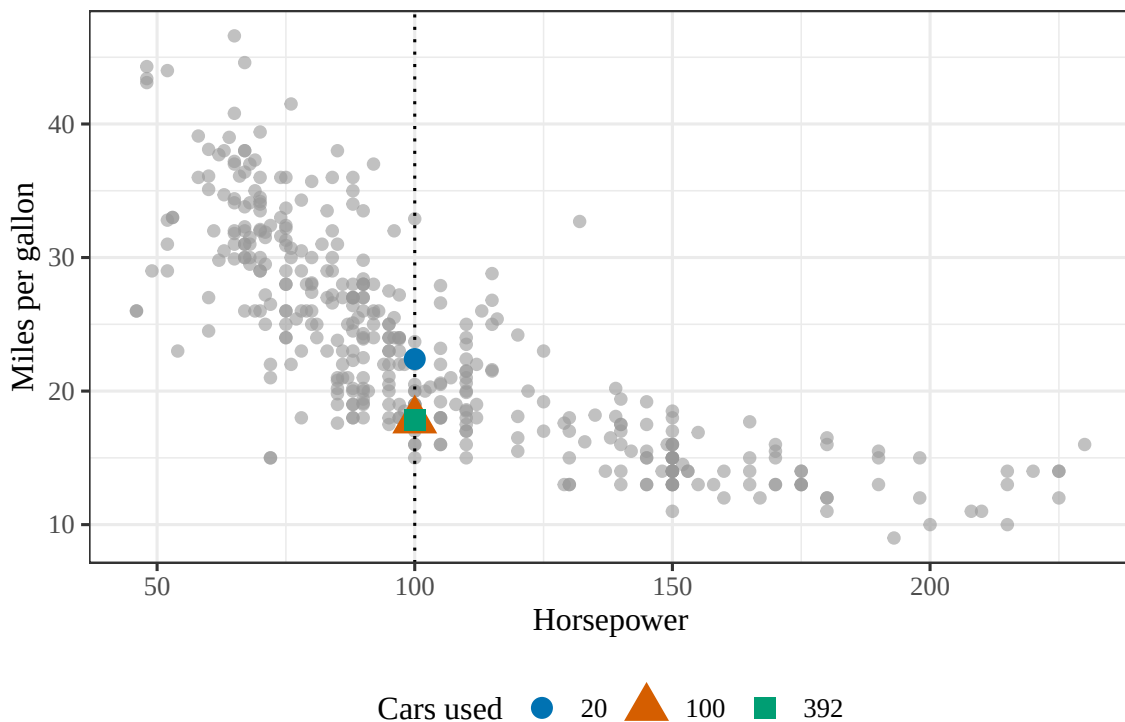
knn_np <- t(sapply(c(20, 100, 392), function(n) {
  train <- auto_np[1:n, ]
  # The prediction needs every stored pair, to find the five nearest.
  nearest <- order(abs(train$horsepower - 100))[1:5]
  c(cars = n, stored = 2 * n, prediction = mean(train$mpg[nearest]))
}))
round(knn_np, 2)
```

	cars	stored	prediction
[1,]	20	40	22.4
[2,]	100	200	17.8
[3,]	392	784	17.8

The line of the parametric-model example keeps two numbers at every size. This rule keeps 40 numbers after 20 cars, 200 after 100 and 784 after all 392, two for each car, because any of them might be among the five nearest to the next input. Its description grows with the training set, so it is non-parametric (Definition 1.7). Its prediction at 100 horsepower also moves with the data, from 22.4 to 17.8 miles per gallon.

1.4.3 R

```
library(ggplot2)
npf_pred <- data.frame(
  horsepower = 100,
  mpg = knn_np[, "prediction"],
  cars = factor(knn_np[, "cars"])
)
ggplot(auto_np, aes(horsepower, mpg)) +
  geom_point(colour = "grey60", alpha = 0.6) +
  geom_vline(xintercept = 100, linetype = "dotted") +
  geom_point(data = npf_pred, aes(colour = cars, shape = cars, size = cars)) +
  scale_size_manual(values = c("20" = 3.5, "100" = 5.5, "392" = 3.5)) +
  scale_colour_manual(values = c("20" = "#0072B2", "100" = "#D55E00", "392" = "#009E73")) +
  scale_shape_manual(values = c("20" = 16, "100" = 17, "392" = 15)) +
  labs(
    x = "Horsepower", y = "Miles per gallon",
    colour = "Cars used", shape = "Cars used", size = "Cars used"
  )
)
```



(a)

1.4.4 Python

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

fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.plot(hp_np, mpg_np, color="grey", marker="o", linestyle="", alpha=0.6, markersize=4);
ax.axvline(100, color="black", linestyle=":");
for n_npf, colour, marker, size in [
    (20, "#0072B2", "o", 8), (100, "#D55E00", "^", 12), (392, "#009E73", "s", 8)
]:
    nearest_npf = np.argsort(np.abs(hp_np[:n_npf] - 100), kind="stable")[:5]
    ax.plot(100, mpg_np[:n_npf][nearest_npf].mean(), color=colour, marker=marker,
            markersize=size, linestyle="", label=f"{n_npf} cars");
ax.set_xlabel("Horsepower");
ax.set_ylabel("Miles per gallon");
```

i Remark (Nearest neighbours and support vectors keep training points)

Remark 1.3 (Nearest neighbours and support vectors keep training points). The averaging rule of Exercise 1.7 is **non-parametric**: it keeps every training point. k -nearest neighbours is the general form of that rule. A support vector machine keeps a subset of the training points.

The course website’s version of this page has an interactive plot here.

i Read

GeeksforGeeks explains parametric and non-parametric methods^a [11].

^a<https://www.geeksforgeeks.org/machine-learning/difference-between-parametric-and-non-parametric-methods/>

1.5 Probabilistic and non-probabilistic models

Exercise 1.10 (What does a probability add?). Two spam filters read the same incoming message x .

- Filter A outputs one label, “spam” or “not spam”.
- Filter B outputs a number between 0 and 1: its estimate of the probability that the message is spam.

Name two actions the mail system can take with Filter B’s output that it cannot take with Filter A’s.

Solution

Solution 1.10. Filter B’s number says how sure the filter is, and the mail system can act on that confidence. Any two of these actions answer the question:

- rank^a messages from most to least likely to be spam;
- leave a message in the inbox, or send it to a person to check, when the probability is close to one half;
- choose a cutoff other than one half, such as quarantining a message only when the probability exceeds 0.9, because losing a real message costs more than reading one spam message.

Filter A’s label carries no confidence, so it supports none of these actions.

^a<https://morrison-lab.github.io/mds/linear-algebra-subspaces.html#def-rank>

Exercise 1.11 (What does a probability add? (part 2)). Use the setup of Exercise 1.10.

Can the mail system recover Filter A’s kind of output from Filter B’s?

Solution

Solution 1.11. Comparing Filter B’s number with a cutoff produces a label, so Filter A’s kind of output can be recovered from Filter B’s.

Exercise 1.12 (What does a probability add? (part 3)). Use the setup of Exercise 1.10.

Can it recover Filter B’s kind of output from Filter A’s?

Solution

Solution 1.12. Recovering Filter B’s kind of output from Filter A’s fails: a label does not say how close the message came to the other label.

Definition 1.8 (Probabilistic model). A **probabilistic model** is a model whose output is a probability distribution. In supervised learning, it produces the conditional distribution $P(y | x)$ of the output given the input. In unsupervised learning, it produces the distribution $P(x)$ of the input.

Example 1.9 (Filter B is probabilistic). Filter B of Exercise 1.10 is a **probabilistic model**. Build one from the 4,601 emails of the spam data [12]: sort each message into a group by whether it contains the word “george” at all and by how often it uses the word “you” (never, up to 2% of its words, or more), and let Filter B output the share of spam among the first half of the data (half A) that fall in the message’s group. Each share comes with a 95% Wilson interval, which stays inside $[0, 1]$ even when a group has no spam.

1.5.1 Python

```
import numpy as np
import pandas as pd

spam_pm = pd.read_csv("data/spam-george-you.csv")

# Group each message by "george" (present or not) and by how much it uses "you".
spam_pm["george_group"] = np.where(spam_pm["george"] > 0, "george", "no george")
spam_pm["you_group"] = np.where(
    spam_pm["you"] == 0, "0", np.where(spam_pm["you"] <= 2, "0 to 2", "over 2")
)
spam_pm["key"] = spam_pm["george_group"] + ", you " + spam_pm["you_group"]

# Filter B learns from half A only: the share of spam in each group.
train_pm = spam_pm[spam_pm["half"] == "A"]
tab_pm = train_pm.groupby("key").agg(n=("spam", "size"), p=("spam", "mean")).reset_index()

# 95% Wilson interval for each share, from its size n and share p.
z_pm = 1.96
scale_pm = 1 + z_pm**2 / tab_pm["n"]
centre_pm = (tab_pm["p"] + z_pm**2 / (2 * tab_pm["n"])) / scale_pm
half_pm = z_pm * np.sqrt(
    tab_pm["p"] * (1 - tab_pm["p"]) / tab_pm["n"] + z_pm**2 / (4 * tab_pm["n"] ** 2)
) / scale_pm
tab_pm["lower"] = centre_pm - half_pm
tab_pm["upper"] = centre_pm + half_pm
print(tab_pm.round(3))
```

	key	n	p	lower	upper
0	george, you 0	164	0.000	0.000	0.023
1	george, you 0 to 2	160	0.006	0.001	0.035
2	george, you over 2	72	0.014	0.002	0.075
3	no george, you 0	537	0.186	0.156	0.221
4	no george, you 0 to 2	612	0.490	0.451	0.530
5	no george, you over 2	756	0.656	0.622	0.689

1.5.2 R

```

spam_pm <- read.csv("data/spam-george-you.csv")

# Group each message by "george" (present or not) and by how much it uses "you".
george_group <- ifelse(spam_pm$george > 0, "george", "no george")
you_group <- ifelse(spam_pm$you == 0, "0", ifelse(spam_pm$you <= 2, "0 to 2", "over 2"))
spam_pm$key <- paste0(george_group, ", you ", you_group)

# Filter B learns from half A only: the share of spam in each group.
train_pm <- spam_pm[spam_pm$half == "A", ]
tab_pm <- data.frame(
  key = sort(unique(train_pm$key)),
  n = as.vector(tapply(train_pm$spam, train_pm$key, length)),
  p = as.vector(tapply(train_pm$spam, train_pm$key, mean))
)

# 95% Wilson interval for each share, from its size n and share p.
z_pm <- 1.96
scale_pm <- 1 + z_pm^2 / tab_pm$n
centre_pm <- (tab_pm$p + z_pm^2 / (2 * tab_pm$n)) / scale_pm
half_pm <- z_pm * sqrt(tab_pm$p * (1 - tab_pm$p) / tab_pm$n + z_pm^2 / (4 * tab_pm$n^2)) / scale_pm
tab_pm$lower <- centre_pm - half_pm
tab_pm$upper <- centre_pm + half_pm
print(round(tab_pm[, -1], 3))

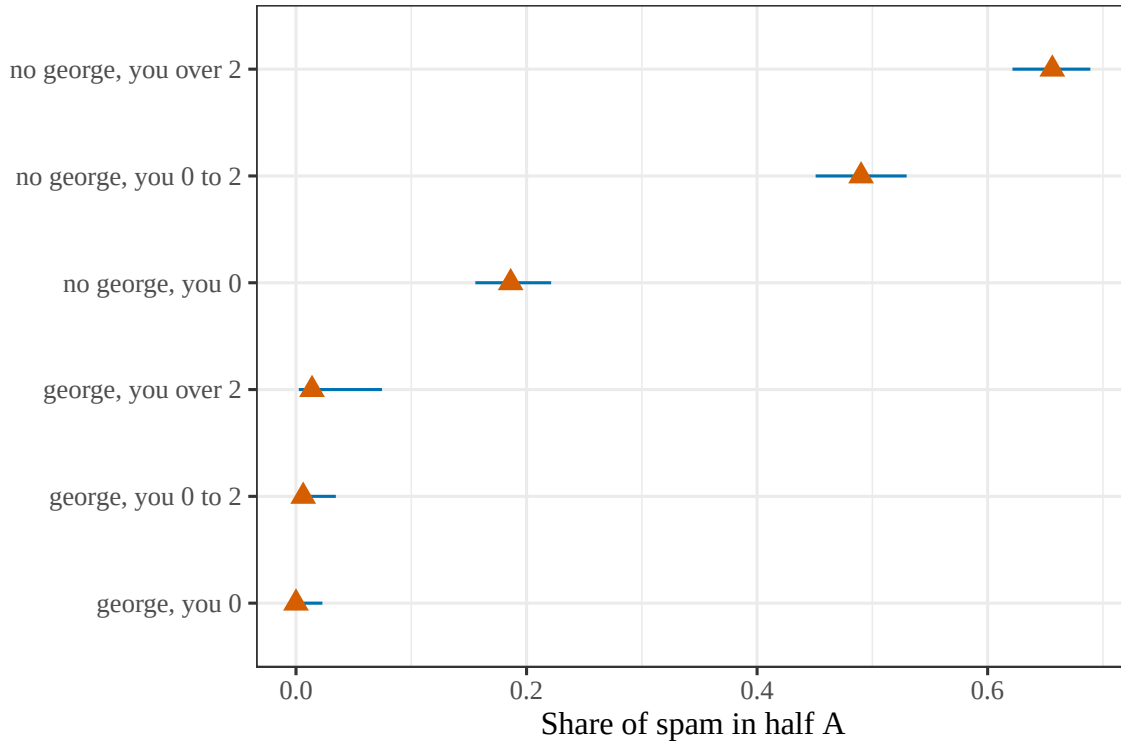
```

	n	p	lower	upper
1	164	0.000	0.000	0.023
2	160	0.006	0.001	0.035
3	72	0.014	0.002	0.075
4	537	0.186	0.156	0.221
5	612	0.490	0.451	0.530
6	756	0.656	0.622	0.689

A message with no “george” and “you” up to 2% of its words gets the output 0.49 (95% interval 0.45 to 0.53). One with no “george” and “you” above 2% gets 0.66 (95% interval 0.62 to 0.69). A message containing “george” that never says “you” gets 0.00 (95% interval 0.00 to 0.02). Filter B says how sure it is, and the mail system can rank, quarantine or send for review by that number.

1.5.3 R

```
library(ggplot2)
ggplot(tab_pm, aes(p, reorder(key, p))) +
  geom_linerange(aes(xmin = lower, xmax = upper), colour = "#0072B2") +
  geom_point(colour = "#D55E00", shape = 17, size = 3) +
  labs(x = "Share of spam in half A", y = NULL)
```

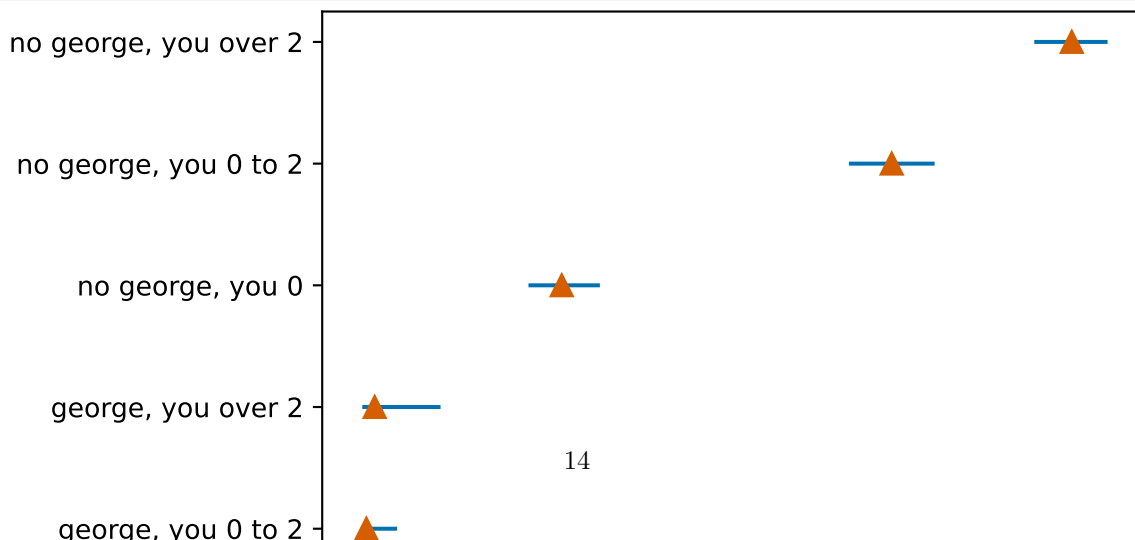


(a)

1.5.4 Python

```
import matplotlib.pyplot as plt

order_pmf = tab_pm.sort_values("p").reset_index(drop=True)
fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
ax.hlines(order_pmf.index, order_pmf["lower"], order_pmf["upper"], color="#0072B2");
ax.plot(order_pmf["p"], order_pmf.index, color="#D55E00", marker="^", markersize=8, linestyle="");
ax.set_yticks(order_pmf.index, order_pmf["key"]);
ax.set_xlabel("Share of spam in half A");
plt.show()
```



Definition 1.9 (Non-probabilistic model). A **non-probabilistic model** is a model whose output is a single label, value or cluster assignment, with no probability distribution attached.

Example 1.10 (Filter A is non-probabilistic). Filter A of Exercise 1.10 is a **non-probabilistic model**. Take the groups and shares `tab_pm` of Example 1.9 and keep only a label: Filter A says “spam” for a group whose training share of spam exceeds one half, and “not spam” otherwise. Score it on the second half of the data (half B), which Filter A never saw:

1.5.5 Python

```
# Filter A's rule, a label for each group, from Filter B's shares.
label_pm = tab_pm.set_index("key")["p"] > 0.5

# Half B: look up each message's group label and compare it with the truth.
test_pm = spam_pm[spam_pm["half"] == "B"]
pred_pm = label_pm.loc[test_pm["key"]].to_numpy()
correct_pm = pred_pm == (test_pm["spam"].to_numpy() == 1)
acc_pm = correct_pm.mean()
n_test_pm = len(correct_pm)
se_pm = np.sqrt(acc_pm * (1 - acc_pm) / n_test_pm)
print(label_pm)

key
george, you 0          False
george, you 0 to 2      False
george, you over 2      False
no george, you 0        False
no george, you 0 to 2    False
no george, you over 2    True
Name: p, dtype: bool

print(round(acc_pm, 3), round(acc_pm - 1.96 * se_pm, 3), round(acc_pm + 1.96 * se_pm, 3))

0.695 0.676 0.714
```

1.5.6 R

```
# Filter A's rule, a label for each group, from Filter B's shares.
label_pm <- setNames(tab_pm$p > 0.5, tab_pm$key)

# Half B: look up each message's group label and compare it with the truth.
test_pm <- spam_pm[spam_pm$half == "B", ]
pred_pm <- label_pm[test_pm$key]
correct_pm <- pred_pm == (test_pm$spam == 1)
acc_pm <- mean(correct_pm)
n_test_pm <- length(correct_pm)
se_pm <- sqrt(acc_pm * (1 - acc_pm) / n_test_pm)
print(label_pm)

      george, you 0      george, you 0 to 2      george, you over 2
      FALSE          FALSE          FALSE
no george, you 0 no george, you 0 to 2 no george, you over 2
      FALSE          FALSE          TRUE

print(round(c(acc_pm, acc_pm - 1.96 * se_pm, acc_pm + 1.96 * se_pm), 3))

[1] 0.695 0.676 0.714
```

Filter A is right on 69.5% of the 2300 messages in half B (95% interval 67.6% to 71.4%, a normal

approximation). Its labels hide how close some calls were: “no george, you 0 to 2” is labelled “not spam” from a share of 0.49, almost a coin flip, and “no george, you over 2” is labelled “spam” from 0.66. The two labels differ, but a label cannot say that the first call was nearly a toss-up and the second was not.

Exercise 1.13 (Joint or conditional?). A probabilistic spam filter (Definition 1.8) can be built in two ways. One version estimates^a the conditional distribution^b $P(y | x)$ of the label given the message. The other estimates the joint distribution^c $P(x, y)$ of messages and labels. Given the joint distribution $P(x, y)$, write the conditional distribution $P(y | x)$ in terms of it.

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

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

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

Solution

Solution 1.13. By the definition of conditional probability^a,

$$P(y | x) = \frac{P(x, y)}{P(x)},$$

and the denominator comes from the joint distribution^b by summing over the labels:

$$P(x) = \sum_y P(x, y).$$

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

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

Exercise 1.14 (Joint or conditional? (part 2)). Use the setup of Exercise 1.13. Given only the conditional distribution^a $P(y | x)$, can you compute $P(x, y)$?

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

Solution

Solution 1.14. No. The two are related by

$$P(x, y) = P(y | x) P(x),$$

Exercise 1.15 (Joint or conditional? (part 3)). Use the setup of Exercise 1.13. If not, what else would you need to compute $P(x, y)$?

Solution

Solution 1.15. Computing the joint distribution^a also requires $P(x)$, the distribution^b of the messages themselves. The conditional version never estimates^c $P(x)$.

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

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

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

Exercise 1.16 (Joint or conditional? (part 4)). Use the setup of Exercise 1.13. Which of the two versions could generate a new, realistic message x ?

Solution

Solution 1.16. Only the joint^a version. It describes how messages arise, so it can draw a message x from $P(x)$, or a message and its label together from $P(x, y)$. The conditional version only scores messages that it is given.

So the joint version can answer every question the conditional version can answer, and it can also generate data; in return, it has to estimate^b the harder object.

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

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

Definition 1.10 (Discriminative model). A **discriminative model** is a probabilistic model (Definition 1.8) that estimates the conditional distribution $P(y | x)$ of the output given the input directly, without estimating the distribution of the input [13].

i Remark (A discriminative spam filter cannot generate email)

Remark 1.4 (A discriminative spam filter cannot generate email). For spam, a **discriminative model** takes an email's word frequencies x and returns the probability that it is spam, but it cannot generate a new email, because it never learned which word frequencies are common.

Definition 1.11 (Generative model). A **generative model** is a probabilistic model (Definition 1.8) that estimates the distribution of the inputs: the joint distribution $P(x, y)$ in supervised learning, or the input distribution $P(x)$ in unsupervised learning [14].

Example 1.11 (Generating a spam email). A **generative model** can produce new data by sampling from the distribution it estimates. For spam, a generative model of $P(x, y)$ could generate the word frequencies of a new spam email by first drawing $y = \text{spam}$ and then drawing x from $P(x | y = \text{spam})$.

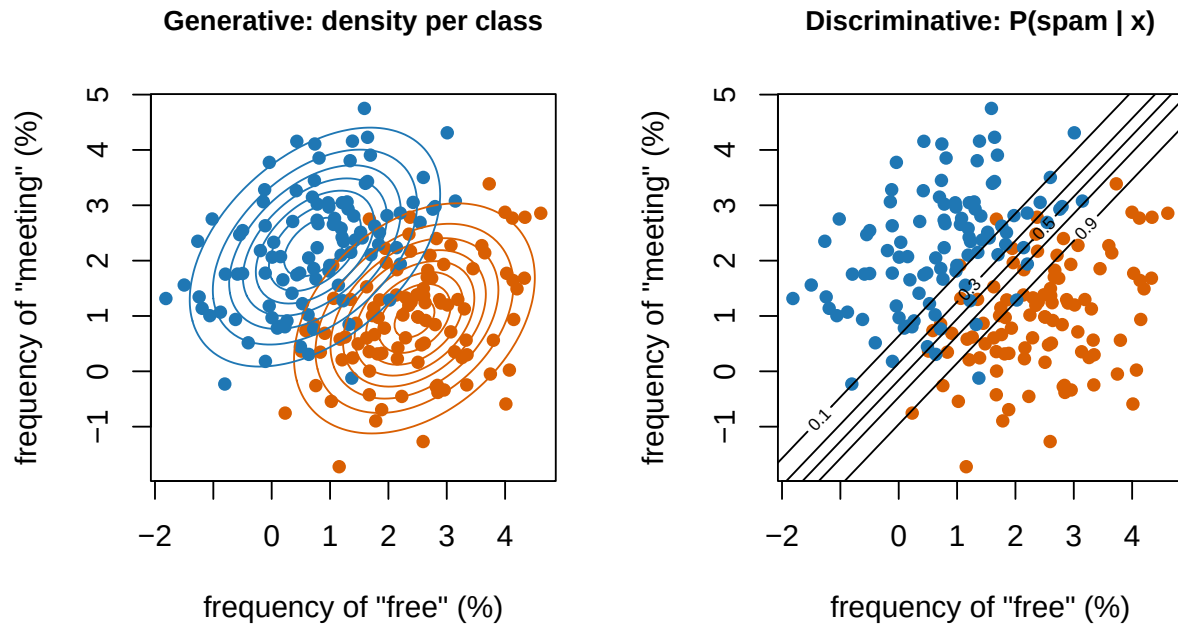


Figure 4: The same 200 emails, described by the frequencies of two words, under the two kinds of model. On the left, a generative model estimates how the word frequencies are distributed within each class, drawn as density contours, and can therefore generate new emails. On the right, a discriminative model estimates only the probability that an email is spam given its frequencies, drawn as contours of that probability, and says nothing about how common any frequencies are.

i Read

GeeksforGeeks explains generative and discriminative algorithms^a [15].

^a<https://www.geeksforgeeks.org/machine-learning/the-difference-between-generative-and-discriminative-machine-learning-algorithms/>

Exercise 1.17 (Four more learning tasks). For each of these four tasks, name the training data, the training algorithm, the model h , the input x and the output y , and place it in the categories of Section 1:

- spam filtering;
- machine translation;
- customer segmentation;
- movie recommendation.

Solution

Solution 1.17.

- **Spam filtering.** Training data: email messages, each with a label saying whether it is spam. Training algorithm: gradient descent for logistic regression. h is a logistic regression model, $x \in \mathbb{R}^p$ is a vector of word counts for one message, and $y \in \{0, 1\}$ is the predicted label. Supervised, classification, parametric.
- **Machine translation.** Training data: pairs^a of sentences, one in each language. Training algorithm: backpropagation through a neural network. h is an encoder–decoder model, x is a sentence in the source language, and y is its translation into the other language. Supervised, and the output is a *sequence* rather than one label, which is a kind this chapter has not listed.

- **Customer segmentation.** Training data: one vector of purchase counts per customer, with no labels at all. Training algorithm: K-means clustering. h is a clustering function, x is a customer's purchase vector, and y is a segment identifier. Unsupervised: nothing here says which segmentation is correct.
- **Movie recommendation.** Training data: the ratings that users gave to the movies they watched. Training algorithm: gradient descent on a low-rank factorization of the ratings table. h is the pair of factor matrices, one row per user and one per movie, x is a (user, movie) pair, and y is the predicted rating. Supervised on the ratings that were observed, with a numeric output, so regression; the pairs without ratings are exactly where the model is used.

^a<https://morrison-lab.github.io/mds/sets.html#def-ordered-pair>

2 Features

Exercise 2.1 (The crime example's feature vector). In the crime example of A first model, end to end, what is the feature vector x , and what is its dimension p ?

Solution

Solution 2.1. The feature vector holds one feature, the neighbourhood's per-capita income, so $x \in \mathbb{R}^1$ and $p = 1$. That is the smallest a feature vector can be, and it let us draw the whole data set in one scatter plot.

Exercise 2.2 (What flexibility cannot recover). Suppose two neighbourhoods in the crime example of A first model, end to end have the same per-capita income but different crime rates. Can any model that sees only income predict both crime rates correctly, however flexible it is?

Solution

Solution 2.2. No. A model that sees only income is a function of income, so two inputs with the same income get the same prediction. At least one of the two neighbourhoods is predicted wrongly, and the most any model can do is predict a value between them. A more flexible model changes how predictions vary with income, not what it can tell apart at a single income.

Exercise 2.3 (What flexibility cannot recover (part 2)). Use the setup of Exercise 2.2. What does that answer say about when the most important decision in building such a model gets made?

Solution

Solution 2.3. So a model can only be as good as the features it is shown, and no amount of flexibility recovers information that was never measured. The choice of what to measure is made before any algorithm^a runs, and it is one of the most important decisions in machine learning. So the crime example flagged per-capita income as a choice rather than a given. The rest of these notes mostly take the features as fixed and ask what to do with them. That is a division of the work, not a claim that the choice is unimportant.

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

Exercise 2.4 (Features of a delivery). A restaurant wants to predict how many minutes a food delivery will take. The output is the delivery time. The inputs^a are deliveries. List five features of a delivery

that the restaurant could record. Mark each one as a number or a category.

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

Solution

Solution 2.4. Distance to the customer (a number), number of items (a number), time of day (a number, or a category such as “lunch” and “dinner”), weather (a category), and whether the customer lives in an apartment (a category). Many other answers work.

Some of these are numbers already and some are categories. The section on [categorical features](#) is about what to do with the second kind.

2.1 Categorical features

Exercise 2.5 (Turning a category into numbers). A feature vector is a vector of numbers, but some features are categorical^a: a car’s region of origin, a colour, a film title.

How would you turn a categorical feature with two categories into a number?

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

Solution

Solution 2.5. One number suffices. Code one category as 0 and the other as 1, or one as -1 and the other as 1; either coding carries the same information.

Exercise 2.6 (Turning a category into numbers (part 2)). Use the setup of Exercise 2.5.

How would you turn a categorical^a feature with K categories into numbers, without implying that one category is larger than another?

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

Solution

Solution 2.6. Give each category its own column. A car from the second region becomes the vector $(0, 1, 0, \dots, 0)$: a 1 in the column for its category and 0 in every other column. Every pair of categories then differs in exactly two columns, so the coding ranks no category above or between any other.

Definition 2.1 (One-hot encoding (indicator coding)). The **one-hot encoding** of a categorical feature with categories $1, \dots, K$ maps category k to the indicator vector^a $e_k \in \{0, 1\}^K$, which has a 1 in position k and 0 in every other position [16].

Other names: **indicator coding**. Statistics texts often drop the column for one *reference* category, leaving $K - 1$ columns, and call the result *dummy coding* or *treatment coding*.

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

Exercise 2.7 (One-hot or integers?). One-hot encoding costs K columns where numbering the categories $1, 2, \dots, K$ costs one. When is the cheaper encoding the right one?

Solution

Solution 2.7. Integer encoding is right when the categories are genuinely **ordered** and the spacing between them is meaningful: a rating of *poor*, *fair*, *good* carries an order, and a model treating it as

1, 2, 3 is using real information.

Exercise 2.8 (One-hot or integers? (part 2)). Use the setup of Exercise 2.7. What goes wrong when the cheaper encoding is not the right one?

Solution

Solution 2.8. It goes wrong on unordered categories, because it invents an order and a spacing that are not there. Encoding red, green, blue as 1, 2, 3 tells a linear model that green sits halfway between red and blue, and that blue is three times red. No such statement was in the data. Tree-based models are the usual exception: they split on thresholds rather than multiplying by weights, so an arbitrary integer coding costs them much less.

i Watch

Starmer compares one-hot, label and target encoding^a [17].

^a<https://www.youtube.com/watch?v=589nCGeWG1w>

i Read

GeeksforGeeks explains:

- one-hot encoding^a [18]
- label encoding^b [19]

^a<https://www.geeksforgeeks.org/machine-learning/ml-one-hot-encoding/>

^b<https://www.geeksforgeeks.org/machine-learning/ml-label-encoding-of-datasets-in-python/>

Exercise 2.9 (How many columns?). The Auto data [8] describe cars by numeric measurements, a region of origin, and a name. Take `mpg` as the output and every other column as a feature. Load `data/auto.csv`^a from this site, and count how many features p the feature vector has after one-hot encoding both `origin` and `name`.

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

Solution

Solution 2.9.

2.1.1 Python

```
import pandas as pd

auto = pd.read_csv("data/auto.csv")
features = auto.drop(columns="mpg")
encoded = pd.get_dummies(features, columns=["origin", "name"])
print(features.shape[1], encoded.shape[1])
```

8 310

2.1.2 R

```

auto <- read.csv("data/auto.csv")
features <- dplyr::select(auto, -mpg)
categorical <- c("origin", "name")

n_numeric <- ncol(features) - length(categorical)
n_levels <- features |>
  dplyr::select(dplyr::all_of(categorical)) |>
  vapply(function(x) length(unique(x)), integer(1))
n_encoded <- n_numeric + sum(n_levels)
print(c(n_numeric, n_levels, n_encoded = n_encoded))

```

	origin	name	n_encoded
6	3	301	310

The 6 numeric features keep one column each. One-hot encoding turns **origin** into 3 columns and **name** into 301 columns, so the feature vector has $p = 310$ features.

Exercise 2.10 (How many columns? (part 2)). Use the setup of Exercise 2.9. What does the count say about encoding **name** this way?

Solution

Solution 2.10. p counts the features in the vector, and K counts the categories of one feature; a feature with K categories contributes K of the p columns. Here **name** alone contributes almost all of them. The data hold 392 cars and 301 distinct names, so the **name** columns nearly identify each car, and a model could fit the training set by memorizing which car is which. Such a model tells us nothing about a car it has not seen.

2.2 Feature scaling

Exercise 2.11 (Distance between two cars). Describe each car in `data/auto.csv`^a by two features: **weight**, in pounds, and **acceleration**, the time in seconds to reach 60 miles per hour. Many methods compare two inputs by the squared Euclidean distance^b between their feature vectors. For the first two cars in the file, what share of the squared distance comes from **weight**?

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

^b<https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-euclidean-distance>

Solution

Solution 2.11.

2.2.1 Python

```

import pandas as pd

cars = pd.read_csv("data/auto.csv")[["weight", "acceleration"]]
raw_gap = (cars.iloc[0] - cars.iloc[1]) ** 2
scaled = (cars - cars.mean()) / cars.std()
scaled_gap = (scaled.iloc[0] - scaled.iloc[1]) ** 2
print(raw_gap["weight"] / raw_gap.sum(), scaled_gap["weight"] / scaled_gap.sum())
0.9999930013647339 0.601174200694182

```

2.2.2 R

```
cars <- read.csv("data/auto.csv") |> dplyr::select(weight, acceleration)
raw_gap <- unlist((cars[1, ] - cars[2, ])^2)
share_raw <- raw_gap[["weight"]] / sum(raw_gap)

scaled <- cars |>
  scale() |>
  tibble::as_tibble()
scaled_gap <- unlist((scaled[1, ] - scaled[2, ])^2)
share_scaled <- scaled_gap[["weight"]] / sum(scaled_gap)
print(c(share_raw = share_raw, share_scaled = share_scaled))
```

```
share_raw share_scaled
0.9999930 0.6011742
```

weight contributes 99.9993% of the squared distance. The two cars differ by 189 pounds and by 0.5 seconds, and the distance sees only the size of those numbers, not their units. Measuring weight in tons instead would make the distance depend almost entirely on **acceleration**.

Exercise 2.12 (Distance between two cars (part 2)). Use the setup of Exercise 2.11. Propose a change to the features that makes both of them contribute comparably.

Solution

Solution 2.12. Measure each feature in units of its own spread: subtract the feature's mean^a and divide by its standard deviation^b. After that change, **weight** contributes 60.1% of the squared distance, and the share reflects how unusual each difference is rather than which unit each feature happens to be recorded in.

Scale matters to methods that measure distance and to methods that reduce the loss in fixed-size steps, since a step that is appropriate for one feature is far too large or too small for another. Features on very different scales can also make the arithmetic numerically unstable. Tree-based methods split one feature at a time on a threshold, so rescaling a feature does not change them.

^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>

Definition 2.2 (Standardization (z-scoring, standard scaling)). The **standardization** [20] of feature i replaces each value x_i with

$$\hat{x}_i \stackrel{\text{def}}{=} \frac{x_i - \hat{\mu}_i}{\hat{\sigma}_i}, \quad (1)$$

where $\hat{\mu}_i$ and $\hat{\sigma}_i$ are the sample mean^a and standard deviation^b of feature i over the training set. Other names: **z-scoring**, **standard scaling**.

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

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

i Remark (Standardization in the statistics notes)

Remark 2.1 (Standardization in the statistics notes). The statistics notes give the same formula for one variable: standardization^a.

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

Definition 2.3 (Min–max normalization (min–max scaling)). The **min–max normalization** [20] of feature i replaces each value x_i with

$$\tilde{x}_i \stackrel{\text{def}}{=} \frac{x_i - x_i^{\min}}{x_i^{\max} - x_i^{\min}}, \quad (2)$$

where $x_i^{\min}$ and $x_i^{\max}$ are the minimum and maximum of feature i over the training set. Other names: **min–max scaling**.

i Remark (Range of the normalized values, and the statistics notes)

Remark 2.2 (Range of the normalized values, and the statistics notes). **Min–max normalization** puts training values within $[0, 1]$. The statistics notes give the same formula for one variable: min–max scaling^a.

^a<https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-min-max-scaling>



Figure 5: The weight and acceleration of the 392 cars in `data/auto.csv`¹, raw, standardized and min–max normalized. In the raw panel both axes use the same number of units per pixel, as a distance computed from the raw features does, so weight, in thousands of pounds, spreads the cars along a line and acceleration, which spans under 20 seconds, barely registers. After either rescaling the two features have comparable spreads, and the shape of the cloud is the same in all three panels. The rescaled panels are drawn on the rescaled values, but their ticks read in pounds and seconds.

i Read

GeeksforGeeks explains feature scaling, normalization and standardization^a [21].

^a<https://www.geeksforgeeks.org/machine-learning/feature-engineering-scaling-normalization-and-standardization/>

Exercise 2.13 (Which data determine the scale?). Equation 1 needs a sample mean $\hat{\mu}_i$ and a standard deviation $\hat{\sigma}_i$ for each feature. Should they be computed from the training set, or from all the data, including the test set?

Solution

Solution 2.13. From the training set only. The mean and standard deviation are fitted to data, just as the model’s parameters are, so they are part of training. Computing them from all the data lets the test set shape the model before the test set is used to judge it. That leaks information out of the held-out

data that the risk-estimation solution set aside, and the resulting estimate of risk is no longer honest. New inputs, test inputs included, are scaled with the training set's $\hat{\mu}_i$ and $\hat{\sigma}_i$. The same holds for $x_i^{\min}$ and $x_i^{\max}$ in Equation 2, so a test value can fall outside $[0, 1]$.

References

- [1] B. Hutchinson, “DATA 471/571 (machine learning) and CSCI 481/581 (deep learning) video lectures.” Western Washington University. Accessed: Sep. 28, 2026. [Online]. Available: https://facultyweb.cs.wvu.edu/~hutchib2/video_lectures/data371/
- [2] J. Starmer, “A gentle introduction to machine learning.” StatQuest, video on YouTube, 2018. Available: https://www.youtube.com/watch?v=Gv9_4yMHFhI
- [3] Wikipedia contributors, “Sequence labeling — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Sequence_labeling
- [4] Wikipedia contributors, “Seq2seq — Wikipedia, the free encyclopedia.” 2026. Available: <https://en.wikipedia.org/wiki/Seq2seq>
- [5] Wikipedia contributors, “Learning to rank — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Learning_to_rank
- [6] Wikipedia contributors, “Structured prediction — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Structured_prediction
- [7] Wikipedia contributors, “Density estimation — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Density_estimation
- [8] 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/>
- [9] Wikipedia contributors, “Anomaly detection — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Anomaly_detection
- [10] Wikipedia contributors, “Nonparametric statistics — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Nonparametric_statistics
- [11] GeeksforGeeks, “Difference between parametric and non-parametric methods.” GeeksforGeeks, tutorial article, 2026. Available: <https://www.geeksforgeeks.org/machine-learning/difference-between-parametric-and-non-parametric-methods/>
- [12] M. Hopkins, E. Reeber, G. Forman, and J. Suermondt, “Spambase.” UCI Machine Learning Repository, 1999. doi: 10.24432/C53G6X².
- [13] Wikipedia contributors, “Discriminative model — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Discriminative_model
- [14] Wikipedia contributors, “Generative model — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Generative_model
- [15] GeeksforGeeks, “The difference between generative and discriminative machine learning algorithms.” GeeksforGeeks, tutorial article, 2025. Available: <https://www.geeksforgeeks.org/machine-learning/the-difference-between-generative-and-discriminative-machine-learning-algorithms/>
- [16] Wikipedia contributors, “One-hot — Wikipedia, the free encyclopedia.” 2026. Available: <https://en.wikipedia.org/wiki/One-hot>
- [17] J. Starmer, “One-hot, label, target and k-fold target encoding, clearly explained.” StatQuest, video on YouTube, 2023. Available: <https://www.youtube.com/watch?v=589nCGeWG1w>
- [18] GeeksforGeeks, “One hot encoding in machine learning.” GeeksforGeeks, tutorial article, 2026. Available: <https://www.geeksforgeeks.org/machine-learning/ml-one-hot-encoding/>
- [19] GeeksforGeeks, “Label encoding in Python.” GeeksforGeeks, tutorial article, 2026. Available: <https://www.geeksforgeeks.org/machine-learning/ml-label-encoding-of-datasets-in-python/>
- [20] Wikipedia contributors, “Feature scaling — Wikipedia, the free encyclopedia.” 2026. Available: https://en.wikipedia.org/wiki/Feature_scaling
- [21] GeeksforGeeks, “Feature engineering - scaling, normalization and standardization.” GeeksforGeeks, tutorial article, 2026. Available: <https://www.geeksforgeeks.org/machine-learning/feature-engineering-scaling-normalization-and-standardization/>

²<https://doi.org/10.24432/C53G6X>