Appendix — Technical background

In this chapter we provide technical background on some of the mathematical concepts used in this book. We focus on two topics. The first is properties of random variables and how random variables can be used to model populations. The second is about the fundamentals of machine learning and how we compute classifiers from data.

Random variables and conditional probabilities

Throughout this book, it's fine to think of all that all probabilities as counting things in a finite population P. We can think of P as a finite set of things or individuals. Probabilities therefore correspond to simple counts of things in the population. We can interpret the probability $\mathbb{P}\{E\}$ as the fraction of the population contained in the set E, called *event* when it appears inside a probability statement. The *conditional probability* statement $\mathbb{P}\{E \mid C\}$ instructs us to restrict the population to the set C and to compute probabilities within this subpopulation C as if it were the whole population.

We generally denote random variables with capital letters U, V, W, X, Y, Z. Random variables are functions that assign values to elements of the probability space. A statement like $\mathbb{P}\{U = u\}$ corresponds to the probability that the random variable assumes the value u. More formally, this is a shorthand for the probability of the event $\{i \in P \colon U(i) = u\}$.

Two random variables U and V are *independent* if for all values u and v that the random variables might assume, we have:

$$\mathbb{P}\{U = u, V = v\} = \mathbb{P}\{U = u\} \cdot \mathbb{P}\{V = v\}\,.$$

A calculation reveals that independent can be written in terms of conditional probabilities as:

$$\mathbb{P}\{U = u \mid V = v\} = \mathbb{P}\{U = u\}\,,$$

This equivalent formulation has an intuitive interpretation. It says

that observing that V assumed value v gives us no hint about the probability of observing the event $U = u$. It's this second characterization that we used in Chapter 2 when we argued that the independence criterion for binary classification (accept/reject) implies equal acceptance rates in all groups.

The notion of independent random variables, extends to *conditional independence*. Two random variables U and V are *conditionally independent* given a third random variable W if for all values u, v and w that the random variables might assume, we have:

$$\mathbb{P}\{U = u, V = v \mid W = w\} = \mathbb{P}\{U = u \mid W = w\} \cdot \mathbb{P}\{V = v \mid W = w\}.$$

Sample and population

We often think of random variables (X, Y) as modeling a population of instances of a classification problem. In almost all decision making scenarios, however, we do not have access to the entire population of instances that we will encounter. Instead, we only have a sample of instances from this population. To give an example, consider a population consisting of all currently eligible voters in the United States and some of their features, such as, age, income, state of residence etc. An unbiased random sample would from this population would correspond to a subset of voters so that each voter has an equal probability of appearing the sample.

Sampling is a difficult problem with numerous pitfalls that can strongly affect the performance of statistical estimators and the validity of what we learn from data. Even defining a good population for the problem we're trying to solve is often tricky.

The theory of machine learning largely ignores these issues. The focus is instead on the challenges that remain even if we have a well-defined population and an unbiased sample from it.

Supervised learning is the prevalent method for constructing classifiers from data. The essential idea is very simple. We assume we have labeled data, also called *training examples*, of the form $(x_1, y_1), ..., (x_n, y_n)$, where each *example* is a pair (x_i, y_i) of an *instance* x_i and a corresponding *label* y_i.

Building models from data

We'll now dive into the main method that machine learning practitioners use to construct classifiers from a sample of data points. We assume we have a sample $S = ((x_1, y_1), ..., (x_n, y_n))$ of labeled data points drawn independently from a population (X, Y).

Fix a loss function $\ell\colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. , the *empirical risk* of a classifier $f\colon \mathcal{X} \to \mathcal{Y}$ with respect to the sample S is defined as

$$R_S(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i) .$$

Empirical risk minimization is the optimization problem of finding a classifier in a given function family that minimizes the empmirical risk:

$$\min_{f \in \mathcal{F}} R_S(f)$$

Introducing empirical risk minimization directly leads to three important questions that we discuss next.

- **Representation:** What is the class of functions $\mathcal{F}$ we should choose?
- **Optimization:** How can we efficiently solve the resulting optimization problem? We will see a number of different optimization methods that under certain circumstances find either a global or local minimum of the empirical risk objective.
- **Generalization:** Will the performance of classifier transfer gracefully from seen training examples to unseen instances of our problem? The most common way to measure generalization performance is to consider the difference between risk and empirical risk $R(f) - R_S(f)$. We will see several mathematical frameworks for reasoning about the gap between risk and empirical risk.

These three questions are intertwined. Our choice of representation influences both the difficulty of optimization and our generalization performance. Improvements in optimization may not help, or could even hurt, generalization.

But there are also important differences between the three. If we can show that optimization works, it will typically be independent of how the sample was chosen. To reason about generalization, however, we will need assumptions about the data generating process. The most common one is that the samples $(x_1, y_1), ..., (x_n, y_n)$ were drawn independently and identically (i.i.d.) from the population (X, Y).

There are also aspects of the problem that don't neatly fall into any of these categories. The choice of the loss function, for example, affects all of the three questions above.

Let's start with a good example to illustrate these questions.

Example: Perceptron

The New York Times wrote in 1958 that the Perceptron algorithm[1]

[1] Frank Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review*, 1958, 65–386.

was:

> the embryo of an electronic computer that (the Navy) expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

So, what is this algorithm? Let's assume we're in a binary classification problem with labels in $\{-1, 1\}$. The Perceptron algorithm aims to find a linear separator $w \in \mathbb{R}^d$ so that $y_i \langle w, x_i \rangle \geq 1$. In other words, the linear function $f(x) = \langle w, x \rangle$ agrees in sign with the labels on all training instances (x_i, y_i).

The algorithm goes about it in the following way:

- At each step $t = 1, 2, ...$, select a random index $i \in \{1, ..., n\}$ and perform the following update step:
  - If $y_i \langle w_t, x_i \rangle < 1$, put
  $$w_{t+1} = w_t(1 - \gamma) + \eta y_i x_i$$
  - Otherwise put $w_{t+1} = w_t(1 - \gamma)$.

Can we represent the Perceptron algorithm as an instance of empirical risk minimization? The answer is that we can and it is instructive to see how.

First, it's clear from the description that we're looking for a linear separator. Hence, our function class is the set of linear functions $f(x) = \langle w, x \rangle$, where $w \in \mathbb{R}^d$. We will sometimes call the vector w the *weight vector* or vector of *model parameters*.

An optimization method that picks a random example at each step and makes an improvement to the model parameters is the popular stochastic gradient method specified by the updated rule:

$$w_{t+1} = w_t - \eta \nabla_{w_t} \ell(f(x_i), y_i)$$

Here, $\nabla \ell(f(x), y_i)$ is the gradient of the loss function with respect to the model parameters w_t on a randomly chosen example (x_i, y_i). We will typically drop the vector w_t from the subscript of the gradient when it's clear from the context. The scalar $\eta > 0$ is a step sise parameter that we will discuss more carefully later. For now, think of it as a small constant.

Consider the loss function

$$\ell(y, \langle w, x \rangle) = \max(1 - y\langle w, x \rangle,\ 0)\,.$$

This loss function is called *hinge loss*. Its gradient is $-yx$ when $y\langle w, x \rangle < 1$ and 0 when $y\langle w, x \rangle > 1$. Note that the gradient is not defined at $y\langle w, x \rangle = 1$.[2]

[2] The loss function is not differentiable everywhere. This is why technically speaking the stochastic gradient method operates with what is called a *subgradient*.

We can see that the hinge loss gives us part of the update rule in the Perceptron algorithm. The other part comes from adding a weight penalty $\frac{\gamma}{2}\|w\|^2$ to the loss function that discourages the weights from growing out of bounds. This weight penality is called ℓ_2-*regularization*, *weight decay*, or *Tikhonov regularization* depending on which field you work in. The purpose of regularization is to promote generalization. We will therefore return to regularization in detail when we discuss generalization in more depth.

Putting the two loss functions together, we get the ℓ_2-regularized empirical risk minimization problem for the hinge loss:

$$\frac{1}{n}\sum_{i=1}^{n}\max(1 - y_i\langle w, x_i\rangle,\ 0) + \frac{\gamma}{2}\|w\|_2^2$$

The Perceptron algorithm corresponds to solving this empirical risk objective with the stochastic gradient method. The optimization problem is also known as *support vector machine* and we will return to it later on.

Representation

Optimization

Surrogate losses

If our goal is to minimize the accuracy of a predictor, why don't we directly solve empirical risk minimization with respect to the *zero-one loss* $\ell(y, z) = \mathbb{1}\{y \neq z\}$ that gives us penalty 1 if our label is incorrect, and penalty 0 if our predicted label z matches the true label y?

The reason is that the zero one loss is hard to optimize. The gradients of the zero-one loss are zero everywhere they're defined, and so we can't expect the gradient-based methods to directly optimize the zero-one loss.

This is why there are numerous loss functions that approximate the zero-one loss in different ways.

- The *squared loss* is given by $\frac{1}{2}(y - z)^2$. Linear least squares regression corresponds to empirical risk minimization with the squared loss.
- The *hinge loss* is $\max\{1 - yz, 0\}$ and *support vector machine* refers to empirical risk minimization with the hinge loss and ℓ_2-regularization.
- The logistic loss is $-\log(\sigma(z))$ when $y = 1$ and $-\log(1 - \sigma(z))$ when $y = -1$, where $\sigma(z) = 1/(1 + \exp(-z))$ is the logistic function. *Logistic regression* corresponds to empirical risk minimization with the logistic loss.

Sometimes we can theoretically relate empirical risk minimization under a surrogate loss to the zero-one loss. In general, however, these loss functions are used heuristically and performance is evaluated by trial-and-error.

Generalization

Bibliographic notes and further reading

Rosenblatt, Frank. "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain." *Psychological Review*, 1958, 65–386.