2022/11/14
1.1 What is machine learning? 定義，なぜ確率を扱うか?
1.2 Supervised learning
1.2.1 Classification
1.2.1.1 Example: classifying Iris flowers 3class, 4features, 150samples.
1.2.1.2 Exploratory data analysis A pairwise scatter plot
1.2.1.3 Learning a classifier 手動で決定木を作る
1.2.1.4 Empirical risk minimization 自動で分類モデルを作るために

1 Introduction

1.1 What is machine learning?

A popular definition of **machine learning** or **ML**, due to Tom Mitchell [Mit97], is as follows:

> A computer program is said to learn from experience E with respect to some class of tasks T, and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Thus there are many different kinds of machine learning, depending on the nature of the tasks T we wish the system to learn, the nature of the performance measure P we use to evaluate the system, and the nature of the training signal or experience E we give it.

In this book, we will cover the most common types of ML, but from a **probabilistic perspective**. Roughly speaking, this means that we treat all unknown quantities (e.g., predictions about the future value of some quantity of interest, such as tomorrow's temperature, or the parameters of some model) as **random variables**, that are endowed with **probability distributions** which describe a weighted set of possible values the variable may have. (See Chapter 2 for a quick refresher on the basics of probability, if necessary.)

There are two main reasons we adopt a probabilistic approach. First, it is the optimal approach to **decision making under uncertainty**, as we explain in Section 5.1. Second, probabilistic modeling is the language used by most other areas of science and engineering, and thus provides a unifying framework between these fields. As Shakir Mohamed, a researcher at DeepMind, put it:[1]

> Almost all of machine learning can be viewed in probabilistic terms, making probabilistic thinking fundamental. It is, of course, not the only view. But it is through this view that we can connect what we do in machine learning to every other computational science, whether that be in stochastic optimisation, control theory, operations research, econometrics, information theory, statistical physics or bio-statistics. For this reason alone, mastery of probabilistic thinking is essential.

1.2 Supervised learning

The most common form of ML is **supervised learning**. In this problem, the task T is to learn a mapping f from inputs $\boldsymbol{x} \in \mathcal{X}$ to outputs $\boldsymbol{y} \in \mathcal{Y}$. The inputs $\boldsymbol{x}$ are also called the **features**,

1. Source: Slide 2 of https://bit.ly/3pyHyPn

(a)

(b)

(c)

Figure 1.1: Three types of Iris flowers: Setosa, Versicolor and Virginica. Used with kind permission of Dennis Kramb and SIGNA.

index	sl	sw	pl	pw	label
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
	...				
50	7.0	3.2	4.7	1.4	Versicolor
	...				
149	5.9	3.0	5.1	1.8	Virginica

Table 1.1: A subset of the Iris design matrix. The features are: sepal length, sepal width, petal length, petal width. There are 50 examples of each class.

covariates, or **predictors**; this is often a fixed-dimensional vector of numbers, such as the height and weight of a person, or the pixels in an image. In this case, $\mathcal{X} = \mathbb{R}^D$, where D is the dimensionality of the vector (i.e., the number of input features). The output $\boldsymbol{y}$ is also known as the **label**, **target**, or **response**.[2] The experience E is given in the form of a set of N input-output pairs $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^N$, known as the **training set**. (N is called the **sample size**.) The performance measure P depends on the type of output we are predicting, as we discuss below.

1.2.1 Classification

In **classification** problems, the output space is a set of C unordered and mutually exclusive labels known as **classes**, $\mathcal{Y} = \{1, 2, \ldots, C\}$. The problem of predicting the class label given an input is also called **pattern recognition**. (If there are just two classes, often denoted by $y \in \{0, 1\}$ or $y \in \{-1, +1\}$, it is called **binary classification**.)

1.2.1.1 Example: classifying Iris flowers

As an example, consider the problem of classifying Iris flowers into their 3 subspecies, Setosa, Versicolor and Virginica. Figure 1.1 shows one example of each of these classes.

2. Sometimes (e.g., in the statsmodels Python package) $\boldsymbol{x}$ are called the **exogenous variables** and $\boldsymbol{y}$ are called the **endogenous variables**.

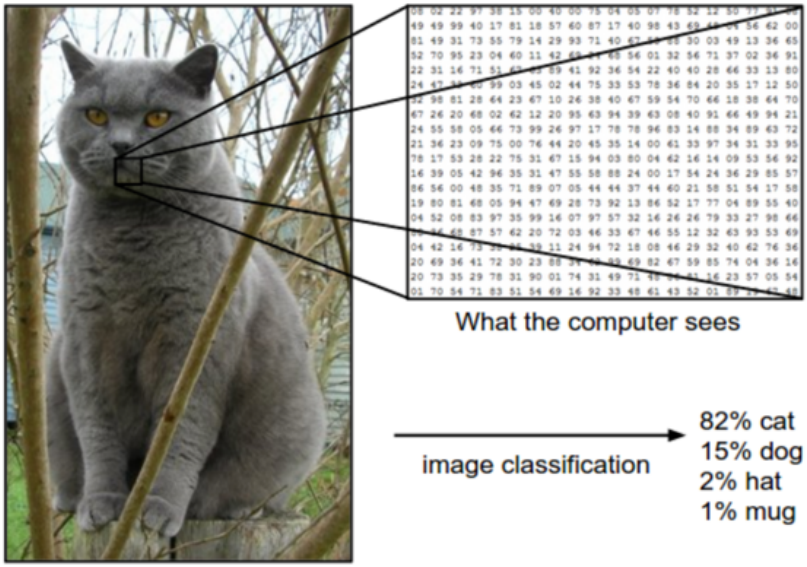


Figure 1.2: Illustration of the image classification problem. From https://cs231n.github.io/. Used with kind permission of Andrej Karpathy.

In **image classification**, the input space $\mathcal{X}$ is the set of images, which is a very high-dimensional space: for a color image with $C = 3$ channels (e.g., RGB) and $D_1 \times D_2$ pixels, we have $\mathcal{X} = \mathbb{R}^D$, where $D = C \times D_1 \times D_2$. (In practice we represent each pixel intensity with an integer, typically from the range $\{0, 1, \ldots, 255\}$, but we assume real valued inputs for notational simplicity.) Learning a mapping $f : \mathcal{X} \to \mathcal{Y}$ from images to labels is quite challenging, as illustrated in Figure 1.2. However, it can be tackled using certain kinds of functions, such as a **convolutional neural network** or **CNN**, which we discuss in Section 14.1.

Fortunately for us, some botanists have already identified 4 simple, but highly informative, numeric features — sepal length, sepal width, petal length, petal width — which can be used to distinguish the three kinds of Iris flowers. In this section, we will use this much lower-dimensional input space, $\mathcal{X} = \mathbb{R}^4$, for simplicity. The **Iris dataset** is a collection of 150 labeled examples of Iris flowers, 50 of each type, described by these 4 features. It is widely used as an example, because it is small and simple to understand. (We will discuss larger and more complex datasets later in the book.)

When we have small datasets of features, it is common to store them in an $N \times D$ matrix, in which each row represents an example, and each column represents a feature. This is known as a **design matrix**; see Table 1.1 for an example.[3]

The Iris dataset is an example of **tabular data**. When the inputs are of variable size (e.g., sequences of words, or social networks), rather than fixed-length vectors, the data is usually stored

3. This particular design matrix has $N = 150$ rows and $D = 4$ columns, and hence has a **tall and skinny** shape, since $N \gg D$. By contrast, some datasets (e.g., genomics) have more features than examples, $D \gg N$; their design matrices are **short and fat**. The term "**big data**" usually means that N is large, whereas the term "**wide data**" means that D is large (relative to N).

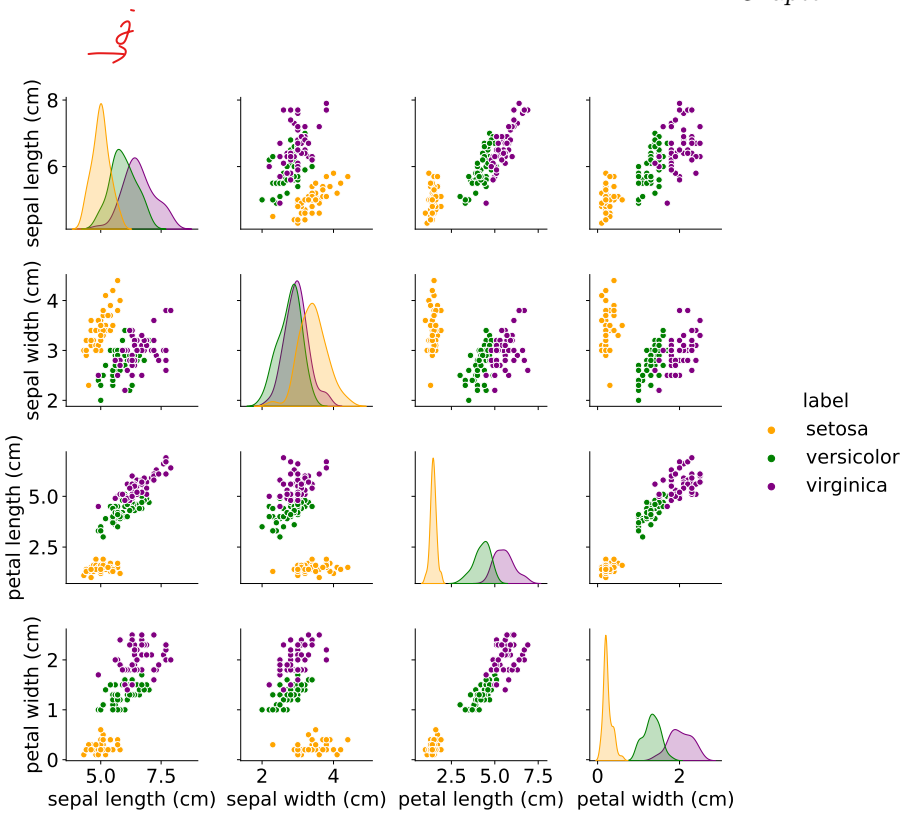


Figure 1.3: Visualization of the Iris data as a pairwise scatter plot. On the diagonal we plot the marginal distribution of each feature for each class. The off-diagonals contain scatterplots of all possible pairs of features. Generated by iris_plot.ipynb

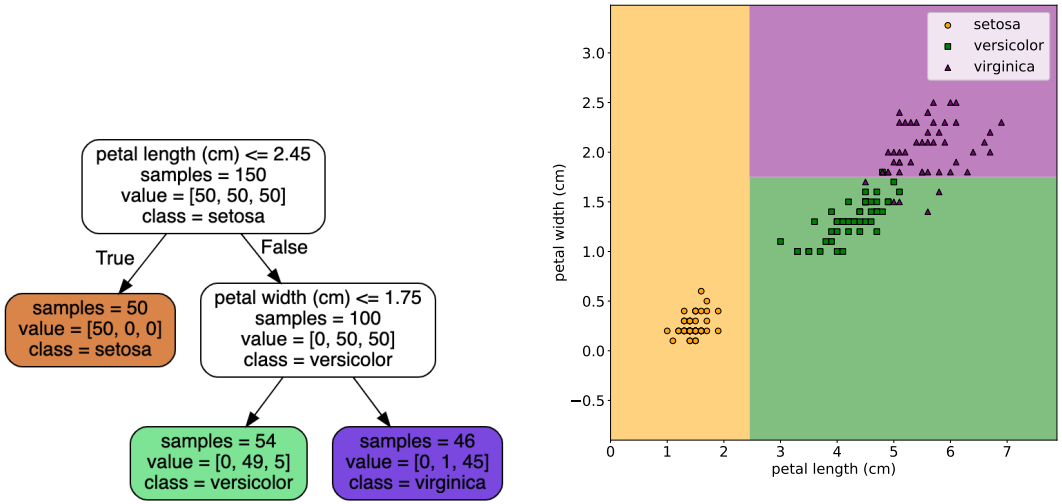
in some other format rather than in a design matrix. However, such data is often converted to a fixed-sized feature representation (a process known as **featurization**), thus implicitly creating a design matrix for further processing. We give an example of this in Section 1.5.4.1, where we discuss the "bag of words" representation for sequence data.

1.2.1.2 Exploratory data analysis

Before tackling a problem with ML, it is usually a good idea to perform **exploratory data analysis**, to see if there are any obvious patterns (which might give hints on what method to choose), or any obvious problems with the data (e.g., label noise or outliers).

For tabular data with a small number of features, it is common to make a **pair plot**, in which panel (i, j) shows a scatter plot of variables i and j, and the diagonal entries (i, i) show the marginal density of variable i; all plots are optionally color coded by class label — see Figure 1.3 for an example.

For higher-dimensional data, it is common to first perform **dimensionality reduction**, and then



(a) (b)

Figure 1.4: Example of a decision tree of depth 2 applied to the Iris data, using just the petal length and petal width features. Leaf nodes are color coded according to the predicted class. The number of training samples that pass from the root to a node is shown inside each box; we show how many values of each class fall into this node. This vector of counts can be normalized to get a distribution over class labels for each node. We can then pick the majority class. Adapted from Figures 6.1 and 6.2 of [Gér19]. Generated by iris_ dtree.ipynb.

to visualize the data in 2d or 3d. We discuss methods for dimensionality reduction in Chapter 20.

1.2.1.3 Learning a classifier

From Figure 1.3, we can see that the Setosa class is easy to distinguish from the other two classes. For example, suppose we create the following **decision rule**:

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \begin{cases} \text{Setosa if petal length} < 2.45 \\ \text{Versicolor or Virginica otherwise} \end{cases} \tag{1.1}$$

This is a very simple example of a classifier, in which we have partitioned the input space into two regions, defined by the one-dimensional (1d) **decision boundary** at $x_{\text{petal length}} = 2.45$. Points lying to the left of this boundary are classified as Setosa; points to the right are either Versicolor or Virginica.

We see that this rule perfectly classifies the Setosa examples, but not the Virginica and Versicolor ones. To improve performance, we can recursively partition the space, by splitting regions in which the classifier makes errors. For example, we can add another decision rule, to be applied to inputs that fail the first test, to check if the petal width is below 1.75cm (in which case we predict Versicolor) or above (in which case we predict Virginica). We can arrange these nested rules into a tree structure,

		Estimate		
		Setosa	Versicolor	Virginica
Truth	Setosa	0	1	1
	Versicolor	1	0	1
	Virginica	10	10	0

Table 1.2: Hypothetical asymmetric loss matrix for Iris classification.

called a **decision tree**, as shown in Figure 1.4a This induces the 2d **decision surface** shown in Figure 1.4b.

We can represent the tree by storing, for each internal node, the feature index that is used, as well as the corresponding threshold value. We denote all these **parameters** by $\boldsymbol{\theta}$. We discuss how to learn these parameters in Section 18.1.

1.2.1.4 Empirical risk minimization

The goal of supervised learning is to automatically come up with classification models such as the one shown in Figure 1.4a, so as to reliably predict the labels for any given input. A common way to measure performance on this task is in terms of the **misclassification rate** on the training set:

$$\mathcal{L}(\boldsymbol{\theta}) \triangleq \frac{1}{N}\sum_{n=1}^{N} \mathbb{I}\left(y_n \neq f(\boldsymbol{x}_n;\boldsymbol{\theta})\right) \tag{1.2}$$

where $\mathbb{I}(e)$ is the binary **indicator function**, which returns 1 iff (if and only if) the condition e is true, and returns 0 otherwise, i.e.,

$$\mathbb{I}(e) = \begin{cases} 1 & \text{if } e \text{ is true} \\ 0 & \text{if } e \text{ is false} \end{cases} \tag{1.3}$$

This assumes all errors are equal. However it may be the case that some errors are more costly than others. For example, suppose we are foraging in the wilderness and we find some Iris flowers. Furthermore, suppose that Setosa and Versicolor are tasty, but Virginica is poisonous. In this case, we might use the asymmetric **loss function** $\ell(y, \hat{y})$ shown in Table 1.2.

We can then define **empirical risk** to be the average loss of the predictor on the training set:

$$\mathcal{L}(\boldsymbol{\theta}) \triangleq \frac{1}{N}\sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n;\boldsymbol{\theta})) \tag{1.4}$$

We see that the misclassification rate Equation (1.2) is equal to the empirical risk when we use **zero-one loss** for comparing the true label with the prediction:

$$\ell_{01}(y, \hat{y}) = \mathbb{I}(y \neq \hat{y}) \tag{1.5}$$

See Section 5.1 for more details.

One way to define the problem of **model fitting** or **training** is to find a setting of the parameters that minimizes the empirical risk on the training set:

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmin}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \operatorname*{argmin}_{\boldsymbol{\theta}} \frac{1}{N} \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n; \boldsymbol{\theta})) \tag{1.6}$$

This is called **empirical risk minimization**.

However, our true goal is to minimize the expected loss on *future data* that we have not yet seen. That is, we want to **generalize**, rather than just do well on the training set. We discuss this important point in Section 1.2.3.

1.2.1.5 Uncertainty

> [We must avoid] false confidence bred from an ignorance of the probabilistic nature of the world, from a desire to see black and white where we should rightly see gray. — Immanuel Kant, as paraphrased by Maria Konnikova [Kon20].

In many cases, we will not be able to perfectly predict the exact output given the input, due to lack of knowledge of the input-output mapping (this is called **epistemic uncertainty** or **model uncertainty**), and/or due to intrinsic (irreducible) stochasticity in the mapping (this is called **aleatoric uncertainty** or **data uncertainty**).

Representing uncertainty in our prediction can be important for various applications. For example, let us return to our poisonous flower example, whose loss matrix is shown in Table 1.2. If we predict the flower is Virginica with high probability, then we should not eat the flower. Alternatively, we may be able to perform an **information gathering action**, such as performing a diagnostic test, to reduce our uncertainty. For more information about how to make optimal decisions in the presence of uncertainty, see Section 5.1.

We can capture our uncertainty using the following **conditional probability distribution**:

$$p(y = c|\boldsymbol{x}; \boldsymbol{\theta}) = f_c(\boldsymbol{x}; \boldsymbol{\theta}) \tag{1.7}$$

where $f : \mathcal{X} \to [0, 1]^C$ maps inputs to a probability distribution over the C possible output labels. Since $f_c(\boldsymbol{x}; \boldsymbol{\theta})$ returns the probability of class label c, we require $0 \leq f_c \leq 1$ for each c, and $\sum_{c=1}^{C} f_c = 1$. To avoid this restriction, it is common to instead require the model to return unnormalized log-probabilities. We can then convert these to probabilities using the **softmax function**, which is defined as follows

$$\text{softmax}(\boldsymbol{a}) \triangleq \left[\frac{e^{a_1}}{\sum_{c'=1}^{C} e^{a_{c'}}}, \ldots, \frac{e^{a_C}}{\sum_{c'=1}^{C} e^{a_{c'}}} \right] \tag{1.8}$$

This maps $\mathbb{R}^C$ to $[0, 1]^C$, and satisfies the constraints that $0 \leq \text{softmax}(\boldsymbol{a})_c \leq 1$ and $\sum_{c=1}^{C} \text{softmax}(\boldsymbol{a})_c = 1$. The inputs to the softmax, $\boldsymbol{a} = f(\boldsymbol{x}; \boldsymbol{\theta})$, are called **logits**. See Section 2.5.2 for details. We thus define the overall model as follows:

$$p(y = c|\boldsymbol{x}; \boldsymbol{\theta}) = \text{softmax}_c(f(\boldsymbol{x}; \boldsymbol{\theta})) \tag{1.9}$$