Supervised and Unsupervised Learning
(for high school and first-year university students)

The next step after least squares and gradient descent

Planned and edited by Yoshio Okamoto / Written by Claude Opus 5.5 (Anthropic)

Ver. 0.1 October 7, 2026

Note: The following content was generated by AI. The editor has checked it, but please be aware that it may still contain errors or mistaken assumptions.

© 2026 Yoshio Okamoto. Free to use for educational purposes (please cite the source).

Introduction

These notes are for people who have learned the least-squares method and gradient descent and want to take the next step. Using typical methods as examples, they explain the two most basic frameworks of machine learning: **supervised learning** and **unsupervised learning**.

Section	Content	Typical method
1	Classification and overview of machine learning	—
2–3	Supervised learning (regression)	Linear regression (review of least squares)
4	Supervised learning (classification)	Logistic regression
5	Generalization and overfitting	Polynomial regression, regularization
6–7	Unsupervised learning (clustering)	The k-means method
8	Unsupervised learning (dimensionality reduction)	Principal component analysis (PCA)
9	Summary	—
Appendices	Mathematical explanations, Python code	—

How to read these notes The main text is written so that it can be read with high-school mathematics (trigonometric functions, exponentials and logarithms, differentiation, the summation sign $\sum$, and the variance and covariance of data). In a few places it goes slightly beyond high school, but an explanation in words is given on the spot. Parts that need tools learned at university (matrices, derivations by partial differentiation, eigenvalues, etc.) are collected in the **footnotes** and in **Appendices A–F** at the end. You can follow the main text even if you skip the appendices. All numerical values in the examples were actually computed (programs in Appendix F).

Notation

- The number of data points is n, and the data are numbered with the subscript i ($i = 1, 2, \ldots, n$). x_i is the value of the i-th data point.
- $\displaystyle\sum_{i=1}^{n} a_i$ means $a_1 + a_2 + \cdots + a_n$.
- $\hat{y}$ ("y hat") is **a value predicted by a model**, and $\bar{x}$ ("x bar") is **a mean**.
- log is the natural logarithm (the logarithm to base $e = 2.718\ldots$).

1 Classifying machine learning

1.1 What does "learning" do?

The least-squares method followed these steps.

1. Choose a model (e.g., a straight line $y = wx + b$).
2. Choose a function that measures how good the model is (e.g., the sum of the squared residuals).
3. Determine the parameters (w, b) so that this function is minimized.

Many methods of machine learning have this three-step form. The function in step 2 is called the **loss function** (or objective function), and gradient descent and the like are used for the minimization in step 3.

> Machine learning = finding, from data, the parameters that minimize the loss function

What differs from method to method is **what data is used and what is taken as the loss**. Machine learning is broadly classified by this difference.

1.2 Do the data have "correct answers"?

	Supervised learning	Unsupervised learning
Data used	Pairs of input and correct answer (x_i, y_i)	Inputs only, x_i
Goal	Obtain a rule that predicts the correct answer from the input	Find structure in the data
Basis of the loss	The gap between prediction and correct answer	How well the data themselves cluster or spread
Typical tasks	Regression, classification	Clustering, dimensionality reduction
Example	Predict weight from height	Divide customers into groups by buying habits

> The "supervisor" (teacher) in **supervised learning** is the **correct answer** (the **label**) attached to each data point. It is like a teacher giving you problems together with their answers.
> In **unsupervised learning**, no correct answers are given. Looking only at the data, we find **structure** such as "groups of similar things" or "the directions in which the data mainly spread."

There is also a separate framework, **reinforcement learning**, which learns using "rewards" obtained as a result of actions, but it is not covered in these notes.

2 The framework of supervised learning

2.1 Setting up the problem

In supervised learning, n pairs of input and correct answer are given:

$$(x_1, y_1),\ (x_2, y_2),\ \ldots,\ (x_n, y_n).$$

x_i is called the **input** (feature), and y_i the **output** (correct answer, label). The input is not necessarily a single number; it may be a set of several numbers (a vector), such as "height and age."

The goal is to prepare a model $\hat{y} = f(x)$ containing parameters, and to determine the parameters so that the predictions $\hat{y}_i = f(x_i)$ are close to the correct answers y_i. "Closeness" is measured by the loss function

$$L = \sum_{i=1}^{n} \big(\text{the gap between prediction } \hat{y}_i \text{ and correct answer } y_i \text{ for the } i\text{-th data point}\big).$$

How the gap for one data point is measured depends on the method.

2.2 Regression and classification

Supervised learning is divided into two kinds by the type of the output y.

- **Regression**: y is a continuous numerical value (weight, temperature, earthquake magnitude, etc.)
- **Classification**: y is one of several kinds (pass/fail, dog/cat/bird, etc.)

The typical method of regression is **linear regression** (Section 3), and of classification **logistic regression** (Section 4).

3 Regression: linear regression (review of least squares)

3.1 Example: predicting weight from height

We have the height x (cm) and weight y (kg) of five people.

Person	1	2	3	4	5
Height x	150	160	165	170	180
Weight y	45	52	58	62	70

Take the model to be the straight line $\hat{y} = wx + b$, and the loss to be the sum of the squared residuals (the differences between correct answer and prediction):

$$L(w, b) = \sum_{i=1}^{n} \big(y_i - (wx_i + b)\big)^2.$$

This is the **least-squares method**.

3.2 The solution

L is a function of two variables, w and b. So we differentiate with respect to w treating b as a constant, and with respect to b treating w as a constant, and set both to 0.[1] Solving these (computation in Appendix A.1) gives

$$w = \frac{S_{xy}}{S_{xx}}, \qquad b = \bar{y} - w\bar{x},$$

[1] Differentiating with respect to one variable while treating the others as constants is called **partial differentiation**, written like $\partial L/\partial w$. In terms of a mountain slope, $\partial L/\partial w$ is the slope in the east–west direction and $\partial L/\partial b$ the slope in the north–south direction. A point where both slopes are 0 is a candidate for the valley bottom (the minimum).

where $\bar{x}$ and $\bar{y}$ are the means, and

$$S_{xx} = \sum_i (x_i - \bar{x})^2, \qquad S_{xy} = \sum_i (x_i - \bar{x})(y_i - \bar{y}).$$

S_{xx} divided by n is the variance of x, and S_{xy} divided by n is the covariance of x and y, so w can also be described as "covariance ÷ variance of x."

In the example, $\bar{x} = 165$, $\bar{y} = 57.4$, $S_{xx} = 500$ and $S_{xy} = 425$, so

$$w = \frac{425}{500} = 0.85, \qquad b = 57.4 - 0.85 \times 165 = -82.85.$$

That is, we obtained the rule "a person 1 cm taller is about 0.85 kg heavier." The weight of a person 175 cm tall is predicted to be $0.85 \times 175 - 82.85 = 65.9$ kg.

> This is the simplest example of supervised learning. From data with correct answers (weights), we obtained a rule that predicts the output for an input we have not yet seen (175 cm).

3.3 When there are several inputs

Even with several inputs (for example, predicting weight from height and age), the idea is the same. Take the model

$$\hat{y} = b + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$$

($x_1, \ldots, x_d$ are the d inputs), and determine $b, w_1, \ldots, w_d$ so that the sum of the squared residuals is smallest. This too can be solved as a system of linear equations by setting the partial derivatives of the loss with respect to each parameter to 0. At university, this system of equations is written as a single formula (the normal equation) using **matrices** (Appendix A.2).

Linear regression is a special case in which the solution can be written as a formula (there is an analytical solution). In many methods of machine learning, the solution cannot be written as a formula, so it is found numerically by **gradient descent**. Gradient descent was the method of repeatedly "moving each parameter a little in the direction opposite to the slope (partial derivative) of the loss." Logistic regression in the next section is an example.

4 Classification: logistic regression

4.1 Example: predicting pass or fail from study time

We have the study time x (hours) and exam result y (pass = 1, fail = 0) of eight people.

Person	1	2	3	4	5	6	7	8
Time x	1	2	3	4	5	6	7	8
Result y	0	0	0	1	0	1	1	1

As shown by the person who passed after 4 hours and the person who failed after 5 hours, pass or fail is not cleanly determined by study time alone. So instead of "pass or fail itself," we predict **the probability of passing**.

4.2 Why a straight line does not work

If we used $\hat{y} = wx + b$ as it is, then $\hat{y} > 1$ for large x and $\hat{y} < 0$ for small x, which cannot be interpreted as a probability. So we pass the value of the line, $z = wx + b$, through a function that squeezes it between 0 and 1.

4.3 The sigmoid function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

is called the **sigmoid function** (logistic function).

z	$-\infty$	-2	0	2	$+\infty$
$\sigma(z)$	0	0.119	0.5	0.881	1

When z is large, e^{-z} approaches 0, so $\sigma(z)$ approaches 1; when z is small, e^{-z} becomes very large, so $\sigma(z)$ approaches 0. The value always satisfies $0 < \sigma(z) < 1$, and the graph is S-shaped (Figure 1(a)). Using this, we take

$$p = \sigma(wx + b)$$

as the model of "the probability that a person who studied x hours passes." This is **logistic regression**.[2]

$p = 0.5$ when $wx + b = 0$, that is, when $x = -b/w$. This is called the **decision boundary**; we predict pass if $p \geq 0.5$ and fail if $p < 0.5$.

4.4 The loss function: cross-entropy

In classification, instead of the squared residuals, the following **cross-entropy** is used as the loss:

$$L(w,b) = -\sum_{i=1}^{n}\Big[\, y_i \log p_i + (1-y_i)\log(1-p_i)\,\Big], \qquad p_i = \sigma(wx_i + b).$$

The formula looks complicated, but its content is simple. Since y_i is 0 or 1,

- for a person with $y_i = 1$ (pass), only $\log p_i$ remains inside [ ];
- for a person with $y_i = 0$ (fail), only $\log(1-p_i)$ remains inside [ ].

p_i is the probability given to "pass," and $1 - p_i$ the probability given to "fail."

> That is, each person's loss is **minus the logarithm of the probability given to the correct answer**. Giving probability 1 to the correct answer makes the loss $-\log 1 = 0$, and the smaller the probability given to the correct answer, the larger the loss ($-\log 0.5 \approx 0.69$, $-\log 0.1 \approx 2.30$).

> Minimizing the cross-entropy is the same as "choosing the parameters so that the observed results are most likely to occur under the model." This is called the **maximum likelihood method**.[3] Thinking in the same way, the least-squares method is the maximum likelihood method under the assumption that "the errors follow a normal distribution." The choice of loss function is tied to probabilistic assumptions about the data.

[2] Despite the word "regression" in its name, it is a classification method. Solving $p = \sigma(z)$ for z gives $z = \log \dfrac{p}{1-p}$. $\dfrac{p}{1-p}$ is the **odds** (the ratio of the probability of passing to that of failing), so logistic regression can also be described as the model "the logarithm of the odds is a linear function of x" (Appendix B.1).

[3] The product over everyone of the probability that each person's result comes out as observed (p_i if $y_i = 1$, $1 - p_i$ if $y_i = 0$) is called the **likelihood**. The logarithm of a product is the sum of the logarithms, so minus the logarithm of the likelihood is the cross-entropy. See Appendix B.2 for details.

4.5 The gradient

To use gradient descent, we need the partial derivatives (the gradient) of L with respect to w and b. Computing them (derivation in Appendix B.3) gives the simple form

$$\frac{\partial L}{\partial w} = \sum_{i=1}^{n} (p_i - y_i)\, x_i, \qquad \frac{\partial L}{\partial b} = \sum_{i=1}^{n} (p_i - y_i).$$

> $p_i - y_i$ is "the gap between the predicted probability and the correct answer." The gradient is **"gap × input" added up over everyone**.

In fact, the gradient of the loss of linear regression also has the form of "residual × input" added up (Appendix A.2).[4]

Using this gradient, we repeat gradient descent:

$$w \;\leftarrow\; w - \eta \cdot \frac{1}{n} \frac{\partial L}{\partial w}, \qquad b \;\leftarrow\; b - \eta \cdot \frac{1}{n} \frac{\partial L}{\partial b}.$$

η (eta) is the **learning rate**, which decides how far to move at each step. Here the gradient is divided by n to make it a mean.

4.6 Results of the example

Starting from $w = b = 0$ and repeating gradient descent with learning rate $\eta = 0.1$ (program in Appendix F.1), the process goes as follows.

Iterations	w	b	Loss L
0	0	0	5.545
10	0.165	−0.124	4.896
100	0.368	−1.196	3.826
1000	0.997	−4.372	2.576
20000	1.282	−5.770	2.503

In the initial state ($w = b = 0$), $z = 0$, so everyone is given probability 0.5 and the loss is $8 \times (-\log 0.5) = 8 \log 2 \approx 5.545$. As training proceeds the loss decreases and converges to about 2.503. The probabilities of passing given by the resulting model are as follows.

Time x	1	2	3	4	5	6	7	8
Probability p	0.011	0.039	0.127	0.345	0.655	0.873	0.961	0.989
Result y	0	0	0	1	0	1	1	1

The decision boundary is $x = -b/w = 5.770/1.282 \approx 4.5$ hours.

> The person with 4 hours (pass) and the person with 5 hours (fail) are on the "wrong sides" of the boundary, but the model does not force itself to fit them; instead it **expresses this as "uncertainty," with probabilities 0.345 and 0.655**. This is why the loss does not go down to 0.

[4] The loss L of logistic regression is convex (bowl-shaped, like the quadratic function $y = x^2$), so gradient descent always reaches the minimum. On the other hand, using the squared residuals on the output of the sigmoid function makes the loss non-convex and the gradient tends to become small, so cross-entropy is used in classification (Appendix B.4).

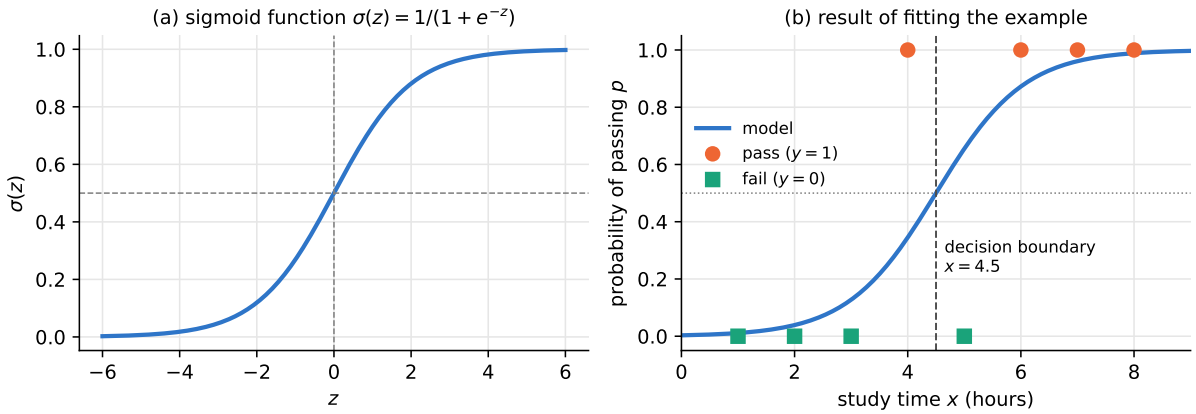


Figure 1 (a) The sigmoid function. It is 0.5 at $z = 0$ and approaches 1 and 0 as $z \to \pm\infty$. (b) The data of the example (points) and the model obtained by training, $p = \sigma(1.282x - 5.770)$ (curve). The dashed line is the decision boundary $x = 4.5$.

4.7 Extension to three or more classes

When classifying into three or more kinds (e.g., dog/cat/bird), a score $z_k = w_k x + b_k$ ($k = 1, \ldots, K$) is computed for each kind, and instead of the sigmoid function, the **softmax function**

$$P(\text{kind } k) = \frac{e^{z_k}}{e^{z_1} + e^{z_2} + \cdots + e^{z_K}}$$

turns them into probabilities. Since e^z is always positive, all the probabilities are positive, and since the denominator is the sum of all of them, they add up to exactly 1. With two kinds ($K = 2$), this is the same as logistic regression.[5] Large language models (LLMs) such as ChatGPT also use this softmax function to output "the probabilities of the next token (word fragment)." It is a very large multi-class classification, with the number of kinds of tokens K reaching more than a hundred thousand.[6]

5 Generalization and overfitting

5.1 The goal is to predict "unseen data" well

What we really want to know in supervised learning is not the error on the data used for training (the **training data**), but **the error on data not yet seen**. The ability to predict well even for unseen data is called **generalization**.

So in practice, the data at hand are divided into training data and **test data**; the model is trained only on the training data, and its performance is evaluated on the test data.

5.2 Example: changing the degree of a polynomial

Consider data on $0 \le x \le 1$ in which a small error (noise) is added to $y = \sin 2\pi x$. The training data are 10 points ($x = 0, \frac{1}{9}, \frac{2}{9}, \ldots, 1$), and the test data are 10 points different from the training data ($x = 0.05, 0.15, \ldots, 0.95$).

[5] Dividing numerator and denominator by e^{z_1} gives $P(\text{kind } 1) = \dfrac{1}{1 + e^{-(z_1 - z_2)}} = \sigma(z_1 - z_2)$, which agrees with the sigmoid function.

[6] For example, some models use about 128,000 kinds of tokens. How LLMs work is dealt with in other documents (such as "Reading microgpt").

We fit a polynomial of degree M,

$$\hat{y} = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M,$$

by the least-squares method (Figure 2). Although it is a polynomial in x, it is linear in the coefficients w_j, so it can be solved by the same method as "when there are several inputs" in Section 3.3 (regarding $x, x^2, \ldots, x^M$ as separate inputs).

The errors are compared using the square root of the mean of the squared residuals (**RMSE**, root mean square error):

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}.$$

It has the same form as the standard deviation and expresses "how far off the predictions are on average." The results are as follows.

Degree M	Training RMSE	Test RMSE	Largest absolute coefficient
0 (constant)	0.711	0.732	0.002
1 (straight line)	0.572	0.502	1.3
3	0.216	0.168	32.5
9	0.000	0.926	about 6×10^5

- With $M = 0, 1$, the model is too simple to represent the shape of the sine, so the errors are large for both training and test data. This is called **underfitting**.
- With $M = 3$, the errors are small for both training and test data.
- With $M = 9$, 10 coefficients can make a curve passing through all 10 points, so the training error becomes 0. But the curve oscillates wildly between the points, and the test error is worse than with $M = 1$. This is called **overfitting**. The extreme values of the coefficients, in the hundreds of thousands, are a sign of overfitting.

Fitting the training data perfectly and predicting well are different things.
An overfitted model has "memorized" not only the rule behind the data ($\sin 2\pi x$) but also the noise that happened to be added.

5.3 Regularization

A typical way to suppress overfitting is **regularization**. A penalty for large coefficients is added to the loss function:

$$L = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_j w_j^2.$$

The second term is the sum of the squares of the coefficients multiplied by λ (lambda); the larger the coefficients, the larger the loss. $\lambda > 0$ is a constant that decides the strength of the penalty. This is called **ridge regression**. Ridge regression too can be solved as a formula with almost the same computation as least squares (Appendix A.3).

The results of adding regularization to the polynomial with $M = 9$ are as follows.

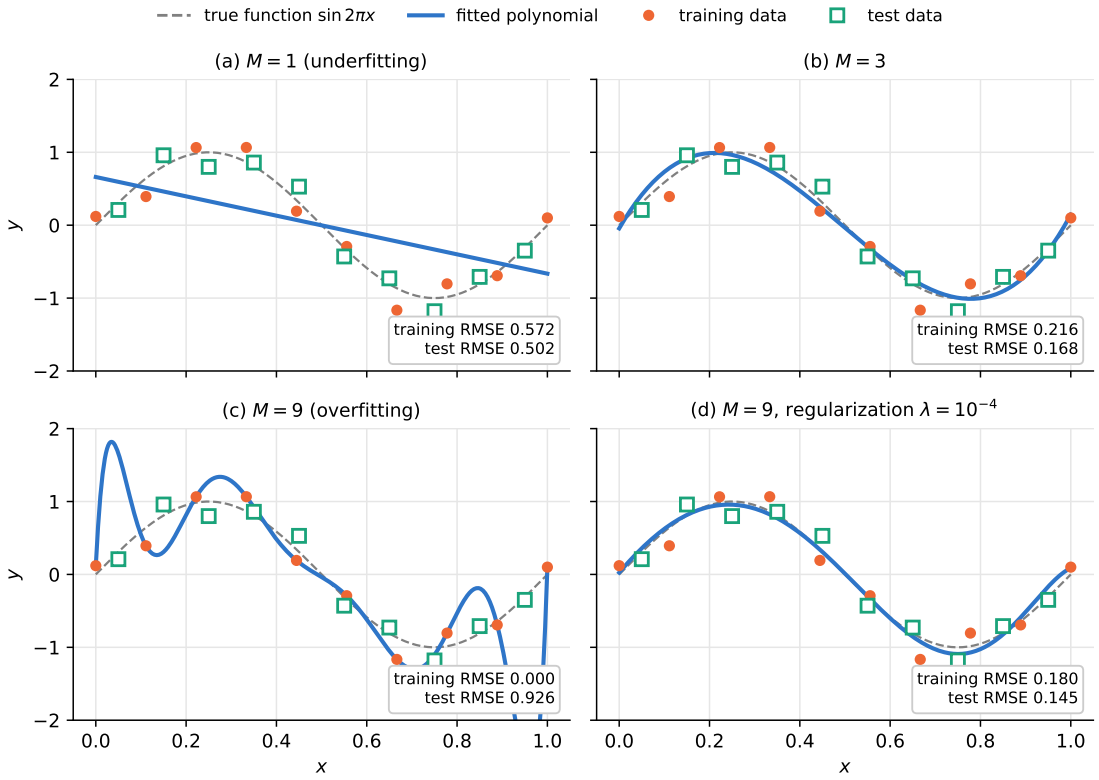


Figure 2 Polynomial degree and overfitting. The gray dashed line is the true function that generated the data, the orange dots are the training data, the green squares are the test data, and the blue curve is the polynomial fitted to the training data. In (c), the curve passes through all the training data but oscillates widely between the points and misses the test data. (d) is the same degree-9 polynomial as (c) with regularization added (Section 5.3).

λ	Training RMSE	Test RMSE	Largest absolute coefficient
0 (no regularization)	0.000	0.926	about 6×10^5
10^{-6}	0.125	0.153	136
10^{-4}	0.180	0.145	12.2
10^{-3}	0.213	0.171	10.8
10^{-2}	0.331	0.288	3.9

Adding even a very small λ stops the runaway coefficients (Figure 2(d)) and greatly improves the test error. However, if λ is made too large, the model becomes too simple and moves toward underfitting.

As the model is made more complex (raising the degree M, lowering λ), the training error keeps decreasing. The test error, on the other hand, decreases at first but starts to increase beyond a certain complexity. This is because the prediction error has two kinds of components, "error because the model is too simple" and "error from being swayed by chance fluctuations in the training data," and making the model more complex decreases the former but increases the latter.[7]

> Quantities such as λ and M, which are decided in advance by people rather than by training, are called **hyperparameters**. Hyperparameters are chosen using data set aside separately from the test data (**validation data**). Choosing them with the test data would overfit to the test data.

6 The framework of unsupervised learning

6.1 Setting up the problem

In unsupervised learning, only data without correct answers, $x_1, x_2, \ldots, x_n$, are given (each x_i is usually a set of several numbers). Since there is no y to predict, a loss of "the gap between prediction and correct answer" cannot be made. Instead, **a criterion based on the properties of the data themselves** is used as the loss (objective function).

The two typical tasks are the following.

- **Clustering**: divide data into groups (clusters) of similar data. The criterion is "how tightly each group is gathered."
- **Dimensionality reduction**: summarize many variables into a few variables, losing as little information as possible. The criterion is "how much of the spread of the data the summarized variables keep."

6.2 The difficulty of unsupervised learning

> Since there are no correct answers in unsupervised learning, there is no direct way to check whether a result is "correct." For example, in clustering, whether dividing the data into two groups or three is correct often cannot be decided from the data alone. **Interpreting the results requires judgment based on knowledge of the field.**

[7] The former is called **bias** and the latter **variance**, and this relation is called the **bias–variance tradeoff** (Appendix C).

7 Clustering: the k-means method

7.1 The idea

We divide the data into K groups. Each group is given one **representative point** (center) $\mu_1, \ldots, \mu_K$ (mu), and each data point belongs to the group of the nearest representative point. A good division is one in which the distances between the data points and the representative points of their groups are small overall.

7.2 The objective function

The square of the distance between points (x, y) and (a, b) in the plane is $(x-a)^2 + (y-b)^2$. Using this, we take the objective function

$$J = \sum_{i=1}^{n} \big(\text{distance from point } i \text{ to the representative point of the group point } i \text{ belongs to}\big)^2$$

and find the "division into groups" and "positions of the representative points" that minimize J.[8]

7.3 The algorithm

Optimizing the division and the positions of the representative points at the same time is difficult, so we alternately repeat the operation of **fixing one and optimizing the other**.

1. **Initialization**: choose the representative points $\mu_1, \ldots, \mu_K$ somehow.
2. **Assignment**: put each data point in the group of the nearest representative point.
3. **Update**: move the representative point of each group to the **mean** (centroid) of the data in that group.
4. Repeat 2 and 3 until the assignment no longer changes.

Why does it work? In the assignment step, the representative points are not moved, and each data point is reattached to the nearest representative point. Every data point's distance becomes smaller or stays the same, so J decreases or stays the same. In the update step, the groups are fixed and the representative points are moved to the centroids. The position that minimizes "the sum of the squared distances from several points" is the mean (centroid) of those points,[9] so in this step too J decreases or stays the same.

J is 0 or more, and there are only finitely many ways to assign the points, so this algorithm always stops after a finite number of steps. However, there is no guarantee that where it stops is the overall minimum of J (it may stop at a **local minimum**). Therefore, in practice, it is run several times with different initial values, and the result with the smallest J is adopted (proof in Appendix D).

7.4 Example

We divide the following 8 points into $K = 2$ groups.

[8] University textbooks write this as $J = \sum_{i=1}^{n} \sum_{k=1}^{K} r_{ik} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$, using a variable r_{ik} that is 1 if point i belongs to group k and 0 otherwise. $\| \ \|$ is the length of a vector (Appendix D).

[9] Let us check this in one dimension. The sum of the squared distances from $x_1, \ldots, x_m$, $f(a) = \sum (x_i - a)^2$, is a quadratic function of a, and from $f'(a) = -2\sum(x_i - a) = 0$ it is smallest at $a = \frac{1}{m}\sum x_i$ (the mean). In the plane, the same holds by considering the x and y coordinates separately.

Point	1	2	3	4	5	6	7	8
Coordinates	$(1, 1)$	$(1.5, 2)$	$(2, 1.5)$	$(1, 2.5)$	$(5, 5)$	$(6, 5.5)$	$(5.5, 6.5)$	$(6.5, 6)$

A person can see at once that they divide into the 4 points at the lower left and the 4 points at the upper right. To check whether the k-means method can find this, we deliberately place the initial representative points badly, at points 1 and 2 at the lower left (program in Appendix F.2, Figure 3).

Iteration	Representative μ_1	Representative μ_2	Points in group 1	J
0	$(1, 1)$	$(1.5, 2)$	point 1 only	132.0
1	$(1, 1)$	$(3.929, 4.143)$	points 1–4	30.85
2	$(1.375, 1.75)$	$(5.75, 5.75)$	points 1–4	4.44

- Iteration 0: points 2–8 are all closer to the representative point $(1.5, 2)$, so group 2 has 7 points.
- Iteration 1: the centroid of group 2, $(3.929, 4.143)$, lies between the lower left and the upper right. Now points 2–4 are closer to the representative point $(1, 1)$ and move to group 1.
- Iteration 2: each centroid moves to the center of the lower-left or upper-right points, and the assignment no longer changes (convergence).

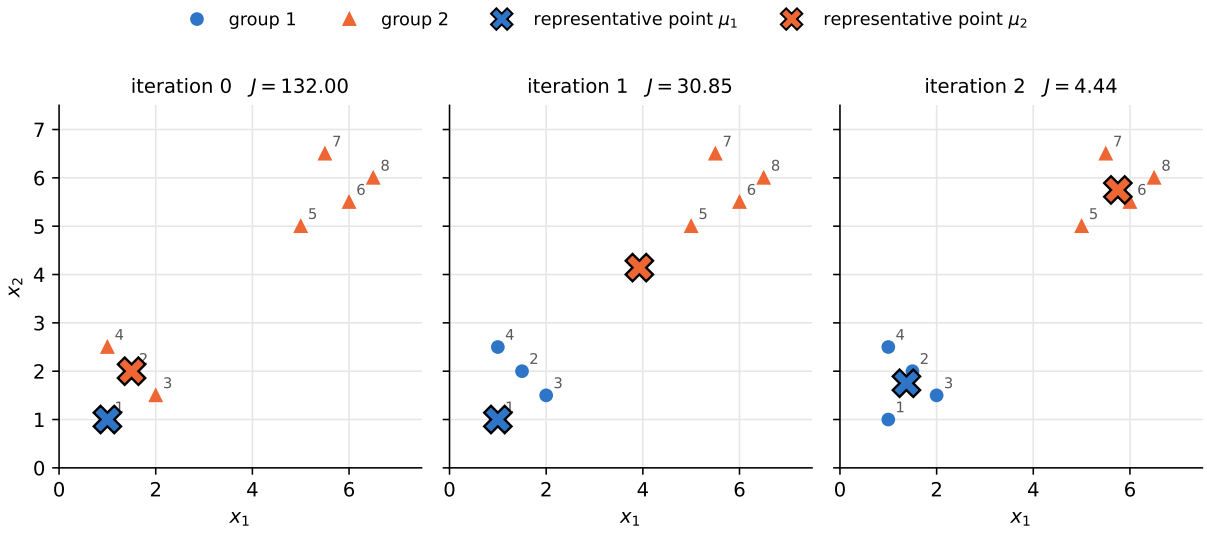


Figure 3 Iterations of the k-means method. Circles (blue) are points assigned to group 1, triangles (orange) points assigned to group 2, and the $\times$ marks are the representative points. The numbers are the point numbers.

Nobody taught it to "divide into lower left and upper right," yet **two groups were found from the arrangement of the data alone**. This is unsupervised learning.

7.5 Points to note

- The number of groups K is decided by people. Increasing K always decreases J (with $K = n$, each group has one point and $J = 0$), so K cannot be decided from the size of J alone.

- Since "distance" is used, the results depend on the units of the variables. For example, the result changes depending on whether height is measured in cm or m, so usually each variable is **standardized** (made to have mean 0 and standard deviation 1) before use.

8 Dimensionality reduction: principal component analysis (PCA)

8.1 The idea

Data with many variables are hard to grasp as a whole. **Principal component analysis** (PCA) finds **the direction in which the data spread out most**, and summarizes the original data by the coordinate along that direction.

8.2 Example: mathematics and science scores

Here are the mathematics and science scores of five people.

Person	A	B	C	D	E
Mathematics x_1	80	60	90	50	70
Science x_2	75	65	85	60	65

People with high mathematics scores tend to have high science scores too, and the points are distributed in a long, narrow region rising to the right. If this "rising-to-the-right direction" could be taken out as a single axis, the two scores could be summarized by one number (so to speak, "overall ability in mathematics and science").

8.3 Moving the mean to the origin

First, subtract the mean from each variable (**centering**). In the example, both means are 70 points. The values with the mean subtracted (deviations) and their variances and covariance are as follows.

Person	A	B	C	D	E
Mathematics deviation u	10	-10	20	-20	0
Science deviation v	5	-5	15	-10	-5

Variance of mathematics $s_u^2 = 200$, variance of science $s_v^2 = 80$, covariance $s_{uv} = 120$

Here the variances and covariance are the mean of the squared deviations and the mean of the products of the deviations (divided by $n = 5$). For example, the covariance is $(10 \cdot 5 + (-10)(-5) + 20 \cdot 15 + (-20)(-10) + 0 \cdot (-5))/5 = 600/5 = 120$.

8.4 Projecting onto various directions

Consider a straight line through the origin (the point of the means), tilted at an angle θ from the u axis. The vector of length 1 pointing along this line is $(\cos\theta, \sin\theta)$. The coordinate of the foot of the perpendicular dropped from each person's point (u_i, v_i) to this line (the **projected** coordinate) is the dot product

$$z_i = u_i \cos\theta + v_i \sin\theta.$$

If $\theta = 0°$, $z_i = u_i$ (the mathematics deviation itself); if $\theta = 90°$, $z_i = v_i$ (the science deviation itself).

The variance $V(\theta)$ of $z_1, \ldots, z_5$ lined up on the line can be expressed with the variances and covariance as[10]

$$V(\theta) = s_u^2 \cos^2\theta + 2 s_{uv} \sin\theta\cos\theta + s_v^2 \sin^2\theta = 200\cos^2\theta + 240\sin\theta\cos\theta + 80\sin^2\theta.$$

Computing it at several angles gives the following.

Angle θ	$0°$	$15°$	$30°$	$45°$	$60°$	$75°$	$90°$	$135°$
Variance $V(\theta)$	200.0	252.0	273.9	260.0	213.9	148.0	80.0	20.0

The spread is largest near $\theta = 30°$, and near $135°$, at right angles to it (the falling-to-the-right direction), there is hardly any spread. The fact that the data are distributed in a long, narrow, rising-to-the-right shape shows up in the numbers.

8.5 The direction of maximum variance

The angle of the maximum can be found exactly by combining the trigonometric functions. Using

$$\cos^2\theta = \frac{1+\cos 2\theta}{2}, \qquad \sin^2\theta = \frac{1-\cos 2\theta}{2}, \qquad \sin\theta\cos\theta = \frac{\sin 2\theta}{2},$$

obtained from the double-angle formulas, we can write

$$V(\theta) = 140 + 60\cos 2\theta + 120\sin 2\theta = 140 + \sqrt{60^2 + 120^2}\,\sin(2\theta + \alpha)$$

(α is the angle with $\tan\alpha = \frac{60}{120}$). Since $\sqrt{60^2 + 120^2} = \sqrt{18000} \approx 134.16$,

- the maximum of V is $140 + 134.16 = 274.16$, reached when $\tan 2\theta = \dfrac{120}{60} = 2$, that is, $\theta \approx 31.7°$;
- the minimum of V is $140 - 134.16 = 5.84$, and its direction is at right angles to that of the maximum ($\theta \approx 121.7°$).

> The direction of maximum variance,
>
> $$(\cos 31.7°,\ \sin 31.7°) \approx (0.851,\ 0.526),$$
>
> is called the direction of the **first principal component**, and the direction at right angles to it that of the **second principal component**.

The first principal component is the "rising-to-the-right" direction, with positive weights on both mathematics and science (the weight on mathematics is larger because the mathematics scores spread more).

8.6 Contribution ratio and principal component scores

From the formula above, adding the variances in two directions at right angles, θ and $\theta + 90°$, the sin terms cancel and the sum is always $140 \times 2 = 280$. This equals the sum of the variances of mathematics and science, $200 + 80 = 280$, and expresses the size of the spread of the whole data. The share of it taken by the first principal component (the **contribution ratio**) is

$$\frac{274.16}{280} \approx 0.979.$$

[10] The mean of z_i is 0, so the variance is $\frac{1}{n}\sum z_i^2$. Adding $z_i^2 = u_i^2\cos^2\theta + 2u_iv_i\sin\theta\cos\theta + v_i^2\sin^2\theta$ over everyone and dividing by n, using $\frac{1}{n}\sum u_i^2 = s_u^2$, $\frac{1}{n}\sum u_iv_i = s_{uv}$ and $\frac{1}{n}\sum v_i^2 = s_v^2$, gives this formula.

That is, about 98% of the spread of the two scores can be expressed by a single number, the first principal component.

Each person's coordinate on the first principal component (the **principal component score**) $z_i = 0.851\,u_i + 0.526\,v_i$ is as follows (program in Appendix F.3).

Person	A	B	C	D	E
Mathematics, science	$80, 75$	$60, 65$	$90, 85$	$50, 60$	$70, 65$
1st principal component score z	11.14	-11.14	24.90	-22.27	-2.63

The two-dimensional data have been arranged on a one-dimensional number line, giving the order C > A > E > B > D.

With three or more variables, a direction cannot be expressed by turning a single angle. At university, the principal components in the general case are computed by finding the **eigenvalues** and **eigenvectors** of the **variance-covariance matrix**, which lists the variances and covariances. The maximum 274.16 and minimum 5.84 found above are exactly the eigenvalues of this matrix (Appendix E).

8.7 Points to note

- The "meaning" of a principal component (here, "overall ability in mathematics and science") does not come out of the computation. People interpret it by looking at the components (weights) of the direction.
- As with the k-means method, the results are affected by the units of the variables. When handling variables with different units (height in cm and weight in kg, etc.), they are usually standardized first.
- With d variables, d principal components are obtained. Using the top few in order of contribution ratio, many variables can be summarized by a few.

9 Summary

9.1 Comparison of the four methods

	Linear regression	Logistic regression	k-means method	Principal component analysis
Type	Supervised (regression)	Supervised (classification)	Unsupervised (clustering)	Unsupervised (dimensionality reduction)
Data	(x_i, y_i), y numerical	(x_i, y_i), y is 0 or 1	x_i only	x_i only
Model	$\hat{y} = wx + b$	$p = \sigma(wx + b)$	Representative points μ_k	Direction $(\cos\theta, \sin\theta)$
Quantity optimized	Sum of squared residuals (minimize)	Cross-entropy (minimize)	Sum of squared distances to representative points (minimize)	Variance of the projection (maximize)
How it is solved	Solvable as a formula	Gradient descent	Alternating assignment and update	Find the direction of maximum variance (in general, an eigenvalue problem)

We see that every method is expressed as an optimization problem of "minimizing (or maximizing) something." The difference is in what is optimized.

- In supervised learning, the criterion of optimization is determined by **comparison with correct answers given from outside**.
- In unsupervised learning, the criterion of optimization is determined by **the properties of the data themselves** (how tightly they gather, how they spread).

9.2 Between supervised and unsupervised learning

The training of an LLM is the task of "predicting the next token of a text."[11] In that it minimizes the gap between prediction and correct answer (the cross-entropy), it has the same form as supervised learning, but the correct answers (the next tokens) are not attached by people; they are obtained automatically from the text itself. Such a method, which automatically creates correct answers from data without correct answers and learns in the form of supervised learning, is called **self-supervised learning**. Since there is no need to attach labels by hand, enormous amounts of text can be used for training.

9.3 Key points

1. Machine learning is an optimization problem of "finding the parameters that minimize a loss function."
2. Supervised learning learns rules of prediction from data with correct answers; unsupervised learning finds structure in data without correct answers.
3. Regression uses the sum of squared residuals as the loss, and classification uses the cross-entropy. Both can be understood as maximum likelihood methods.
4. Having a small error on the training data and predicting unseen data well are different things (overfitting).
5. Interpreting the results of unsupervised learning requires human judgment.

Appendix A The mathematics of linear regression

This appendix uses partial derivatives and matrices (first-year university linear algebra).

Notation Vectors are written in bold lowercase ($\mathbf{x}$) and matrices in uppercase (X). Unless stated otherwise, vectors are column vectors. The superscript T denotes the **transpose**, not a power. Transposing swaps the rows and columns of a matrix; the transpose A^{T} of a matrix A is the matrix whose (i, j) entry is A_{ji}. $\mathbf{a}^{\mathsf{T}}\mathbf{b} = a_1 b_1 + \cdots + a_d b_d$ is the dot product (row vector $\times$ column vector = one number), and $\|\mathbf{a}\| = \sqrt{\mathbf{a}^{\mathsf{T}}\mathbf{a}}$ is the length (norm) of a vector.

A.1 Derivation of w and b

We take the partial derivatives of $L(w, b) = \sum_i (y_i - wx_i - b)^2$ and set them to 0:

$$\frac{\partial L}{\partial b} = -2\sum_i (y_i - wx_i - b) = 0, \qquad \frac{\partial L}{\partial w} = -2\sum_i (y_i - wx_i - b)\, x_i = 0.$$

[11] How LLMs work and are trained is dealt with in other documents (such as "Reading microgpt").

Dividing the first equation by n gives $\bar{y} - w\bar{x} - b = 0$, that is, $b = \bar{y} - w\bar{x}$. Substituting this into the second gives $\sum_i \big((y_i - \bar{y}) - w(x_i - \bar{x})\big)x_i = 0$, and using $\sum_i (x_i - \bar{x}) = 0$ and $\sum_i (y_i - \bar{y}) = 0$ to replace x_i by $x_i - \bar{x}$, we obtain

$$S_{xy} - wS_{xx} = 0 \quad \Longrightarrow \quad w = \frac{S_{xy}}{S_{xx}}.$$

A.2 Writing with matrices: the normal equation

For d inputs, set

$$X = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1d} \\ 1 & x_{21} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{nd} \end{pmatrix}, \qquad \mathbf{w} = \begin{pmatrix} b \\ w_1 \\ \vdots \\ w_d \end{pmatrix}, \qquad \mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}.$$

x_{ij} is the j-th input of the i-th data point, and the 1s in the first column of X are for the intercept b. Then the prediction is $\hat{\mathbf{y}} = X\mathbf{w}$, and the loss can be written

$$L(\mathbf{w}) = \|\mathbf{y} - X\mathbf{w}\|^2 = (\mathbf{y} - X\mathbf{w})^\mathsf{T}(\mathbf{y} - X\mathbf{w}) = \mathbf{y}^\mathsf{T}\mathbf{y} - 2\mathbf{w}^\mathsf{T}X^\mathsf{T}\mathbf{y} + \mathbf{w}^\mathsf{T}X^\mathsf{T}X\mathbf{w}.$$

The partial derivatives with respect to the components of $\mathbf{w}$, arranged as a vector (the gradient vector), are

$$\nabla_\mathbf{w} L = -2X^\mathsf{T}\mathbf{y} + 2X^\mathsf{T}X\mathbf{w} = -2X^\mathsf{T}(\mathbf{y} - X\mathbf{w}).$$

Setting this to $\mathbf{0}$ gives the **normal equation**

$$X^\mathsf{T}X\,\mathbf{w} = X^\mathsf{T}\mathbf{y} \quad \Longrightarrow \quad \mathbf{w} = (X^\mathsf{T}X)^{-1}X^\mathsf{T}\mathbf{y}.$$

For $d = 1$, this agrees with the result of Appendix A.1.

The gradient $-2X^\mathsf{T}(\mathbf{y} - X\mathbf{w})$ has the form "the residuals $\mathbf{y} - X\mathbf{w}$ multiplied by the inputs and added up." It is the same "gap $\times$ input" form as the gradient of logistic regression (Section 4.5).

Polynomial regression The polynomial of degree M in Section 5.2 can be solved with the same normal equation by taking the i-th row of X to be $(1,\ x_i,\ x_i^2,\ \ldots,\ x_i^M)$.

A.3 The solution of ridge regression

The gradient of the regularized loss $L(\mathbf{w}) = \|\mathbf{y} - X\mathbf{w}\|^2 + \lambda\|\mathbf{w}\|^2$ is $-2X^\mathsf{T}(\mathbf{y} - X\mathbf{w}) + 2\lambda\mathbf{w}$, so setting it to $\mathbf{0}$ gives

$$(X^\mathsf{T}X + \lambda I)\,\mathbf{w} = X^\mathsf{T}\mathbf{y} \quad \Longrightarrow \quad \mathbf{w} = (X^\mathsf{T}X + \lambda I)^{-1}X^\mathsf{T}\mathbf{y}$$

(I is the identity matrix). Only λI is added to the normal equation.[12]

[12] Even when $X^\mathsf{T}X$ has no inverse, or computing its inverse is unstable, adding $\lambda > 0$ guarantees that $X^\mathsf{T}X + \lambda I$ has an inverse. This is another effect of regularization.

Appendix B The mathematics of logistic regression

B.1 Properties of the sigmoid function

Differentiating $\sigma(z) = 1/(1+e^{-z})$ gives

$$\sigma'(z) = \frac{e^{-z}}{(1+e^{-z})^2} = \sigma(z)\big(1-\sigma(z)\big),$$

expressed in terms of the function itself. Thanks to this property, the gradient found in B.3 takes a neat form.

Also, solving $p = \sigma(z)$ for z gives $z = \log\dfrac{p}{1-p}$. $\dfrac{p}{1-p}$ is the odds (the ratio of the probability of passing to that of failing), so logistic regression is the model "the logarithm of the odds is a linear function of x."

B.2 Where cross-entropy comes from (maximum likelihood)

If the model is correct, the probability that the results of the n people come out exactly as observed (the **likelihood**) is

$$\mathcal{L}(w,b) = \prod_{i=1}^{n} p_i^{y_i}(1-p_i)^{1-y_i}$$

($\prod$ is the product symbol; multiply by p_i if $y_i = 1$ and by $1-p_i$ if $y_i = 0$). Choosing the w, b that maximize the likelihood, as the parameters that best explain the observed data, is the **maximum likelihood method**.

A product is awkward to handle, so we take the logarithm, and add a minus sign to turn maximization into minimization:

$$-\log\mathcal{L} = -\sum_i \big[y_i \log p_i + (1-y_i)\log(1-p_i)\big],$$

which agrees with the cross-entropy.

Minimizing the cross-entropy is exactly the maximum likelihood method.

Least squares and the normal distribution In regression, if we assume that the correct answer arises as "prediction $\hat{y}_i$ + an error following a normal distribution with mean 0 and standard deviation s," the probability density of observing y_i is $\frac{1}{\sqrt{2\pi}s}\exp\!\big(-\frac{(y_i-\hat{y}_i)^2}{2s^2}\big)$. Multiplying over everyone, taking the logarithm and adding a minus sign gives, apart from constants, $\frac{1}{2s^2}\sum_i (y_i-\hat{y}_i)^2$. That is, maximizing the likelihood is the same as minimizing the sum of squared residuals: **the least-squares method is the maximum likelihood method under the assumption that the errors follow a normal distribution**.

B.3 Derivation of the gradient

Let the loss of one data point be $\ell = -[y\log p + (1-y)\log(1-p)]$, with $p = \sigma(z)$ and $z = wx+b$. By the chain rule for composite functions,

$$\frac{\partial \ell}{\partial w} = \frac{\partial \ell}{\partial p}\cdot\frac{\partial p}{\partial z}\cdot\frac{\partial z}{\partial w}.$$

Each factor is

$$\frac{\partial \ell}{\partial p} = -\frac{y}{p} + \frac{1-y}{1-p} = \frac{p-y}{p(1-p)}, \qquad \frac{\partial p}{\partial z} = p(1-p), \qquad \frac{\partial z}{\partial w} = x,$$

so multiplying them, $p(1-p)$ cancels and $\dfrac{\partial \ell}{\partial w} = (p-y)\,x$. Similarly, for b it is $(p-y)$. Adding over everyone gives the formulas of Section 4.5.

B.4 Convexity

The loss L of logistic regression is a convex function (of w, b), so gradient descent always reaches the minimum. On the other hand, if the squared residual $(y-p)^2$ is used on the output of the sigmoid function, the loss is not convex, and since a factor $p(1-p)$ remains in the gradient, the gradient becomes almost 0 where p is near 0 or 1 (where the prediction is badly wrong yet confident). This is why cross-entropy is used in classification.

Appendix C The bias–variance tradeoff

As stated in Section 5.3, making the model more complex (raising the degree M, lowering λ) decreases the training error monotonically, but the test error decreases at first and starts to increase beyond a certain complexity.

This is because the prediction error (the expected value of its square) can be roughly divided into the following three parts.

- **Bias**: the error because the model is too simple to represent the true function.
- **Variance**: the error from the model being swayed by chance fluctuations in the training data. It is larger for models whose fitted curves change greatly each time the training data are taken again.
- **Noise**: the error contained in the data themselves, which no model can reduce.

Making the model more complex decreases the bias but increases the variance; this relation is called the **bias–variance tradeoff**. Figure 2(a) is an example with large bias, and (c) an example with large variance.

Appendix D The mathematics of the k-means method

Let the coordinates of point i be the vector $\mathbf{x}_i$, the representative point of group k be $\boldsymbol{\mu}_k$, and r_{ik} be a variable that is 1 if point i belongs to group k and 0 otherwise. Then the objective function is

$$J = \sum_{i=1}^{n} \sum_{k=1}^{K} r_{ik} \, \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2.$$

Assignment and update can be written as

$$r_{ik} = \begin{cases} 1 & k = \operatorname*{argmin}_j \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2 \\ 0 & \text{otherwise} \end{cases}, \qquad \boldsymbol{\mu}_k = \frac{\sum_i r_{ik}\,\mathbf{x}_i}{\sum_i r_{ik}}$$

(argmin_j means "the j that minimizes").

Proof that J always decreases In the assignment step, with $\boldsymbol{\mu}_k$ fixed, each data point is assigned to the nearest representative point. Each term $\|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$ becomes as small as possible, so J decreases or stays the same.

In the update step, with r_{ik} fixed, taking the partial derivative of J with respect to $\boldsymbol{\mu}_k$ and setting it to 0,

$$\frac{\partial J}{\partial \boldsymbol{\mu}_k} = -2 \sum_i r_{ik}(\mathbf{x}_i - \boldsymbol{\mu}_k) = \mathbf{0} \quad \Longrightarrow \quad \boldsymbol{\mu}_k = \frac{\sum_i r_{ik}\mathbf{x}_i}{\sum_i r_{ik}},$$

so the centroid is the position that minimizes J (J is a convex quadratic function of $\boldsymbol{\mu}_k$). Therefore in this step too J decreases or stays the same.

Since $J \geq 0$ and there are only finitely many assignments (at most K^n), the algorithm always stops after a finite number of steps unless it returns to the same assignment. However, there is no guarantee that where it stops is the overall minimum of J.

Appendix E Principal component analysis and eigenvalues

E.1 The variance-covariance matrix

Let $\bar{\mathbf{x}}$ be the mean of data $\mathbf{x}_i$ with d variables. The coordinate of the data projected onto the direction of a vector $\mathbf{w}$ of length 1 ($\|\mathbf{w}\| = 1$) is $z_i = \mathbf{w}^\mathsf{T}(\mathbf{x}_i - \bar{\mathbf{x}})$, and its variance can be written

$$\frac{1}{n}\sum_{i=1}^{n} z_i^2 = \mathbf{w}^\mathsf{T}\Sigma\,\mathbf{w}, \qquad \Sigma = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^\mathsf{T}.$$

Σ is the **variance-covariance matrix**; its diagonal entries are the variances of the variables, and its off-diagonal entries are the covariances. In the example,

$$\Sigma = \begin{pmatrix} 200 & 120 \\ 120 & 80 \end{pmatrix}.$$

Setting $\mathbf{w} = (\cos\theta, \sin\theta)^\mathsf{T}$ gives $\mathbf{w}^\mathsf{T}\Sigma\mathbf{w} = 200\cos^2\theta + 240\sin\theta\cos\theta + 80\sin^2\theta$, which agrees with $V(\theta)$ in the main text.

E.2 Derivation by the method of Lagrange multipliers

The problem is to find the $\mathbf{w}$ that maximizes $\mathbf{w}^\mathsf{T}\Sigma\mathbf{w}$ under the condition $\|\mathbf{w}\| = 1$. For this, we take the partial derivatives of

$$F(\mathbf{w}, \lambda) = \mathbf{w}^\mathsf{T}\Sigma\mathbf{w} - \lambda(\mathbf{w}^\mathsf{T}\mathbf{w} - 1)$$

with respect to $\mathbf{w}$ and set them to $\mathbf{0}$. Since Σ is a symmetric matrix,

$$\nabla_{\mathbf{w}} F = 2\Sigma\mathbf{w} - 2\lambda\mathbf{w} = \mathbf{0} \quad \Longrightarrow \quad \Sigma\mathbf{w} = \lambda\mathbf{w},$$

so $\mathbf{w}$ must be an **eigenvector** of Σ. Then the variance is $\mathbf{w}^\mathsf{T}\Sigma\mathbf{w} = \lambda\,\mathbf{w}^\mathsf{T}\mathbf{w} = \lambda$, so

> **the eigenvector for the largest eigenvalue is the direction of the first principal component, and the variance in that direction is the eigenvalue itself.**

Since Σ is a symmetric matrix, eigenvectors for different eigenvalues are orthogonal to each other. This is why the second principal component in the main text was at right angles to the first.

E.3 Computation for the example

The characteristic equation $\det(\Sigma - \lambda I) = 0$ is

$$(200 - \lambda)(80 - \lambda) - 120^2 = \lambda^2 - 280\lambda + 1600 = 0,$$

so

$$\lambda = 140 \pm \sqrt{18000} \quad \Longrightarrow \quad \lambda_1 \approx 274.16, \quad \lambda_2 \approx 5.84.$$

These agree with the maximum and minimum found in the main text by combining trigonometric functions. The eigenvector for λ_1 is $\mathbf{w}_1 \approx (0.851,\ 0.526)^{\mathsf{T}}$. The total variance is the sum of the eigenvalues, that is, the trace of Σ (the sum of its diagonal entries), $200 + 80 = 280$, and the contribution ratio is $\lambda_1/(\lambda_1+\lambda_2) \approx 0.979$. The program in Appendix F.3 finds the eigenvector by the **power method**, which uses the property that repeatedly multiplying any vector by Σ and rescaling it to length 1 brings it closer to the direction of the eigenvector of the largest eigenvalue.

E.4 Correspondence with the stress tensor

The **stress tensor**, studied in earth science and structural mechanics, is also a symmetric matrix. In two dimensions, the stress tensor is

$$\begin{pmatrix} \sigma_{xx} & \tau_{xy} \\ \tau_{xy} & \sigma_{yy} \end{pmatrix}$$

(σ_{xx} and σ_{yy} are normal stresses and τ_{xy} is the shear stress); its eigenvalues are the **principal stresses**, and its eigenvectors the **principal stress axes**.

> Principal component analysis is the computation of the same "principal axes" for the variance-covariance matrix, and is **mathematically exactly the same problem** as finding the principal stresses and principal stress axes of the stress tensor.

This correspondence can be seen directly in the formulas of Sections 8.4 and 8.5 of the main text. The normal stress acting on a plane whose normal is tilted at angle θ from the x axis is

$$\sigma_n(\theta) = \sigma_{xx}\cos^2\theta + 2\tau_{xy}\sin\theta\cos\theta + \sigma_{yy}\sin^2\theta = \frac{\sigma_{xx}+\sigma_{yy}}{2} + \frac{\sigma_{xx}-\sigma_{yy}}{2}\cos 2\theta + \tau_{xy}\sin 2\theta,$$

which has the same form as $V(\theta)$ in the main text. The correspondence is "variance of mathematics $\leftrightarrow \sigma_{xx}$, variance of science $\leftrightarrow \sigma_{yy}$, covariance $\leftrightarrow \tau_{xy}$." **Mohr's stress circle** is a diagram of this formula's center $\frac{\sigma_{xx}+\sigma_{yy}}{2}$ and radius $\sqrt{\left(\frac{\sigma_{xx}-\sigma_{yy}}{2}\right)^2 + \tau_{xy}^2}$; applied to the example, it is a circle with center 140 and radius 134.16, and the two points where the circle crosses the horizontal axis, 274.16 and 5.84, correspond to the "principal stresses," that is, the two eigenvalues (the variances of the first and second principal components). That the direction of the principal axes is determined by $\tan 2\theta = \dfrac{2\tau_{xy}}{\sigma_{xx}-\sigma_{yy}}$ is also the same as $\tan 2\theta = \dfrac{240}{120} = 2$ in the main text.

Appendix F Python code (standard library only)

These programs compute the examples in the main text. They use no external libraries (such as NumPy), only the standard features of Python 3.

F.1 Logistic regression (Section 4)

```
import math

# Study time (hours) and result (1 = pass, 0 = fail)
x = [1, 2, 3, 4, 5, 6, 7, 8]
y = [0, 0, 0, 1, 0, 1, 1, 1]
n = len(x)

def sigmoid(z):
    return 1.0 / (1.0 + math.exp(-z))
```

```
w, b = 0.0, 0.0     # initial values
eta = 0.1           # learning rate

for step in range(1, 20001):
    gw = gb = 0.0
    for xi, yi in zip(x, y):
        p = sigmoid(w * xi + b)
        gw += (p - yi) * xi      # dL/dw
        gb += (p - yi)           # dL/db
    w -= eta * gw / n
    b -= eta * gb / n
    if step in (1, 10, 100, 1000, 20000):
        L = -sum(yi * math.log(sigmoid(w*xi+b))
                 + (1-yi) * math.log(1-sigmoid(w*xi+b))
                 for xi, yi in zip(x, y))
        print(f"step {step:5d}: w={w:7.4f} b={b:8.4f} Loss={L:.4f}")

print("boundary x =", round(-b / w, 3))
for xi in x:
    print(xi, "hours -> probability of passing", round(sigmoid(w * xi + b), 3))
```

Output:

```
step     1: w= 0.0875 b=  0.0000 Loss=5.1253
step    10: w= 0.1651 b= -0.1242 Loss=4.8956
step   100: w= 0.3675 b= -1.1960 Loss=3.8259
step  1000: w= 0.9971 b= -4.3722 Loss=2.5761
step 20000: w= 1.2823 b= -5.7703 Loss=2.5030
boundary x = 4.5
1 hours -> probability of passing 0.011
2 hours -> probability of passing 0.039
3 hours -> probability of passing 0.127
4 hours -> probability of passing 0.345
5 hours -> probability of passing 0.655
6 hours -> probability of passing 0.873
7 hours -> probability of passing 0.961
8 hours -> probability of passing 0.989
```

F.2 The k-means method (Section 7)

```
# Divide 2D points into K = 2 groups
points = [(1, 1), (1.5, 2), (2, 1.5), (1, 2.5),
          (5, 5), (6, 5.5), (5.5, 6.5), (6.5, 6)]
K = 2
centers = [points[0], points[1]]   # initial representative points (a deliberately bad choice)

def dist2(p, q):
    return (p[0] - q[0])**2 + (p[1] - q[1])**2

for it in range(10):
    # (1) Assignment: put each point in the group of the nearest representative point
    labels = [min(range(K), key=lambda k: dist2(p, centers[k]))
              for p in points]
    J = sum(dist2(p, centers[l]) for p, l in zip(points, labels))
    print(f"iteration {it}: centers={[(round(c[0],3), round(c[1],3)) for c in centers]}"
          f" assignment={labels} J={J:.3f}")
    # (2) Update: the centroid of each group becomes the new representative point
    new = []
```

```
    for k in range(K):
        g = [p for p, l in zip(points, labels) if l == k]
        new.append((sum(p[0] for p in g) / len(g),
                    sum(p[1] for p in g) / len(g)))
    if new == centers:
        print("converged")
        break
    centers = new
```

Output (in the assignment, 0 is group 1 and 1 is group 2):

```
iteration 0: centers=[(1, 1), (1.5, 2)] assignment=[0, 1, 1, 1, 1, 1, 1, 1] J=132.000
iteration 1: centers=[(1.0, 1.0), (3.929, 4.143)] assignment=[0, 0, 0, 0, 1, 1, 1, 1] J=30.852
iteration 2: centers=[(1.375, 1.75), (5.75, 5.75)] assignment=[0, 0, 0, 0, 1, 1, 1, 1] J=4.438
converged
```

F.3 Principal component analysis (Section 8)

The eigenvector is found by the power method (Appendix E.3).

```
import math

# Mathematics and science scores of 5 people
data = [(80, 75), (60, 65), (90, 85), (50, 60), (70, 65)]
n = len(data)

# (1) Subtract the means (centering)
m = [sum(d[j] for d in data) / n for j in range(2)]
c = [(d[0] - m[0], d[1] - m[1]) for d in data]

# (2) Variance-covariance matrix Sigma (divided by n)
S = [[sum(ci[a] * ci[b] for ci in c) / n for b in range(2)]
     for a in range(2)]
print("means", m)
print("Sigma =", S)

# (3) Find the largest eigenvalue and its eigenvector by the power method
v = [1.0, 0.0]
for _ in range(100):
    u = [S[0][0]*v[0] + S[0][1]*v[1],
         S[1][0]*v[0] + S[1][1]*v[1]]          # u = Sigma v
    norm = math.sqrt(u[0]**2 + u[1]**2)
    v = [u[0] / norm, u[1] / norm]
lam = (v[0]*(S[0][0]*v[0] + S[0][1]*v[1])
       + v[1]*(S[1][0]*v[0] + S[1][1]*v[1]))   # lambda = v^T Sigma v
trace = S[0][0] + S[1][1]
print("direction of the 1st principal component w1 =", [round(t, 4) for t in v])
print("eigenvalue lambda1 =", round(lam, 3), " contribution ratio =", round(lam / trace, 4))

# (4) 1st principal component score of each person: z = w1 . (x - mean)
for d, ci in zip(data, c):
    print(d, "-> z =", round(v[0]*ci[0] + v[1]*ci[1], 2))
```

Output:

```
means [70.0, 70.0]
Sigma = [[200.0, 120.0], [120.0, 80.0]]
direction of the 1st principal component w1 = [0.8507, 0.5257]
eigenvalue lambda1 = 274.164  contribution ratio = 0.9792
(80, 75) -> z = 11.14
```

```
(60, 65) -> z = -11.14
(90, 85) -> z = 24.9
(50, 60) -> z = -22.27
(70, 65) -> z = -2.63
```