CHAPTER 1

MATHEMATICAL PRELIMINARIES

Swati Gupta and Jad Salem

Georgia Institute of Technology, Atlanta, Georgia
Last updated: September 30, 2022

This chapter is a work in progress and has not been subjected to the usual scrutiny reserved for formal publications. If you find errors in this draft, please let the authors know. We would be happy to acknowledge you in the final version.

1.1 Set theory

Sets are often considered the most fundamental of mathematical objects — the building blocks of mathematics, if you will. While a precise definition of a set can be found in most set theory textbooks, the following loose definition will suffice for the purposes of this course.

Definition 1: set

A *set* is an unordered collection of distinct objects.

The word *set* is used colloquially in the English language, which may give the reader an intuitive idea of what a set is. For instance, your local home goods store may sell a set of cookware, consisting of a skillet, a sauce pan, and a stock pot. There is *no inherent order* on the objects (i.e., this is the same as a set containing a skillet, a stock pot, and a sauce pan), and the three objects are *distinct*.

A set containing the objects a, b, c is typically denoted as $\{a, b, c\}$, and the objects are referred to as *elements*. The notation $a \in S$ means that the element a is in the set S.

EXAMPLE 1.1

Consider the set $S = \{0, 2, 4, 6\}$. Since sets are composed of *unordered* and *distinct* elements, the sets

$$\{0, 2, 4, 6\},\ \{0, 4, 2, 6\},\ \{6, 2, 4, 0, 2\}$$

are all equal, and they all have four elements! We can express that 2 is an element of S by writing "$2 \in S$," and we can express that 5 is not an element of S by writing "$5 \notin S$."

Certain sets that are commonly discussed have special notation. For instance, the set of integers $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$ is denoted by $\mathbb{Z}$, which is short for the German word for number. See Table 1.1 for a list of commonly used sets.

Table 1.1 Common sets

$\mathbb{N}$	the set of natural numbers $\{0, 1, 2, \ldots\}$
$\mathbb{Z}$	the set of integers
$\mathbb{Q}$	the set of rational numbers (fractions)
$\mathbb{R}$	the set of real numbers (like $2, -3.1, \sqrt{2}, \pi$)
$\mathbb{Z}_+$	the set of positive integers $\{1, 2, 3, \ldots\}$
$\mathbb{R}_+$	the set of positive real numbers
$\mathbb{R}^n$	the set of n-tuples of real numbers
$[n]$	the set of positive integers up to n: $\{1, 2, \ldots, n\}$

You may have noticed that certain sets in Table 1.1 are "bigger" than other sets, in the sense of containment. For example, any element of $\mathbb{Z}$ is also an element of $\mathbb{R}$ (any integer is a real number). We denote this relationship by saying that $\mathbb{Z}$ is a subset of $\mathbb{R}$.

Definition 2: subset

Let A, B be sets. We say that A is a *subset* of B, and write $A \subseteq B$, if every element of A is also an element of B.

EXAMPLE 1.2

Since every natural number is rational, we can say that $\mathbb{N} \subseteq \mathbb{Q}$. On the other hand, since $(0, 0) \notin \mathbb{R}^3$, it follows that $\mathbb{R}^2$ is not a subset of $\mathbb{R}^3$.

There are several operations that can be used to create new sets out of old sets. For example, suppose we have two sets A and B. The *union* of A and B, denoted $A \cup B$, is the set of all element in at least one of the two sets. Similarly, the intersection of A and B, denoted $A \cap B$, is the set of common elements. We can rephrase these definitions as follows:

$$e \in A \cup B \iff e \in A \textbf{ or } e \in B;$$
$$e \in A \cap B \iff e \in A \textbf{ and } e \in B.$$

Let's take a concrete example. Let $A = \{1, 2, 3\}$ and $B = \{3, 4\}$. Then $A \cup B = \{1, 2, 3, 4\}$, and $A \cap B = \{3\}$. Notice that in this example, $A \cap B \subseteq A \subseteq A \cup B$. In fact, this phenomenon is always true — can you see why?

Sets are often described by qualities of their elements. For instance, one can talk about the set of *even* natural numbers, or the set of *prime* integers. When a set is defined in such a way, the set is often denoted using *set-builder notation*. This is done by specifying the underlying set from which elements are being considered, and listing the defining qualities of the elements. For instance, the set of prime integers can be written as $\{n \in \mathbb{Z} : n \text{ is prime}\}$.

Maxima and suprema of sets. We will often want to know the largest value of a set (e.g., the largest regret among all problem instances). We call the largest number in set the *maximum*. More precisely:

Definition 3: maximum

Let $S \subseteq \mathbb{R}$ be a subset of the real numbers. We say that a real number M is the *maximum of* S, and write $M = \max S$, if

1. $M \in S$, and
2. for every $x \in S$, $x \leq M$.

EXAMPLE 1.3 maxima of finite sets

Convince yourself of the following: $\max\{3, 2, 1\} = 3$ and $\max\{n \in \mathbb{N} : n \leq 7\} = 7$.

Sometimes qualities defining the set in question are placed underneath the "max" symbol. For instance, if we wanted to notate the largest prime integer less than 10, the following are two equivalent ways to do so:

$$\max\{n \in \mathbb{Z} : n \text{ is prime and } n \leq 10\} = \max_{\substack{n \leq 10 \\ n \text{ prime}}} n = 7.$$

Some sets, however, do not have a maximum. For instance, any set which is unbounded from above (such as $\mathbb{Z}, \mathbb{R}, \mathbb{N}$) does not have a maximum. Even bounded sets may not have a maximum, such as the open interval $I = (0, 1)$, i.e., all numbers between 0 and 1 (not including 0 and 1). One might be tempted to say that 1 is the maximum of I, but this is incorrect: since $1 \notin I$, condition 1 of Definition 3 is not satisfied. However, it is accurate to say that 1 is an *upper bound* for I, since 1 is bigger than any element of I. In fact, since there are no smaller upper bounds for I, we can say that 1 is the *least upper bound*, or *supremum*, of I.

Definition 4: supremum

Let $S \subseteq \mathbb{R}$ be a subset of real numbers. We say that s is the *supremum of* S, and write $s = \sup S$, if

1. s is an upper bound for S (i.e., for every $x \in S$, $x \leq s$), and
2. for every upper bound s' of S, $s \leq s'$, i.e., s is the smallest possible upper bound for S.

The main difference between maxima and suprema is that a maximum is an element of the set, whereas a supremum need not be. Importantly, this means that *any nonempty set which is bounded above has a supremum*. As discussed above, $\sup(0,1) = 1$, but $(0,1)$ has no maximum. However, the closed interval $[0,1]$ contains its supremum:

$$\sup[0,1] = \max[0,1] = 1.$$

In fact, this phenomenon is always true:

Proposition 1: maxima are suprema

Let $S \subseteq \mathbb{R}$ be a set with a maximum $M = \max S$. Then $M = \sup S$ as well.

As with maxima, the conditions of the set in question are often listed under the "sup" symbol.

EXAMPLE 1.4 suprema conventions

The set $\left\{1 - \frac{1}{n} : n \in \mathbb{Z}_+\right\}$ has no maximum, but it has an supremum:

$$\sup\left\{1 - \frac{1}{n} : n \in \mathbb{Z}_+\right\} = \sup_{n \in \mathbb{Z}_+}\left(1 - \frac{1}{n}\right) = 1.$$

Check that there is no number less than 1 that upper bounds every element of the example set.

Finally, we would like to note that all of these ideas can be applied for minima as well. We can extend the concept of a minimum to a *greatest lower bound*, or *infimum*, which exists for any set which is nonempty and bounded from below. The infimum of a set S is denoted by $\inf S$, and if a set S has a minimum, then $\min S = \inf S$.

EXAMPLE 1.5 infima conventions

The set $\left\{\frac{1}{n} : n \in \mathbb{Z}_+\right\}$ has no minimum, but it has an infimum:

$$\inf\left\{\frac{1}{n} : n \in \mathbb{Z}_+\right\} = \inf_{n \in \mathbb{Z}_+} \frac{1}{n} = 0.$$

EXERCISES

1.1.1 Order the sets $\mathbb{R}$, $\mathbb{Z}$, $\mathbb{Q}$, and $\{1, 3, 5\}$ by subset containment.

1.1.2 Give an example of a set with a minimum but no maximum. Give an example of a set with a maximum but no minimum.

1.1.3 Give an example of a nonempty set without a maximum. Does your set have a supremum?

1.1.4 Suppose A and B are sets with maxima. Show that $\max(A \cup B) = \max\{\max A, \max B\}$.

1.1.5 Let A, B be finite sets. Prove that $|A \cup B| \leq |A| + |B|$. This inequality is an example of a *union bound*.

1.2 Big-Oh analysis

This course focuses on the implementation and analysis of algorithms, and so we must ask: how does one measure the performance of an algorithm? There are many aspects of algorithms that are of interest, such as running time, space requirements, number of queries to a helper algorithm, how far an algorithm is from some benchmark, and more.

EXAMPLE 1.6 linear search

You are tasked with designing an algorithm which takes as input a list of length n, and outputs whether or not "3" is in the list. Your algorithm iterates through the list, and returns "no" if "3" is not found. If "3" is found, then the algorithm terminates, returning "yes." In this case, the *worst-case* number of comparisons that the algorithm makes is n, even though the actual number of comparisons could be as few as 1.

As seen in Example 1.6, these performance quantities are functions of the input. In machine learning, the input often includes some time horizon T, which is the number of decisions the algorithm will make over time. The performance measure that is typically of concern in such settings is called *regret*, which measures deviation from some "optimal" benchmark (see Section 2.3 for more details). Let us introduce the big-Oh notation with the example of running time of an algorithm.

Suppose we have two algorithms, Alg_1 and Alg_2, for the same problem with input size T. Alg_1 has a running time of $T^{1/2}$, and Alg_2 has a running time of $100T^{1/3}$. If the goal is to minimize running time (or runtime), which algorithm should be chosen?

Due to the large coefficient in the runtime of Alg_2, the runtime of Alg_1 is lower than the runtime of Alg_2 for *small* values of T. For *large* values of T, this trend reverses, and the runtime of Alg_1 becomes higher than the runtime of Alg_2. The question now becomes: should we care more about small values of T, or large values of T? Typically, one cares more about the runtime for large inputs, because (a) applications often involve big data, and (b) small instances can often be solved quickly even by inefficient algorithms. For these reasons, one would typically choose Alg_2 over Alg_1.

Let us now be more precise about how to compare functions such as $T^{1/2}$ and $100T^{1/3}$. The typical way to do so is with *big-oh analysis*.

Definition 5: big-oh

Let $f, g : \mathbb{N} \to \mathbb{R}$ be two functions on the natural numbers. We say that f is big-oh of g, and write $f \in \mathcal{O}(g)$, if there exist $c > 0$ and $n_0 \in \mathbb{N}$ such that $|f(n)| \leq c|g(n)|$ for all $n \geq n_0$.

Stated differently, $f \in \mathcal{O}(g)$ means that for large enough input, f is smaller than some constant multiple of g. To put this in the context of runtime, suppose Alg_3 has a runtime of $f(T)$ and Alg_4 has a runtime of $g(T)$. If we know that $f \in \mathcal{O}(g)$ and $g \notin \mathcal{O}(f)$, then Alg_3 would be preferable (in the big-oh sense). Let's take an example to make sense of Definition 5.

EXAMPLE 1.7

Suppose we have algorithms Alg_3 and Alg_4 with runtimes of $2n^2 + n + 1$ and n^2, respectively. Notice that $n^2 \in \mathcal{O}(2n^2 + n + 1)$, which can be seen by setting $n_0 = 0$ and $c = 1$ in Definition 5. Conversely, observe that for $n \geq 1$,

$$2n^2 + n + 1 \leq 2n^2 + 2n + 2 \leq 2n^2 + 2n^2 + 2n^2 \leq 6n^2 \, .$$

It follows that $2n^2 + n + 1 \in \mathcal{O}(n^2)$ by setting $n_0 = 1$ and $c = 6$ in Definition 5. Notice that big-oh essentially "hides" constant factors.

Now that we have a technical way to compare two algorithms, we should discuss how to check if $f \in \mathcal{O}(g)$. Of course, one could use Definition 5 and try to find appropriate constants c and n_0. This, however, can be quite tedious. Instead, the following proposition can be used, in most cases,[1] to determine if $f \in \mathcal{O}(g)$ more easily.

Proposition 2: criteria for big-oh comparison

Let $f, g : \mathbb{N} \to \mathbb{R}$.

- If $\lim_{n\to\infty} \frac{|f(n)|}{|g(n)|} = 0$, then $f \in \mathcal{O}(g)$ and $g \notin \mathcal{O}(f)$.
- If $\lim_{n\to\infty} \frac{|f(n)|}{|g(n)|} = \infty$, then $f \notin \mathcal{O}(g)$ and $g \in \mathcal{O}(f)$.
- If $\lim_{n\to\infty} \frac{|f(n)|}{|g(n)|} = c$, for some constant $c > 0$, then $f \in \mathcal{O}(g)$ and $g \in \mathcal{O}(f)$.

These criteria simplify our calculations greatly. For example, let us return to Alg_1 and Alg_2.

EXAMPLE 1.8

[1]For some functions f, g, the limit described in Proposition 2 diverges, and the proposition cannot be applied. That said, functions describing the performance of an algorithm are typically well-behaved, and the limit will typically exist.

Recall that Alg_1 has a runtime of $T^{1/2}$, and Alg_2 has a runtime of $100T^{1/3}$. Taking the limit of the quotient, we get

$$\lim_{T\to\infty} \frac{|T^{1/2}|}{|100T^{1/3}|} = \lim_{T\to\infty} \frac{T^{1/6}}{100} = \infty \,.$$

By Proposition 2, it follows that $100T^{1/3} \in \mathcal{O}(T^{1/2})$, and $T^{1/2} \notin \mathcal{O}(100T^{1/3})$. This verifies our previous assertion that Alg_2 is better, according to big-oh analysis.

EXAMPLE 1.9

Suppose one algorithm has runtime $n!$, and another algorithm has runtime n^n. Which algorithm is faster, according to big-oh analysis? To answer this question, we will determine the value of $\lim_{n\to\infty} \frac{n!}{n^n}$. To do so, observe that

$$\lim_{n\to\infty} \frac{n!}{n^n} \cdot \frac{(n-1)^{n-1}}{(n-1)!} = \lim_{n\to\infty} \left(1 - \frac{1}{n}\right)^{n-1} = \frac{1}{e} \,,$$

and so by the ratio test, $\lim_{n\to\infty} \frac{n!}{n^n} = 0$. By Proposition 2, it follows that $n! \in \mathcal{O}(n^n)$ and $n^n \notin \mathcal{O}(n!)$. This tells us that the algorithm with runtime $n!$ is better, according to big-oh analysis.

EXERCISES

1.2.1 Suppose you have the option of running three algorithms, Alg_1, Alg_2, Alg_3, with running times $\mathcal{O}(n \ln n)$, $\mathcal{O}(n^3)$, $300n^2$, respectively. Which algorithm would you select (i.e., which one is the fastest according to big-oh analysis) to run on input instances with arbitrary sizes n?

1.2.2 Show that $\log_{b_1}(n) \in \mathcal{O}\big(\log_{b_2}(n)\big)$ for any $b_1, b_2 > 1$.

1.2.3 Show that if $f(n) \in \mathcal{O}(g(n))$, then $\lambda f(n) \in \mathcal{O}(g(n))$ as well, for any $\lambda \in \mathbb{R}$.

1.2.4 Let $f, g, h : \mathbb{N} \to \mathbb{R}$. Show that if $f \in \mathcal{O}(h)$ and $g \in \mathcal{O}(h)$, then $f + g \in \mathcal{O}(h)$.

1.3 Probability

Suppose we roll a fair, 6-sided die (i.e., a die for which each outcome $1, \ldots, 6$ is equally likely). Then the probability of rolling a "3" is $\frac{1}{6}$. To put this statement into standard mathematical notation, we could let X represent the outcome of rolling the die, and say:

$$\mathbb{P}(\{X = 3\}) = \frac{1}{6} \,.$$

In this example, we call $\Omega = \{1, \ldots, 6\}$ the *outcome space* and X a *random variable*. The event "a three is rolled" is denoted by $\{X = 3\}$, and can formally be thought of as the set of outcomes for which a three is rolled (i.e., just $\{3\}$). Now consider the event $E = \{X \leq 3\}$ of rolling no higher than a 3. Then E corresponds to the set of outcomes

$\{1, 2, 3\} \subseteq \Omega$. Suppose we would like to know $\mathbb{P}(E)$. Since we assumed the die was fair, each outcome has probability $\frac{1}{6}$, which allows us to use *additivity of probabilities*:

Proposition 3: additivity of probabilities

Let E_1 and E_2 be disjoint events (i.e., $E_1 \cap E_2 = \emptyset$). Then $\mathbb{P}(E_1 \cup E_2) = \mathbb{P}(E_1) + \mathbb{P}(E_2)$.

Using Proposition 3, we can calculate the probability of event $E = \{X \leq 3\}$:

$$\begin{aligned}\mathbb{P}(\{X \leq 3\}) &= \mathbb{P}(\{X = 1\} \cup \{X = 2\} \cup \{X = 3\}) \\ &= \mathbb{P}(\{X = 1\}) + \mathbb{P}(\{X = 2\}) + \mathbb{P}(\{X = 3\}) \qquad \text{by Prop. 3} \\ &= \frac{1}{6} + \frac{1}{6} + \frac{1}{6} = \frac{1}{2}\end{aligned}$$

EXAMPLE 1.10

As above, let X denote the outcome of rolling a fair 6-sided die. Then the event $\{X \text{ is even}\}$ is the same as the set of outcomes $\{2, 4, 6\} \subseteq \Omega$. We can calculate the probability of this event using additivity again:

$$\begin{aligned}\mathbb{P}(\{X \text{ is even}\}) &= \mathbb{P}(\{X = 2\} \cup \{X = 4\} \cup \{X = 6\}) \\ &= \mathbb{P}(\{X = 2\}) + \mathbb{P}(\{X = 4\}) + \mathbb{P}(\{X = 6\}) \qquad \text{by Prop. 3} \\ &= \frac{1}{6} + \frac{1}{6} + \frac{1}{6} = \frac{1}{2}\end{aligned}$$

When events intersect, then Proposition 3 cannot be applied as is. For example, consider the events $E_1 = \{X \leq 3\}$ and $E_2 = \{X \text{ is odd}\}$. Then

$$\frac{2}{3} = \mathbb{P}(E_1 \cup E_2) \neq \mathbb{P}(E_1) + \mathbb{P}(E_2) = 1.$$

However, we could still apply Proposition 3 if we decompose $E_1 \cup E_2$ into disjoint sets. For instance, we can write

$$E_1 \cup E_2 = (E_1 \setminus E_2) \cup (E_2 \setminus E_1) \cup (E_1 \cap E_2). \tag{1.1}$$

It follows that for any (possibly intersecting) events E_1 and E_2:

$$\begin{aligned}\mathbb{P}(E_1 \cup E_2) &= \mathbb{P}\big((E_1 \setminus E_2) \cup (E_2 \setminus E_1) \cup (E_1 \cap E_2)\big) \\ &= \mathbb{P}(E_1 \setminus E_2) + \mathbb{P}(E_2 \setminus E_1) + \mathbb{P}(E_1 \cap E_2) \qquad \text{by (1.1)} \\ &= \mathbb{P}(E_1) + \mathbb{P}(E_2) - \mathbb{P}(E_1 \cap E_2).\end{aligned}$$

In summary, we have just shown the following "inclusion-exclusion" property of probabilities:

Proposition 4: inclusion-exclusion for probabilities

Let E_1 and E_2 be any two events. Then $\mathbb{P}(E_1 \cup E_2) = \mathbb{P}(E_1) + \mathbb{P}(E_2) - \mathbb{P}(E_1 \cap E_2)$.

EXAMPLE 1.11 inclusion-exclusion for probabilities

Consider the events $E_1 = \{X \leq 3\}$ and $E_2 = \{X \text{ is odd}\}$ from above. In this case, $E_1 \cap E_2 = \{1, 3\}$, and so Proposition 4,

$$\mathbb{P}(E_1 \cup E_2) = \mathbb{P}(E_1) + \mathbb{P}(E_2) - \mathbb{P}(E_1 \cap E_2) = \frac{1}{2} + \frac{1}{2} - \frac{1}{3} = \frac{2}{3}.$$

Conditional probabilities and independence. Let E_1 and E_2 be two events. Suppose we would like to know: what is the probability of event E_1 *given* that event E_2 occurs? We refer this as a *conditional probability*.

Definition 6: conditional probability

Let E_1 and E_2 be two events, with $\mathbb{P}(E_2) \neq 0$. The probability of E_1 given E_2, denoted $\mathbb{P}(E_1 \mid E_2)$, is $\mathbb{P}(E_1 \mid E_2) = \frac{\mathbb{P}(E_1 \cap E_2)}{\mathbb{P}(E_2)}$.

Notice that the conditional probability $\mathbb{P}(E_1 \mid E_2)$ is obtained by intersecting E_1 with E_2, and normalizing by $\mathbb{P}(E_2)$. The reason that we intersect with E_2 is that we are assuming E_2 occurs, which means that we should ignore all outcomes outside of E_2. The reason that we normalize by $\mathbb{P}(E_2)$ is to ensure that $\mathbb{P}(E_2 \mid E_2) = 1$, which is intuitively what we want.

Condition probabilities provide one way to measure how "related" two events are. If two events E_1 and E_2 are fairly unrelated, then one would expect $\mathbb{P}(E_1)$ to be quite similar to $\mathbb{P}(E_1 \mid E_2)$ (in other words, the occurrence of E_2 doesn't have a big impact on the probability of E_1). When two events are completely unrelated, they are call *independent*, which we define precisely below.

Definition 7: independence

Events E_1 and E_2 are *independent* if $\mathbb{P}(E_1 \cap E_2) = \mathbb{P}(E_1) \cdot \mathbb{P}(E_2)$.

Notice that if E_1 and E_2 are independent, then

$$\mathbb{P}(E_1 \mid E_2) = \frac{\mathbb{P}(E_1 \cap E_2)}{\mathbb{P}(E_2)} = \frac{\mathbb{P}(E_1)\mathbb{P}(E_2)}{\mathbb{P}(E_2)} = \mathbb{P}(E_1)\,,$$

which matches our intuitive notion of independence.

Caution: independence is *not* the same thing as disjointness. For instance, let X_1 be the roll of a fair 6-sided die, and X_2 the roll of a different fair 6-sided die. Consider the events $E_1 = \{X_1 \leq 2\}$, $E_2 = \{X_1 \geq 3\}$, and $E_3 = \{X_2 = 5\}$. Then E_1 and E_2 are disjoint, but not independent; on the other hand, E_1 and E_3 are independent, but not disjoint.

Expected value. Again, let X denote the roll of a fair die. We would like to know the average of X, i.e., the average number rolled by a fair die. This average is referred to as the *expected value* of X.

Definition 8: expected value

Let X be a random variable with finite outcome space $\Omega = \{\omega_1, \ldots, \omega_n\} \subseteq \mathbb{R}$. Then the *expected value of* X, denoted $\mathbb{E}[X]$, is

$$\mathbb{E}[X] = \sum_{\omega \in \Omega} \mathbb{P}(X = \omega) \cdot \omega = \mathbb{P}(X = \omega_1) \cdot \omega_1 + \cdots + \mathbb{P}(X = \omega_n) \cdot \omega_n.$$

EXAMPLE 1.12 expected die roll

Let X be the outcome of a die roll. In this case, the outcome space is $\{1, \ldots, 6\}$, and the expected value is

$$\mathbb{E}[X] = \sum_{i=1}^{6} \mathbb{P}(\{X = i\}) \cdot i = \sum_{i=1}^{6} \frac{i}{6} = \frac{7}{2}.$$

EXAMPLE 1.13 Bernoulli random variables

Let X have outcome space $\Omega = \{0, 1\}$, with $\mathbb{P}(\{X = 1\}) = p$ and $\mathbb{P}(\{X = 0\}) = 1 - p$. In other words, X is a *Bernoulli random variable with parameter* p. The expected value of X is

$$\mathbb{E}[X] = \sum_{i=0}^{1} \mathbb{P}(\{X = i\}) \cdot i = (1 - p) \cdot 0 + p \cdot 1 = p.$$

Suppose we know that $\mathbb{E}[X_1] = m_1$ and $\mathbb{E}[X_2] = m_2$. Can we calculate $\mathbb{E}[X_1 + X_2]$? One useful quality of expected values is that they respect linear combinations. In particular, we can use *linearity of expectation* to simplify the expected value of a linear combination of random variables.

Proposition 5: linearity of expectation

Let $X_1, \ldots, X_n$ be random variables with finite outcome spaces, and let $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$. Then

$$\mathbb{E}[\alpha_1 X_1 + \cdots + \alpha_n X_n] = \sum_{i=1}^{n} \alpha_i \mathbb{E}[X_i].$$

EXAMPLE 1.14 binomial random variables

Let $X_1, \ldots, X_n$ be independent Bernoulli random variables with parameter p, and let $X = \sum_{i=1}^{n} X_i$. Then X is called a *binomial random variable with parameters n and p*. By linearity of expectation, we have

$$\mathbb{E}[X] = \mathbb{E}\Big[\sum_{i=1}^{n} X_i\Big] = \sum_{i=1}^{n} \mathbb{E}[X_i] = \sum_{i=1}^{n} p = np.$$

Continuous random variables. All the examples and propositions above were for random variables with finite outcome spaces, but the theory extends easily to continuous random variables. Continuous random variables typically have $\mathbb{R}$ (or some subset of $\mathbb{R}$) as an outcome space, and so the outcome space cannot be "summed over" as we did above in defining expected value. Instead, we must use the continuous analog of a sum, which is to say, an integral.

Before we can discuss the expected value of a continuous random variable. For many continuous random variables, singleton outcomes have probability 0. For example, if X is normally distributed, then $\mathbb{P}(\{X = x\}) = 0$ for any $x \in \mathbb{R}$. In order to get around this issue and measure probabilities in small intervals, we use *probability density functions*.

Definition 9: density function

Let X be a continuous random variable with outcome space $\mathbb{R}$. A *density function* for X, denoted $f_X(x)$, is a function satisfying

$$\mathbb{P}(\{X \in [a, b]\}) = \int_a^b f_X(x)\,\mathrm{d}x$$

for all $a \leq b \in \mathbb{R}$.

EXAMPLE 1.15 uniform random variables

Let X be a uniform random variable on $[0, 1]$, i.e., $X \sim \text{Unif}[0, 1]$. Convince yourself that

$$f_X(x) = \begin{cases} 0 & x < 0 \\ 1 & 0 \leq x \leq 1 \\ 0 & 1 < x \end{cases}.$$

We are now ready to define the expected value of a continuous random variable.

Definition 10: expected value of a continuous random variable

Let X be a random variable with outcome space $\mathbb{R}$ and density function $f_X(x)$. Then, provided that $x f_X(x)$ is integrable over $\mathbb{R}$, the *expected value* of X is

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x f_X(x)\,\mathrm{d}x.$$

EXAMPLE 1.16 expected value of a uniform random variable

Let $X \sim \text{Unif}[a, b]$ be uniformly distributed over the interval $[a, b]$. Then

$$\mathbb{E}[X] = \int_a^b \frac{x}{b-a}\,\mathrm{d}x = \frac{a+b}{2}\,.$$

Work out the details to convince yourself that this is true!

Concentration inequalities. One core idea in machine learning is that by observing many data points, one can estimate the "average" value of the underlying distribution. The probabilistic idea that bolsters this is that sample means converge to true means. Here we discuss several useful inequalities that relate a random variable to its mean. We begin with Chebyshev's inequality, which gives quadratic decay of random variables from their means.

Proposition 6: Chebyshev's inequality

Let X be a continuous random variable with finite mean μ and finite variance σ^2. Then for any $t > 0$,

$$\mathbb{P}[\{|X - \mu| \geq t\sigma\}] \leq \frac{1}{t^2}.$$

Hoeffding's inequality is useful in bounding the convergence rate of a sample mean. In particular, Hoeffding's inequality shows exponential convergence to the true mean as the sample size increases.

Proposition 7: Hoeffding's inequality

Let $X_1, \ldots, X_n$ be independent random variables with outcome space $\Omega \subseteq [0, 1]$, and let $\overline{X} = \frac{1}{n}\sum_{i=1}^n X_i$ be the sample mean. Then

$$\mathbb{P}(\{|\overline{X} - \mathbb{E}[\overline{X}]| \geq t\}) \leq 2\exp(-2nt^2).$$

EXERCISES

1.3.1 Consider k independent Bernoulli variables $X_1, X_2, \ldots, X_k$ with parameters $0 < p_i < 1$ for $i = 1, \ldots, k$, i.e., $\text{Prob}(X_i = 1) = p_i$. What is the expectation of the sum $\sum_{i=1}^k X_i$? What if these variables are not independent?

1.4 Linear algebra

In this course, we will often work over the vector space $\mathbb{R}^d$ (d-dimensional Euclidean space). In addition to being a vector space, $\mathbb{R}^d$ is equipped with an *inner product*, which provides the geometric backbone of $\mathbb{R}^d$. The inner product we will use is the standard dot product.

Definition 11: dot product

Let $\mathbf{x} = (x_1, \ldots, x_d), \mathbf{y} = (y_1, \ldots, y_d) \in \mathbb{R}^d$. Then $\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^d x_i y_i$.

The dot product satisfies the following properties.

Proposition 8: dot product properties

For any $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^d$ and $c \in \mathbb{R}$, the following hold:

1. (symmetry) $\mathbf{x} \cdot \mathbf{y} = \mathbf{y} \cdot \mathbf{x}$;
2. (linearity) $(c\mathbf{x} + \mathbf{y}) \cdot \mathbf{z} = c(\mathbf{x} \cdot \mathbf{z}) + \mathbf{y} \cdot \mathbf{z}$;
3. (positive definiteness) $\mathbf{x} \cdot \mathbf{x} \geq 0$, and
$$\mathbf{x} \cdot \mathbf{x} = 0 \iff x = 0.$$

Inner products, such as dot product, provide us with rich structure, and induce a notion of angle, size, and distance. We will first discuss measuring size in $\mathbb{R}^d$, which we do via a *norm*.

Definition 12: 2-norm

Let $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$. The *2-norm* of $\mathbf{x}$, denoted $\|\mathbf{x}\|_2$ or $\|\mathbf{x}\|$, is

$$\|\mathbf{x}\|_2 = \sqrt{\mathbf{x} \cdot \mathbf{x}} = \Big(\sum_{i=1}^d x_i^2 \Big)^{1/2}.$$

The 2-norm is the usual Euclidean measure of size, or distance from the origin. It satisfies the following properties.

Proposition 9: 2-norm properties

For any $\mathbf{x} \in \mathbb{R}^n$ and $c \in \mathbb{R}$, the following hold:

1. (homogeneity) $\|c\mathbf{x}\|_2 = |c| \|\mathbf{x}\|_2$.
2. (non-negativity) $\|\mathbf{x}\|_2 \geq 0$;
3. (non-degeneracy) $\|\mathbf{x}\|_2 = 0 \iff \mathbf{x} = 0$;
4. (triangle inequality) $\|\mathbf{x} + \mathbf{y}\|_2 \leq \|\mathbf{x}\|_2 + \|\mathbf{y}\|_2$.

Having a notion of size (like the 2-norm) also allows for a notion of distance. In particular, the 2-norm induces the following metric:

Definition 13: Euclidean (ℓ^2) distance

For any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, the *distance* between $\mathbf{x}$ and $\mathbf{y}$ induced by the 2-norm is $\|\mathbf{x} - \mathbf{y}\|_2$.

The reader should check that the following properties hold (hint: use Proposition 9).

Proposition 10: Euclidean distance properties

Let $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^d$, and let $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2$. Then the following hold:

1. (identity of indiscernibles) $d(\mathbf{x}, \mathbf{y}) \geq 0$, and $d(\mathbf{x}, \mathbf{y}) = 0 \iff \mathbf{x} = \mathbf{y}$;
2. (symmetry) $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$; and
3. (triangle inequality) $d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z})$.

Other norms. The 2-norm is probably the most intuitive notion of size in $\mathbb{R}^d$, since it matches our geometric notion of length. There are, however, many other ways to measure sizes of vectors. For example, the *1-norm* of a vector $x \in^d$ is defined as $\|x\|_1 = \sum_{i=1}^{d} |x_i|$, and its *supremum norm* (or ∞-*norm*) is $\|x\|_\infty = \max_{i=1,\ldots,d} |x_i|$. One can check that the properties in Proposition 9 also apply to these norms.

Linear programming. Linear programming is the problem of maximizing a linear function subject to linear constraints. This problem is quite general, has been well-studied, and has widespread applications in operations research, computer science, and combinatorial optimization. We formally define linear programming below.

Definition 14: linear programming

The *linear programming* problem with input $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^m$, and $A = [\mathbf{a_1} \cdots \mathbf{a_m}]^\top \in \mathbb{R}^{m \times n}$, is the following optimization problem:

$$\max\{\mathbf{c}^\top \mathbf{x} : A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0\}.$$

Here vector inequality is entry-wise (for example, $(x_1, x_2) \leq (y_1, y_2)$ means $x_1 \leq y_1$ and $x_2 \leq y_2$). The linear function $\mathbf{c}^\top \mathbf{x}$ is called the *objective function*, and each inequality $\mathbf{a}_i^\top \mathbf{x} \leq b_i$ is called a *constraint*.

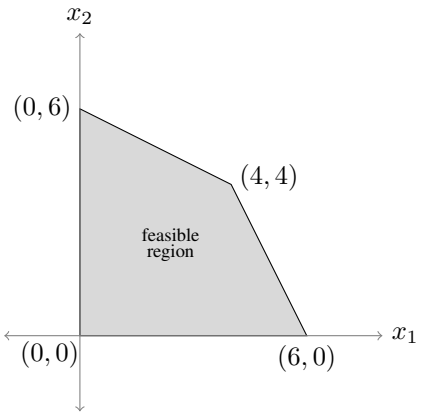
EXAMPLE 1.17 formulating an LP

Suppose you are manufacturing generators, and would like to maximize the total number of generators produced in a given month. Generators are produced at two different plants, P_1 and P_2, and the generator produced by P_1 differ from those produced by P_2. Those produced at P_1 weigh 50kg and take up 2m^3, while those produced at P_2 weigh 100kg and take up 1m^3. Due to shipping constraints, only 600kg can be shipped, and there is only enough space for 12m^3.

We can formulate the problem of determining how to produce the generators as a linear program. Let x_1 denote the number of generators produced in P_1, and x_2 the number of generators produced in P_2. Then we are trying to maximize $x_1 + x_2$. We thus arrive at the following LP:

$$\begin{array}{ll} \text{maximize} & x_1 + x_2 \\ \text{subject to} & 50x_1 + 100x_2 \leq 600 \\ & 2x_1 + x_2 \leq 12 \\ & x_1, x_2 \geq 0. \end{array}$$

The *feasible region* of a linear program is the set of points which satisfy the constraints. For the LP above, we obtain the following feasible region:



The optimal point must be somewhere on this polytope — and in fact, one of the vertices must be optimal. Since our example is so small, we can manually check the objective value at each vertex to determine that $(4, 4)$ is the optimal point, with objective value 8.

In Example 1.17, we formulated an LP and manually checked each vertex to determine the optimum. In general, this may not be feasible, as the number of vertices can be prohibitively large (exponential in the number of constraints). There are, however, more efficient ways of solving LPs, such as the simplex method, and solving LPs is known to be polynomial-time solvable.

EXAMPLE 1.18 linear programming duality

Consider the following linear program.

$$\begin{array}{ll} \text{maximize} & 3x_1 + 2x_2 \\ \text{subject to} & x_1 + x_2 \leq 4 \\ & 2x_1 - x_2 \leq 3 \\ & x_1 + 2x_2 \leq 5 \\ & x_1, x_2 \geq 0. \end{array}$$

Suppose we want to upper bound the value of this LP. To do this, suppose we have some point (x_1, x_2) that satisfies the constraints. If we add up two times the first

constraint plus one times the third constraint, we get

$$13 = 2(4) + 1(5) \geq 2(x_1 + x_2) + 1(x_1 + 2x_2) = 3x_1 + 4x_2.$$

Since $x_1, x_2 \geq 0$, this implies the following bound on the objective:

$$3x_1 + 2x_2 \leq 3x_1 + 4x_2 \leq 13.$$

Since (x_1, x_2) was an arbitrary feasible point, we can conclude that the value of this LP is at most 13.

Can we find a better bound? To do so, let's generalize the approach we took above. Instead of multiplying the constraints by 2, 0, and 1, we will multiply them by y_1, y_2, y_3, respectively:

$$\begin{array}{lll} \text{maximize} & 3x_1 + 2x_2 & \\ \text{subject to} & x_1 + x_2 \leq 4 & y_1 \\ & 2x_1 - x_2 \leq 3 & y_2 \\ & x_1 + 2x_2 \leq 5 & y_3 \\ & x_1, x_2 \geq 0. & \end{array}$$

In order to carry out the same analysis as above, we need to ensure that

$$3x_1 + 2x_2 \leq y_1(x_1 + x_2) + y_2(2x_1 - x_2) + y_3(x_1 + 2x_2).$$

In other words, we need to ensure that

$$\begin{aligned} y_1 + 2y_2 + y_3 &\geq 3 \\ y_1 - y_2 + 2y_3 &\geq 2. \end{aligned}$$

If this is the case, and if $y_1, y_2, y_3 \geq 0$, then we have that

$$3x_1 + 2x_2 \leq 4y_1 + 3y_2 + 5y_3,$$

which gives us a bound on our LP. Stated in other terms, we have just shown that

$$\begin{array}{lll} \text{maximize} & 3x_1 + 2x_2 & \\ \text{subject to} & x_1 + x_2 \leq 4 & \\ & 2x_1 - x_2 \leq 3 & \\ & x_1 + 2x_2 \leq 5 & \\ & x_1, x_2 \geq 0. & \end{array} \quad \leq \quad \begin{array}{ll} \text{min} & 4y_1 + 3y_2 + 5y_3 \\ \text{subject to} & y_1 + 2y_2 + y_3 \geq 3 \\ & y_1 - y_2 + 2y_3 \geq 2 \\ & y_1, y_2, y_3 \geq 0. \end{array}$$

Thus, the value of our original maximization LP is bounded above by the value of a specific minimization LP. This minimization LP is called the *dual* of our original LP, and the bound that we showed is called *weak duality*. In fact, one can show that the two LPs have the same value—a result which is called *strong duality*.

EXERCISES

1.4.1 Let $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$. Show that $\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_2$. In other words, show that $\max\{|x_i| : 1 \leq i \leq d\} \leq \|\mathbf{x}\|_2$.

1.4.2 Decide which of the following statements are true for all $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$.

(a) $\|\mathbf{x}\|_1 \geq \|\mathbf{x}\|_2$.

(b) $\|\mathbf{x}\|_2 \geq \|\mathbf{x}\|_1$.

(c) $\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_2 \leq \|\mathbf{x}\|_1$.

1.4.3 Suppose $A, B \in \mathbb{R}^{3\times 3}$, and $\operatorname{rank}(A) = 1$, and $\operatorname{rank}(B) = 2$. Give an upper bound and a lower bound on $\operatorname{rank}(AB)$, and provide an example to demonstrate that your bounds are tight. (In other words, if the bounds you give are $\ell \leq \operatorname{rank}(AB) \leq u$, then provide matrices A_1, B_1, A_2, B_2 with $\operatorname{rank}(A_1) = \operatorname{rank}(A_2) = 1$, $\operatorname{rank}(B_1) = \operatorname{rank}(B_2) = 2$, $\operatorname{rank}(A_1B_1) = \ell$ and $\operatorname{rank}(A_2B_2) = u$).

1.4.4 Give an example of a linear program with an unbounded objective (i.e., the optimal objective value is ∞).

1.4.5 Give an example of a linear program with an empty feasible region (i.e., an LP for which no points satisfy all the constraints).

1.4.6 Suppose you are the operations manager for a peanut butter company, and you need to supply peanut butter to two stores: S_1 and S_2. S_1 requires 100 jars, and S_2 requires 200 jars. There are two plants producing peanut butter, P_1 and P_2, and the shipping costs vary: it costs c_{ij} to send one jar from P_i to S_j. Additionally, P_2 can only produce up to 150 jars. Formulate an LP to determine the most cost-efficient way to supply the two stores with peanut butter. Then, find the dual of this LP.

1.4.7 Consider the linear program

$$\begin{array}{ll} \text{maximize} & 2x_1 - 9x_2 + x_3 \\ \text{subject to} & \mathbf{a}_1^\top \mathbf{x} \leq b_1 \\ & \mathbf{a}_2^\top \mathbf{x} \leq b_2 \\ & \mathbf{a}_3^\top \mathbf{x} \leq b_3 \\ & x_1, x_2, x_3 \geq 0\,, \end{array}$$

where $\mathbf{x} = (x_1, x_2, x_3)$ are the decision variables, and let v_1 denote the optimal objective value of this LP. Let v_2 denote the optimal objective value of the following LP:

$$\begin{array}{ll} \text{maximize} & 2x_1 - 9x_2 + x_3 \\ \text{subject to} & \mathbf{a}_1^\top \mathbf{x} \leq b_1 \\ & \mathbf{a}_2^\top \mathbf{x} \leq b_2 \\ & x_1, x_2, x_3 \geq 0. \end{array}$$

Is it true that $v_1 \leq v_2$? Is it true that $v_2 \leq v_1$?

HINTS

1.4.1 Square both sides of the inequality.

1.4.2 Plugging in the vector $\mathbf{x} = (1, 1, \ldots, 1)$ can help rule some options out.

1.4.4 Make sure the feasible region is unbounded (otherwise the objective will surely be bounded).

1.4.5 To rephrase the exercise, come up with a linear program whose constraints are contradictory (i.e., cannot all be satisfied by the same point).

1.5 Convex analysis (optional)

Suppose you are trying to find the global minimum of a function over $\mathbb{R}$. In general, this task can be difficult—even impossible—since functions can have many local minima, and functions can oscillate wildly. So, in order to minimize a function with provable guarantees, restrictions must be made on the function.

The most common restriction is to consider only *convex* functions, which have many nice properties such as having at most one local minimum. A function is *convex* if the region above its graph is a convex set. Equivalently:

Definition 15: convex function

A function $f : \mathbb{R}^d \to \mathbb{R}$ is *convex* if for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ and $\lambda \in [0, 1]$,

$$f\big(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}\big) \leq \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y}).$$

To check if a *continuous* function is convex, it suffices to check that Definition 15 holds for $\lambda = \frac{1}{2}$.

Proposition 11: continuity-based convexity condition

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a continuous function. Then f is *convex* if and only if for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$,

$$f\left(\frac{\mathbf{x}+\mathbf{y}}{2}\right) \leq \frac{f(\mathbf{x}) + f(\mathbf{y})}{2}.$$

For differentiable functions, we can characterize convexity in terms of derivatives.

Proposition 12: first-order convexity condition

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a differentiable function. Then f is convex if and only if for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$,

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top (\mathbf{y} - \mathbf{x}).$$

EXERCISES

1.5.1 Let $f : \mathbb{R}^d \to \mathbb{R}$ be a non-differentiable convex function. Show that $cf(\mathbf{x})$ is a convex function, for any $c \geq 0$.

1.5.2 Let $f : \mathbb{R}^d \to \mathbb{R}$ be a non-differentiable convex function. Show that $g(\mathbf{x}) = f(\mathbf{x}) + c$ is a convex function, for any $c \in \mathbb{R}$.

1.5.3 Let $f : \mathbb{R}^d \to \mathbb{R}$ be a twice continuously differentiable function. Then f is *convex* if and only if its Hessian is positive semidefinite (i.e., $\nabla^2 f(\mathbf{x}) \succeq 0$ for all $\mathbf{x} \in \mathbb{R}^d$). Using this fact, determine if $f(x, y) = \frac{1}{2}(x^2 + y^2) + xy$ is convex.

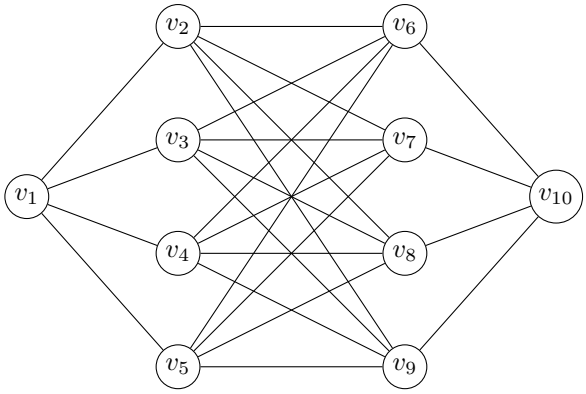
HINTS

1.5.1 Use Definition 15.

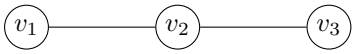
1.5.2 Use Definition 15.

1.6 Graph Theory (optional)

A graph is a collection of vertices and edges. Graphs are often depicted pictorially, for example:



We can formally represent a graph G as an ordered pair (V, E) of a vertex set V and an edge set E. An edge from vertex v_1 to vertex v_2 is typically denoted as an unordered pair (v_1, v_2). For example, the graph $G = (V = \{v_1, v_2, v_3\}, E = \{(v_1, v_2), (v_2, v_3)\})$ is as depicted below:

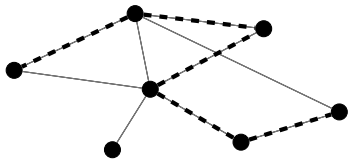


Definition 16: path

Let $G = (V, E)$ be a graph. A *path* in G is a sequence of distinct vertices $(v_1, v_2, \ldots, v_k)$ such that $(v_i, v_{i+1}) \in E$ for all $i = 1, \ldots, k - 1$.

EXAMPLE 1.19 path

Below is a graph with a path marked in dashes.



We say that a graph is *connected* if it is in one piece. More precisely:

Definition 17: connected graph

A graph $G = (V, E)$ is *connected* if for any vertices $v_1, v_2 \in V$, there is a path from v_1 to v_2.

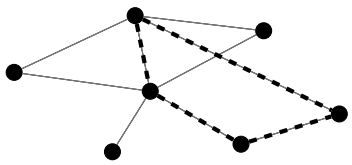
Definition 18: cycle

Let $G = (V, E)$ be a graph. A *cycle* in G is a sequence of vertices $(v_1, v_2, \ldots, v_k, v_1)$, with $k \geq 3$, such that

- $(v_i, v_{i+1}) \in E$ for all $i = 1, \ldots, k-1$;
- $(v_k, v_1) \in E$; and
- the vertices $v_1, \ldots, v_k$ are distinct.

EXAMPLE 1.20 cycle

Below is a graph with a cycle marked in dashes.



Notice that if a graph has a cycle, then there are multiple paths between certain pairs of vertices. In that sense, graphs with cycles are not minimally connected. For this reason, we will sometimes be concerned with graphs without cycles.

Definition 19: acyclic graph

A graph $G = (V, E)$ is *acyclic* if it contains no cycles.

Definition 20: tree

A graph $G = (V, E)$ is a *tree* if G is acyclic and connected.

A *subgraph* of a graph G is any graph obtained by deleting edges and vertices (along with incident edges) from G. More precisely:

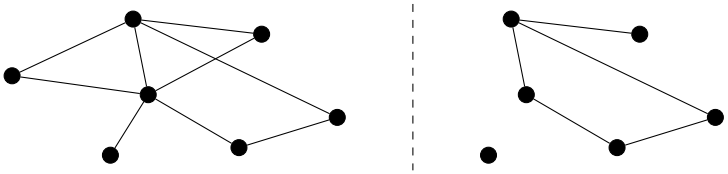
Definition 21: subgraph

Let $G = (V, E)$ be a graph. A graph $H = (V', E')$ is a *subgraph* of G if:

1. $V' \subseteq V$ and
2. $E' \subseteq E$.

EXAMPLE 1.21 subgraph

Below is a graph (a) with one of its subgraphs (b).



(a) (b)

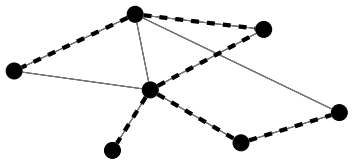
A minimially connected subgraph (i.e., one for which deleting any edge disconnects the graph) covering all vertices of a connected graph is called a *spanning tree*.

Definition 22: spanning tree

Let $G = (V, E)$ be a graph. A *spanning tree* of G is a connected, acyclic subgraph $T = (V, E')$ of G.

EXAMPLE 1.22 spanning tree

Below is a graph with a spanning tree marked in dashes.



EXERCISES

1.6.1 Let $G = (V, E)$ be a connected graph on $n = |V|$ vertices, with $n \geq 1$, and let $T = (V, E')$ be a spanning tree of G. Show that $|E'| = n - 1$.

1.6.2 K_3, sometimes referred to as a *triangle*, is the complete graph on 3 vertices. That is, K_3 is the graph on 3 vertices where each pair of vertices is connected by an edge. Find the maximum number of triangles in a graph on n vertices.

1.6.3 Design an algorithm for determining whether a graph is connected. Express the running time of your algorithm using big-oh notation.

1.6.4 Design an algorithm for determining the shortest path between two vertices in a graph. Express the running time of your algorithm using big-oh notation.

REFERENCES

[1] Lay, David C. *Linear Algebra and its applications*, 5th edition. Pearson (2016).

[2] Goodaire, Edgar G., and Michael M. Parmenter. *Discrete mathematics with graph theory.* Prentice Hall PTR, 1997.

[3] Grimmett, Geoffrey, and Dominic Welsh. *Probability: an introduction.* Oxford University Press, 2014.

HINTS

1.1.1 Consider $\mathbb{Z}$ and $\mathbb{Q}$ for example. Note that every integer n can be written as a fraction $\frac{n}{1}$, and so *every integer is a rational number*. This tells us that $\mathbb{Z} \subseteq \mathbb{Q}$. Conversely, since there are rational numbers which are not integers (e.g., $\frac{1}{2}$), so it is **not** the case that $\mathbb{Q} \subseteq \mathbb{Z}$. Try to make similar statements about the other pairs of sets.

1.1.2 Consider, for example, a bounded interval which is open on one end and closed on the other.

1.1.4 Make the argument simpler by assuming $\max A \leq \max B$. Under this assumption, there are two things to show: $\max B$ is an upper bound on $A \cup B$, and $\max B \in A \cup B$.

1.2.2 Use the change-of-base formula for logarithms.

1.2.3 By definition, we know that there exist c_1, n_1 satisfying $|f(n)| \leq c_1|g(n)|$ for all $n \geq n_1$. Use c_1 and n_1 to find appropriate c and n_0.

1.2.4 Use the triangle inequality.

1.4.1 Square both sides of the inequality.

1.4.4 Make sure the feasible region is unbounded (otherwise the objective will surely be bounded).

1.4.5 To rephrase the exercise, come up with a linear program whose constraints are contradictory (i.e., cannot all be satisfied by the same point).

1.5.1 Use Definition 15.

1.5.2 Use Definition 15.

1.6.1 Use induction on n.

1.6.2 Note that any three vertices form a triangle if each pair of vertices is connected by an edge. So, you can upper bound the number of triangles in a graph by the number of sets of three vertices.

1.6.3 Pick a vertex and start walking to other vertices. If you can reach every other vertex in the graph, then you know that the graph is connected. There are various ways you can do this.

1.6.4 Start at one of the two vertices (say, v_1) and begin walking the graph. At each new vertex, you will have a choice as to which vertex to visit next. Try to make these choices in a way that ensures that each path taken is a shortest path from v_1.

SOLUTIONS

1.1.1 $\{1, 3, 5\} \subseteq \mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{R}$.

1.1.2 The set $[0, 1)$ has a minimum but no maximum. The set $(0, 1]$ has a maximum but no minimum.

1.1.4 There are two cases to consider: $\max A \leq \max B$ and $\max B \leq \max A$. Without loss of generality, we will simply assume that $\max A \leq \max B$.

We first show that $\max B$ is an upper bound on $A \cup B$. To show this, let $x \in A \cup B$. If $x \in A$, then $x \leq \max A \leq \max B$, and if $x \in B$, then $x \leq \max B$. So, $\max B$ is an upper bound on $A \cup B$.

It remains to show that $\max B \in A \cup B$. However, this is true because $\max B \in B \subseteq A \cup B$.

1.2.2 For all $n \geq 1$, we have that $\log_{b_1}(n) = \frac{\log_{b_2}(n)}{\log_{b_2}(b_1)}$. So, Definition 5 is satisfied for $c = \frac{1}{\log_{b_2}(b_1)}$ and $n_0 = 1$.

1.2.3 Since $f(n) \in \mathcal{O}(g(n))$, there exist c_1, n_1 such that $|f(n)| \leq c_1|g(n)|$ for $n \geq n_1$. Then for $c = c_1|\lambda|$ and $n_0 = n_1$,

$$|\lambda f(n)| \leq |\lambda| c_1 |g(n)| = c|g(n)| \text{ for all } n \geq n_0.$$

1.2.4 By assumption, there exist c_1, n_1, c_2, n_2 such that

$$\begin{aligned} |f(n)| &\leq c_1|h(n)| \quad \text{for } n \geq n_1, \text{ and} \\ |g(n)| &\leq c_2|h(n)| \quad \text{for } n \geq n_2. \end{aligned}$$

Then for $n \geq \max\{n_1, n_2\}$,

$$\begin{aligned} |f(n) + g(n)| &\leq |f(n)| + |g(n)| && \text{by the triangle inequality} \\ &\leq c_1|h(n)| + c_2|h(n)| && \text{since } n \geq \max\{n_1, n_2\} \\ &= (c_1 + c_2)|h(n)|. \end{aligned}$$

So, setting $c = c_1 + c_2$ and $n_0 = \max\{n_1, n_2\}$, we see that Definition 5 is satisfied.

1.4.1 Without loss of generality, let $|x_1| = \max\{|x_i| : 1 \leq i \leq d\}$. Then

$$\max\{|x_i| : 1 \leq i \leq d\}^2 = x_1^2 \leq \sum_{i=1}^{d} x_i^2 = \|\mathbf{x}\|_2^2.$$

Taking the square root of both sides gives the desired inequality.

1.4.4 One example is $\max\{x : x \geq 0\}$. Here the feasible region is $[0, \infty)$, and the objective is unbounded.

1.4.5 One example is

$$\begin{aligned} \text{maximize} \quad & x_1 + 2x_2 \\ \text{subject to} \quad & x_1 + x_2 \leq -1 \\ & x_1, x_2 \geq 0. \end{aligned}$$

1.5.1 Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ and $\lambda \in [0,1]$. Then

$$\begin{aligned} cf\big(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}\big) &\leq c\big[\lambda f(\mathbf{x}) + (1-\lambda)f(\mathbf{y})\big] && \text{by convexity of } f \\ &= \lambda c f(\mathbf{x}) + (1-\lambda)cf(\mathbf{y}). \end{aligned}$$

1.5.2 Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ and $\lambda \in [0,1]$. Then

$$\begin{aligned} g\big(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}\big) &= f\big(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}\big) + c \\ &\leq \lambda f(\mathbf{x}) + (1-\lambda)f(\mathbf{y}) + c \\ &= \lambda\big(g(\mathbf{x}) - c\big) + (1-\lambda)\big(g(\mathbf{y}) - c\big) + c \\ &= \lambda g(\mathbf{x}) + (1-\lambda)g(\mathbf{y}). \end{aligned}$$

1.6.1 If $n = 1$, then any tree has 0 edges, and if $n = 2$, then the only spanning tree is K_2, which has 1 edge. Now let $G = (V, E)$ be a connected graph on $n > 2$ vertices, and suppose that for all $2 \leq k < n$, any tree on k vertices has $k - 1$ edges.

Let $T = (V, E')$ denote any spanning of G, and let e be any edge of T. Then by acyclicity of T, $T - e$ is disconnected. It follows that $T - e$ is composed of two disjoint trees, $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ with $|V_1| + |V_2| = n$. By the inductive hypothesis, $|E_1| = |V_1| - 1$ and $|E_2| = |V_2| - 1$. So,

$$|E'| = |E_1| + |E_2| + |\{e\}| = (|V_1| - 1) + (|V_2| - 1) + 1 = |V| - 1 = n - 1.$$

1.6.2 Since any three vertices can form a triangle, we have that $\#\text{triangles} \leq \binom{n}{3}$.

1.6.3 Run a depth-first search on G. If G is connected and some vertices are never reached, let v be a "nearest" missed vertex (i.e., v is never visited by the DFS, but some neighbor u of v is reached. When the algorithm reaches u, v must be added to the queue of vertices to visit. This contradicts that the DFS never reaches v.

1.6.4 Let s and t be the vertices in question (i.e., you are trying to find the shortest path from s to t). Now run a breadth-first search starting from s. We claim, more generally, that *every path produced by the BFS is a shortest path from* s. We proceed by induction on the distance from s. If $d(s, v) = 1$, then s and v are connected by an edge, and the BFS will traverse the edge (s, v).

Now let v be a vertex of minimal distance from s for which the s-v path produced by BFS is not a shortest path. Let $P = sx_1 \dots x_k v$ be the path from s to v produced by the BFS, and let $P' = sy_1 \dots y_r v$ be a shortest path. Then by the inductive hypothesis, BFS produces a shortest path $P'' = sz_1 \dots z_\ell y_r$ from s to y_r. But then, since v is a neighbor of y_r, the path produced by BFS from s to v must have distance at most

$$d(s, y_r) + 1 \leq \text{len}(P') < \text{len}(P),$$

a contradiction.

CHAPTER 2

INTRODUCTION TO ONLINE LEARNING

SWATI GUPTA AND JAD SALEM

Georgia Institute of Technology, Atlanta, Georgia
Last updated: September 30, 2022

This chapter is a work in progress and has not been subjected to the usual scrutiny reserved for formal publications. If you find errors in this draft, please let the authors know. We would be happy to acknowledge you in the final version.

We all engage with optimization on a daily basis. The roads we travel on are governed by traffic rules that try to minimize congestion. We receive targeted advertisements that try to maximize sales. We use social media platforms that personalize our feeds to maximize screen time. These optimization problems, and many others, are not static in time. If a popular restaurant opens, then traffic patterns may shift, and an adjustment of traffic lights might be appropriate. Our interests shift over time, so an advertiser might want to change how they target advertisements over time. Similarly, social and political dynamics might affect how we engage with social media, so a social media company may want to change the way they organize users' feeds over time. These are all complex and highly relevant problems, which explains the burgeoning literature on the topic. In this chapter, we will introduce the *online learning* framework, examples, and related concepts.

2.1 Introduction to Online Learning

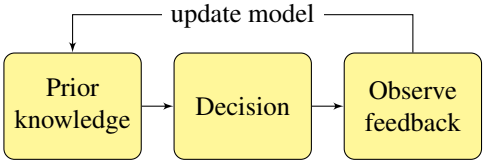
Suppose you are trying to design an algorithm which classifies emails as "spam" or "not spam." In this setting, emails arrive one by one over time, and the algorithm should classify the emails as they arrive. Problems, such as this one, where the input arrives as a stream over time, are referred to as *online* problems; problems in which the entire input is accessible to the algorithm at once are called *offline* problems (often called the batch setting). An ideal goal to strive for is to minimize the number of misclassified emails.

Food for Thought 1: algorithm goals

What other goals could one have when designing an algorithm? Suppose the typical distribution of spam emails is 90% emails with coupons and advertisements to buy products, 5% emails with fake medicines, 3% emails with fake university degrees, and 2% emails asking for bank account details. Given this breakdown, can you think of an algorithmic goal in designing a spam-filtering algorithm? Does your goal prioritize the proper identification of certain types of spam emails?

One strategy to design an algorithm that reduces the total number of mistakes of an algorithm would be to design a decision rule—that is, an algorithm which takes in an email and outputs whether or not it is spam—and apply this same decision rule to each subsequent email. This strategy, however, can be improved upon greatly. When an email is misclassified, users are able to report the misclassification, and the algorithm will have access to this information. This feedback can allow for more informed decisions in the future—it makes sense to base a decision at time t on all available information, which means taking into account all previous emails and their true classifications. Algorithms for online problems which use feedback to inform their future decisions are called *online learning* algorithms. Using this feedback can result in much more accurate decisions over time, since the algorithm's data set to learn from increases.

In general, the *online learning framework* can be described as follows: (1) **decisions** (e.g., spam classification, or approving someone for a credit card, or choosing a route in a map) are made by the algorithm iteratively; and (2) after a decision is made, the algorithm receives feedback in the form of a numerical **feedback/loss** (e.g., whether the marked email was actually spam or not, or whether the person who got the loan paid it back or not, or the time it took to travel on the selected path). A higher loss means that the decision was bad, and the goal of the algorithm will be to reduces the losses in each round. We will make this more precise in the later sections. The last and most important part of the framework is (3) **updates** to the algorithm's model using the feedback in each round, so that better decisions can be taken thereafter. This loop of making decisions, observing feedback and updating algorithm's model continue and thus, the algorithm is able to *learn online.*



This framework will be discussed in more detail in Section 2.3. For now, let us consider another example.

EXAMPLE 2.1 advertisement

Suppose you are on an advertising team, and are trying to figure out which of several advertisements is most effective. Each time a user is shown one of the advertisements, your algorithm can observe whether or not the advertisement is effective, e.g., whether or not the user clicks on the advertisement. In this setting, the online decisions that are made are the choices of advertisements (which advertisement to show), and the feedback obtained by the algorithm is whether the user clicked on the advertisement or not (0 or 1). The loss, in this case, is 1 unit whenever the user does not click on the shown advertisement.

2.2 Mind Reader Experiment

We begin this section with a food for thought.

Food for Thought 2: predictability

Imagine there is an algorithm trying to predict which coffee shop you will go to on each day—*Amélie's Bakery* or the *Blue Donkey Coffee Shop*. If you like one of these better, perhaps the algorithm could just predict that coffee shop and be correct, say, 90% of the time. Could you change your coffee drinking places so that the algorithm could not be correct more than 50% of the time? Or even lesser?

To see whether you would be successful in fooling a "coffee shop prediction algorithm," let us consider the following simple game. At each round, you must choose either "left" or "right," and an algorithm tries to guess your choice. If the algorithm guesses correctly, it gets a point, and otherwise, you get a point. In other words, the algorithm is trying to accurately predict your moves, and you are trying to trick the algorithm into guessing wrong. This game can be found at `https://web.media.mit.edu/~guysatat/MindReader/index.html`, and the reader is encouraged to play.

Suppose you make your decisions truly at random: at each time t, you choose $X_t =$ "left" with probability $\frac{1}{2}$, and all of these choices are independent. Then after T rounds, regardless of the strategy of the algorithm, the expected score of the algorithm would be $T/2$. In this case, you would expect to win about 50% of the time. Similarly, in the coffee example above, if one chooses coffee shops at random, then the algorithm should be fooled 50% of the time. However, after playing this game, you will likely find that the algorithm is more difficult to beat.

Why is this? Perhaps the flaw in the above logic is assuming our choices to be random (arguably, humans are not true random number generators). Whether consciously or unconsciously, the decisions we make tend to follow some pattern. While this pattern may be so subtle that a person cannot identify it, statistical tools and machine learning algorithms may be able to exploit the pattern to make more accurate predictions.

Food for Thought 3: flipping a bad algorithm

Suppose we have an algorithm which predicts which of the two coffee shops one will go to, and this algorithm is accurate less than 50% of the time. Can we use this

Food for Thought 3: flipping a bad algorithm (cont.)

algorithm to make correct predictions more than 50% of the time? The answer is yes—if we simply take the prediction of the algorithm and make the opposite prediction, then this new algorithm would be correct more than 50% of the time.

What if there were three coffee shops, and you have an algorithm which predicts the correct coffee shop less than $\frac{1}{3}$ of the time. Is it possible to use this algorithm to make correct predictions more than $\frac{1}{3}$ of the time?

In many applications in real life, the loss functions or feedback on decisions are generated through natural socio-economic interactions. However, often we would like our algorithms to perform well even when bad data is input or *worst-case* losses are presented to the algorithm. This is modeled by thinking of losses generated by an *adversary* in each round. Think of an adversary as someone who knows what the algorithm is going to do (except randomness within the algorithm, but we will get to that later), and generates losses to cause the algorithm to make mistakes. For example, think back about the coffee shop prediction algorithm. If you go to coffee shops with the intention of tricking the algorithm, then you are acting as an adversary. Our goal in this course will be to design algorithms that can predict *well*, in spite of the feedback or losses being generated through an adversary or by adversarial behavior.

2.3 General Framework

In this section, we discuss a general yet simple framework for online learning problems. This framework will encompass all of the online learning problems presented in this course.

In the simplest online learning setting, an algorithm (sometimes referred to as the decision-maker) makes decisions from a decision set $\mathcal{K}$ over time, using only feedback from past decisions (not future decisions). For instance, an online navigation algorithm which returns a "shortest path" between two places will do so without knowledge of the congestion that will be faced on the path. In this case, the decision set $\mathcal{K}$ would be the set of paths from the starting location to the destination. An online algorithm makes decisions iteratively—one decision per time step—and observes feedback only after committing to a decision.

Form of feedback. The numerical feedback on a decision observed by the algorithm is typically a measure of how good the decision was. The goal is either to minimize this quantity (e.g., if the feedback measures deviation from the optimum) or maximize this quantity (e.g., if the feedback measures success). When the feedback is to be minimized, it is often referred to as *loss*, and when it is to be maximized, it is referred to as *reward*. These two settings are essentially the same: maximizing a reward function is the same as minimizing the negative reward. For this reason, we will typically restrict our attention to the loss setting and think of rewards as negative losses. Since the loss at time t depends on the decision made by the algorithm, the loss $\ell_t : \mathcal{K} \to \mathbb{R}$ is a function from the decision space to the real numbers.

EXAMPLE 2.2 pricing

Suppose you are selling a book, and must choose whether to sell it at \$5 or \$10. This choice can be changed over time, as new customers arrive. Suppose there are two types of customers—one that would pay \$5 for the book, and the other that would be willing to pay \$10 for the book. The seller (i.e., the online learning algorithm), however, does not know which type of a customer each person is. If the only goal is to maximize profit, then the optimal price point at time t would be $\$5$ for low paying customers and $\$10$ for high paying customers. If a customer who's willing to pay \$5 only is offered the book at \$10, then they will leave. Whenever this happens, the loss experienced by the seller (i.e., algorithm) is \$6, where \$1 is for a "missed sale" or unhappy customer, and it is 0 if the sale is actually made.

Example 2.2 above above concerns dynamic pricing—the practice of varying prices of goods across time and across customer segments. As the pricing of goods affects quality of life, the societal impact of pricing algorithms should be taken into account. This is something we will discuss in more depth later in the course.

Food for Thought 4: price discrimination

In the real world, there are ethical concerns with pure profit or sales maximization by dynamically changing prices. If an algorithm routinely assigns higher prices to some demographic group, then the algorithm could be inadvertently starving some populations of essential goods such as medicines or books. See, for instance, this ProPublica piece highlighting how unfettered dynamic pricing can result in higher prices for lower-income people.

At this point, you might wonder *which functions can be loss functions?* Minimizing arbitrary online functions over an infinite decision space is an impossible task (why?), and so assumptions must be made about loss functions. In this course, loss functions will typically be linear and sometimes *non-linear but convex* (see Section 1.5), which gives some hope for minimization.

EXAMPLE 2.3 infinite decision spaces with arbitrary losses

Let's revisit the "coffee shop prediction" algorithm. Suppose you can even teleport at *any point in time*, to either *Amelie's Bakery* or *Blue Donkey Coffee*. In this case, the decision space $\mathcal{K} = \{\text{Amelie's}, \text{Blue Donkey}\} \times [-\infty, \infty]$, where the second set is the time at which you decide to go. The algorithm now has to predict the time, as well as the coffee shop. This decision set is unbounded (due to the time dimension), and in this case, it is highly unlikely that the algorithm would correctly predict the coffee shop as well as the time at which you decide to visit. Since it is so difficult to learn over infinite decision spaces, we often either assume that the decision space is finite, or compact and bounded (e.g., polytopes).

The second crucial assumption that we need to be able to develop meaningful algorithms is that of the loss functions being bounded or finite at each time step.

Food for Thought 5: unbounded loss

Suppose we are in a setting with unbounded loss functions. In this case, suppose without any information in the first round, the algorithm takes a decision $x_1 \in \mathcal{K}$. If the loss for this decision is essentially infinite (say a million), and the losses thereafter for any decision at times $t > 1$ decrease significantly (say the loss at time t is less than t^{-3} for $t > 1$), then there is no way that an algorithm can recover from the initial infinite loss. For this reason, we will assume that loss functions are bounded, so that we can develop good algorithms.

This assumption is quite natural in many settings. For example, suppose you are selling salt and pepper shakers, and are choosing how to price them. If each salt and pepper shaker set costs \$1 to make, then the maximum amount of loss that the algorithm can experience in one round is \$1. These bounds typically result from costs of production or finiteness of resources, and can be made without much loss of generality.

In the online learning framework, there is a loss function ℓ_t for each time step t that is revealed after the algorithm takes a decision, and so the loss functions can vary over time. We will assume that the set of loss functions is $\mathcal{L} \subseteq \{\ell : \mathcal{K} \to [a, b]\}$, i.e., there are known bounds on the loss functions. We will typically assume that these loss functions are chosen *adversarially*; i.e., the loss functions are chosen (with knowledge of the algorithm) in order to maximize the amount of loss suffered by the algorithm.[1] Note that not all real-world settings are adversarial; e.g., dependent on the route you select, the cars on the road do not try to congest that route adversarially. That said, we would like to keep the framework as general as possible, and develop algorithms which remain competitive even in the adversarial setting.

We will think of the online learning framework as a repeated interaction between the algorithm Alg and the adversary: Alg makes a decision, then the adversary chooses a loss function, then Alg makes a decision, and so on. For example, in the Mind Reader example of Section 2.2, Alg would make a left/right decision, and then the adversary (i.e., the player trying to beat the computer) will select a loss by selecting left/right, and then Alg would make a left/right decision, etc. The online learning framework, stated precisely, is:

online learning framework

In each iteration $t = 1, \dots, T$:

1. Alg selects a decision $x_t \in \mathcal{K}$;
2. A convex loss function $\ell_t \in \mathcal{L}$ is decided by the adversary;
3. Alg suffers (and observes) a loss of $\ell_t(x_t)$ during this round; some additional information about the function $\ell_t(\cdot)$ may be observed by the algorithm as well, dependent on the setting.

[1] To get an intuition of the adversarial setting, think of a game of tic tac toe. In each round, a player strategically places their mark with the goal of preventing their opponent from forming a string of three marks. In this way, if you are playing tic tac toe, your opponent can be thought of as an *adversary*.

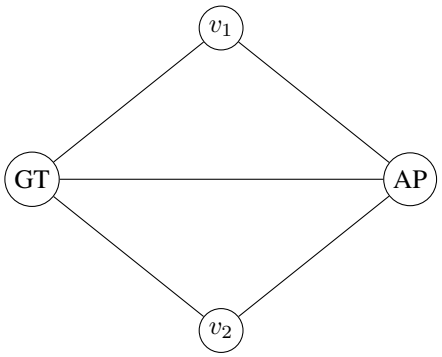
online learning framework (cont.)

4. Alg incorporates this information for the next round.

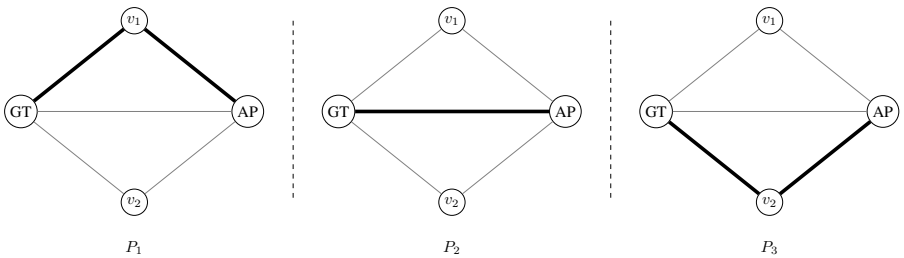
Full versus partial information. Once a loss function ℓ_t is set (either by nature or an adversary), the algorithm uses information about the loss function to adapt for future decisions. In these notes, we classify online learning problems by how much information about the loss functions are available to the algorithm. In the *full information setting*, the algorithm has access to the entire function $\ell_t(\cdot)$ (or in certain settings, the gradient $\nabla \ell_t$); in other words, once $\ell_t : \mathcal{K} \to \mathbb{R}$ is set, the algorithm gets the information about the entire loss function, i.e., it can access $\ell_t(x)$ for any $x \in \mathcal{K}$. In the *partial information setting*, the algorithm can only query ℓ_t at certain points; one important class of online learning problems with partial information is *bandit* problems, where if the algorithm makes decision x_t at time t, then it would only observe $\ell_t(x_t)$, and no other function values. Let's consider an example to differentiate between different settings.

EXAMPLE 2.4 congestion

Suppose you are designing a navigation algorithm which seeks to find the fastest path from Georgia Tech (GT) to the airport (AP), given current traffic. Suppose the following graph represents the possible roads that can be taken to get to AP.



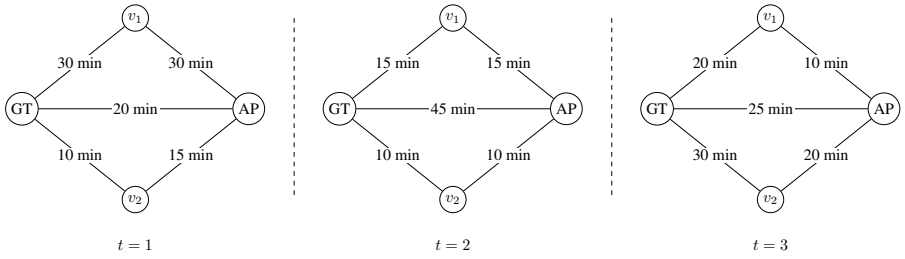
In particular, this means that there are three possible paths to take:



To put this in the language on online learning, the decision space is $\mathcal{K} = \{P_1, P_2, P_3\}$.

Depending on the current congestion of the roads, the time it takes to travel these paths can vary, and it is natural to consider total travel time as the loss. Suppose that

at time $t = 1$ (you can think of it as the first hour), the time required to cross edge (GT, AP) is 20 minutes. Then, if the algorithm decides to suggest the direct path from GT to AP, the loss would be 20 minutes (check that indeed, minimizing the loss over time would mean that the algorithm takes the least congested paths). Perhaps the travel times for $t = 1, 2, 3$ are:



Suppose that after choosing a path P, if the algorithm is given the travel times of all edges, this would be considered a *full information setting*. If the algorithm is only given the total time taken on the path selected, then this is called the *bandit setting*. If on the other hand, the algorithm is given the travel time for each edge traversed (but not the other edges), then this setting would be considered a *semi-bandit setting*, since the algorithm is not given full feedback. The last two regimes are called *partial information* settings.

The full and partial information settings are quite different in spirit; both involve learning which decisions are "better" than others, but the partial information settings have the added complication of information gathering. In the partial information setting, some *exploration* is required: in order to learn which decisions are good, a variety of decisions must be selected over time. In such problems, there is typically a trade-off between how much time is spent gathering this information (*exploration*) and how much time is spent using this information to strategically make decisions (*exploitation*).

With all these variations of the online learning framework (choice of decisions, choice of loss functions, feedback types), we find that the simple online learning model can be tweaked for many interesting scenarios! Some important applications are spam detection (online classification or regression), sequential investment or portfolio management, aggregating weather predictions (experts problem), product recommendations (bandit setting), and personalized product recommendations (contextual bandits setting), to name a few. We will develop algorithms for some of these applications later in the course. We now discuss the metric of deciding when an algorithm is good.

Regret in online learning: We measure the performance of online learning algorithms using a notion known as **regret**, which is the difference between the loss incurred by the algorithm and the best fixed decision in hindsight.

Definition 23: regret

The *regret* of an online learning algorithm Alg is

$$\text{regret}_T(\mathsf{Alg}) = \sup_{\ell_1,\ldots,\ell_T \in \mathcal{L}} \left\{ \sum_{t=1}^{T} \ell_t(x_t) - \min_{x \in \mathcal{K}} \sum_{t=1}^{T} \ell_t(x) \right\},$$

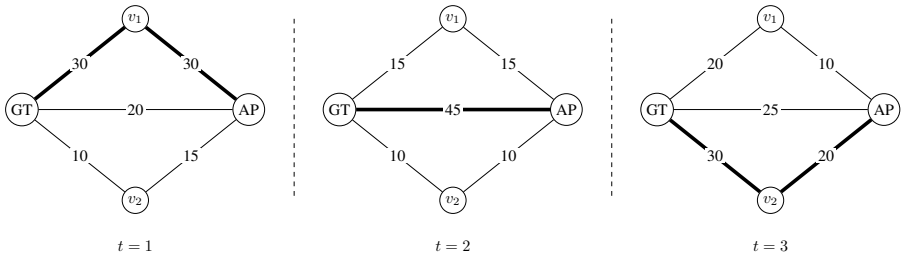
Definition 23: regret (cont.)

where $x_1, \ldots, x_T$ are the decisions made by the algorithm, and $\ell_1, \ldots, \ell_T$ are the loss functions chosen by the adversary.

Recall from Section 1.1 that "sup" refers to the supremum of a set, which is a generalization of the maximum.

EXAMPLE 2.5 congestion (Example 2.4, cont.)

Recall that you are designing a navigation algorithm which seeks to find the fastest path from Georgia Tech (GT) to the airport (AP), and the travel times (in minutes) vary as:



Then, if the algorithm chooses the bolded paths above (P_1, P_2, and P_3, in that order), the total loss (travel time) experienced by the algorithm is 155 minutes. The "benchmark" (or adversary in the worst case), on the other hand, has only three choices, since they cannot change paths between time steps. The total loss experienced by the adversary for these choices are listed below:

Path	P_1	P_2	P_3
Loss	120	90	95

So, the adversary would choose path P_2, as this is the optimal fixed path. In summary, for this choice of loss functions (congestions) and these algorithm choices, the algorithm experiences a loss of 155, and the adversary experiences a loss of 90.

Note that in this example, we are comparing the performance of an algorithm to an adversary *given a fixed sequence of loss functions*. Throughout these notes, we will develop tools for obtaining *provable* regret bounds (recall that regret is the supremum over *all* sequences of loss functions). From this example, we know that the regret of the algorithm is at least 65.

Intuitively, the algorithm performs well if the regret is small. This "smallness" is formalized with big-oh analysis, which is introduced in Section 1.2.

EXAMPLE 2.6 linear regret

Consider the following online learning problem: the decision space is $\mathcal{K} = [0, 1]$, and the possible loss functions are $\mathcal{L} = \{x, 1 - x\}$. Suppose an algorithm Alg is as bad as possible; i.e., whenever $\ell_t(x) = x$, $x_t = 1$, and whenever $\ell_t(x) = 1 - x$, $x_t = 0$. In

this case,

$$\begin{aligned}\text{regret}_T(\mathsf{Alg}) &= \sup_{\ell_1,\ldots,\ell_T\in\mathcal{L}} \left\{ \sum_{t=1}^{T} \ell_t(x_t) - \min_{x\in\mathcal{K}} \sum_{t=1}^{T} \ell_t(x) \right\} \\ &= \sup_{\ell_1,\ldots,\ell_T\in\mathcal{L}} \left\{ T - \min_{x\in\mathcal{K}} \sum_{t=1}^{T} \ell_t(x) \right\} \\ &\geq \sup_{\ell_1,\ldots,\ell_T\in\mathcal{L}} \left\{ T - \sum_{t=1}^{T} \ell_t\left(\frac{1}{2}\right) \right\} \\ &= \frac{T}{2}.\end{aligned}$$

Whenever we have that $\text{regret}_T(\mathsf{Alg}) \in \Theta(T)$, we say that the regret is *linear*. In this example, we saw that by making the worst possible decisions, we incurred linear regret. In fact, since loss functions are typically bounded, this phenomenon is quite general: an algorithm which consistently makes the worst choice will typically experience linear regret.

Based on Example 2.6, one goal in designing an online learning algorithm is to achieve *sublinear regret*, i.e., $\text{regret}_T(\mathsf{Alg})/T \to 0$ as $T \to \infty$. In other words, we would like the regret to grow more slowly than the number of iterations.[2] If an algorithm achieves sublinear regret, then the average loss the algorithm suffers after a long period of time converges to the **best loss the algorithm could suffer with a single fixed decision** with the knowledge of the loss functions $\ell_1, \ldots, \ell_t$. This is a good goal since after initial rounds of learning, where the algorithm is allowed to make some mistakes, in the long run it does converge to the best decision in hindsight.

Food for Thought 6: regret

Note that the benchmark used to define performance, the notion of regret, compares the algorithm's losses with the loss produced by the best possible fixed decision x^* only; i.e., the benchmark is the minimizer of all the losses seen up to time T:

$$x^* \in \arg\min_{x\in\mathcal{K}} \sum_{t=1}^{T} \ell_t(x).$$

However, the *adversary* has full control over the sequence of loss functions. The algorithm, on the other hand, has no control over or knowledge of future loss functions, but is able to change decisions over time. In this way, neither the algorithm nor the adversary is "more powerful" than the other, and meaningful, interesting algorithms can be designed for online learning problems . What other metrics of performance can one use to measure the quality of online algorithms?

EXERCISES

[2]We will often use "big-oh" notation in this course to compare the growth of functions. Loosely, $f(x) \in \mathcal{O}(g(x))$ if $f(x) \leq Mg(x)$ for some constant M and large values of x. See Section 1.2 for more details.

2.3.1 Imagine an online learning application wherein a retail company, MaySee's, wants to set prices for a watch and learn what the customers are willing to pay for the watch. Each time period, a customer arrives at the shop and sees the price set by MaySee's. If the price is below what the customer is willing to pay, then they buy the watch. MaySee's observes this sale and sets the price again for a new watch for the next customer. Every time a customer does not buy a watch, the company observes a loss of 1 unit (since it looses a customer to Norkstorm). The watch costs \$50, and the company cannot sell it for below this price. The company also knows that the maximum price a customer is willing to pay for the watch is \$1000, but most of the customers are willing to pay a much less price. If the company finds the right price between \$50 and \$1000 it can both gain customers as well as make a profit on the watches. Which of the following is the decision set that MaySee's can choose from in this online learning problem?

(a) $\{0, 1, 2, 3, \ldots\}$ (the number of watches sold)

(b) $(0, \infty)$ (the price of the watch, in dollars)

(c) $[50, 1000]$ (the price of the watch, in dollars)

(d) $\{\text{Yes}, \text{No}\}$ (whether or not a customer buys the watch)

2.3.2 You have started a travel agency, and you are trying to figure out how your customers would like to get to Lake Lanier from Atlanta. There are three options:

> **Option 1:** take the subway and the line 408 bus (which is least expensive and most environmentally friendly);
>
> **Option 2:** take a taxi (which is convenient but costs money);
>
> **Option 3:** rent a car (which is convenient and cheaper than the taxi, but involves the responsibility of driving).

Your clientele is mostly undergraduate students at Georgia Tech, and therefore, you strongly suspect that one of these options is the clear winner. You decide to use an online learning algorithm to learn the best option. As each client comes in, you decide to show them one of the options (i.e., decisions are Options 1,2 or 3), and later call and get their feedback on whether they liked that option or would have preferred another option by ranking the three options. If your algorithm's suggested option was last in their ranking, let your algorithm incur a loss of 2. If it suggested their second most liked option, then the algorithm incurs a loss of 1. Otherwise the algorithm incurs no loss.

For example, suppose to user A, the algorithm Alg suggested Option 2, but the user's preference was "$1 > 2 > 3$." Then, the loss of ALG in this round is 1, since Option 2 was second best. If the user's preference was "$2 > 1 > 3$," then loss of ALG would be 0, since Option 2 was best. If the user's preference was "$3 > 1 > 2$," then the loss of ALG would be 2, since Option 2 was the least preferred option.

What is the total loss (over three time periods) incurred by the algorithm given the following information?

1. At $t = 1$: Adnan shows up. ALG suggests Option 2 to him. After the trip, he gives the feedback that his preference is $3 > 2 > 1$.
2. At $t = 2$: Keisha shows up. ALG suggests Option 2 to her. After the trip, she gives the feedback that her preference is $1 > 3 > 2$.

3. At $t = 3$: Elon shows up. ALG suggests Option 3 to him. After the trip, he gives the feedback that he indeed loved the suggestion, but would have preferred taking the taxi. His preference is $2 > 3 > 1$.

2.4 Applications of Online Learning

Online learning is useful wherever decisions need to be taken with partial information, where either the users come one by one in time, or data comes with time, or the data is too big that it has to be accessed sequentially (big data setting), or whenever the impact of decisions by the algorithm can only be partially observed.

We next discuss some examples of problems that can be modeled via the online learning framework. These examples will show you how powerful this simple framework is. Problems from diverse domains such as online routing, ad selection for search engines and spam filtering can all be modeled as special cases, as discussed in previous sections. In this section, we briefly survey a few special cases and how they fit into the online learning framework.

1. Prediction from expert advice. The *experts problem* is one of the most fundamental problems in online learning. To motivate it, suppose you have two financial advisers who advise you on when to sell stocks. Sometimes, their advice conflicts, and you can ultimately observe who was right and who was wrong. Over time, you can get a sense of which adviser is better, and rely on the better one more heavily.

In the experts problem, we have a set of n experts $\{1, \ldots, n\}$, and in each round, we must choose a probability distribution over the experts. For example, if $n = 2$, we might decide on a distribution of $(.3, .7)$ in round 10, which means that Expert 1's advice will be taken with probability 0.3 in round 10. Our decision set is therefore

$$\mathcal{K} = \Big\{ p \;\Big|\; \sum_{i=1}^{n} p(i) = 1,\ p \geq 0 \Big\},$$

which is called the *probability simplex*. In the experts framework, Expert i at time t experiences loss $\ell_t(i)$, and ℓ_t is the loss vector of all experts at time t. The loss incurred from decision p_t at time t is

$$p_t(1)\ell_t(1) + \cdots + p_t(n)\ell_t(n) = p_t^\top \ell_t.$$

We will further assume that the loss functions ℓ_t are *bounded*; i.e., $\sup_{t,i} |\ell_t(i)| \leq L$ for some $L \in \mathbb{R}$. e.g., if $L = 1$, then the adversary must choose each loss function ℓ_t from the following set:

$$\mathcal{L} = \{\ell \in \mathbb{R}^n \mid -1 \leq \ell(i) \leq 1, \text{for each expert } i\} = [-1, 1]^n.$$

Thus, prediction from expert advice is a special case of online learning in which the decision set is the set of probability distributions (i.e., the simplex) and the cost functions are linear and bounded. A *good* online learning algorithm will operate *as good* as the best the best expert in hindsight (up to some small additive error), even without having the knowledge of losses in the future. This example will be very important throughout the class as we can think of the experts as being a set of paths (in a routing application), or a set of advertisements (in a recommendations applications), or a set of stock portfolios (in

a financial investment problem). In each of these settings, the goal of the algorithm will be to select a path, advertisement or stock portfolio at each time, so that its performance is nearly as good as if it knew what was going to happen in the future but could pick a single decision (i.e., minimize its regret).

2. Recommender Systems. Suppose you run a movie streaming platform, Netflicks™, and would like to develop a system for predicting user preferences for movies. If the platform carries n movies, a user's movie preferences can be stored as a row vector $x \in \mathbb{R}^n$, where x_i corresponds to how much the user likes movie i. More generally, if there are m users, then the preferences of all users can be represented as a matrix $M \in \mathbb{R}^{m \times n}$, where M_{ij} is how much user i likes movie j. These preferences, however, can change over time.

In this example, each decision is a predicted preference matrix $M^{(t)} \in \mathbb{R}^{m \times n}$, and so $\mathcal{K} \subseteq \mathbb{R}^{m \times n}$. The feedback here might come from a user i rating a movie j that they watched, and the predicted preference matrices $M^{(t)}$ are updated each time a new rating is submitted. In the adversarial setting, the choice of (user, movie) pair and rating are chosen adversarially, although in the real world, this example may not be adversarial (a user typically does not watch a bad movie simply to trick the recommendation algorithm).

Let (i_t, j_t) be the (user, movie) pair at time t. The loss incurred at time t should correspond to whether or not the predicted preference $M^{(t)}_{i_t j_t}$ of i_t for j_t was close to the true preference y_t. We can measure this deviation by the *squared error* between predicted and actual preferences, namely,

$$\ell_t\big(M^{(t)}\big) = \big(M^{(t)}_{i_t j_t} - y_t\big)^2,$$

which is a convex function. So, in the adversarial setting, at each time step t, the adversary chooses the loss function be selecting i_t, j_t, and y_t.

As in most online learning problems, we use regret to measure performance. We want an algorithm which has low regret, which means that on average, it should be about as accurate as the optimal *fixed* preference matrix $M \in \mathcal{K}$.

Food for Thought 7: recommendation engines

In the real world, the mathematics of online learning is used *in conjunction* with all side-information available in the application that makes it easier for the algorithms to predict. What side information can such a recommendations platform use about users to decide which hotels to show them in a city queried by a user? Listen to the "AI in the Industry" podcast by Dan Faggella here, about how the Vice President of Data Science at `MakeMyTrip.com` thinks about this problem.

2.5 Connections to Machine Learning (optional)

In the online learning framework, note that we did not make any assumptions on the distribution of data, nor did we make assumptions on the relationship between input data and their true labels. Machine learning models typically consider data-value pairs (x_i, y_i) (for $i = 1, \ldots, n$, where each $x_i \in \mathbb{R}^n, y_i \in \mathbb{R}$) and use a big chunk of data (typically 80% of the data) to fit a model that minimizes losses to make predictions of the value y on unseen data points x in the test set. For example, simple least-squares linear regression minimizes

the total squared error $\sum_{i=1}^{n}(y_i - w^T x_i - b)^2$ to find the best linear model fit w, b on the training data, and then uses the model $y = wx + b$ to make subsequent predictions of y on the unseen test set. Many such "offline" machine learning algorithms can be now viewed in the online framework where the algorithm learns and gets better with respect to each data point it sees iteratively, and this is often useful in dealing with large amounts of data that is difficult to store in a single machine. In online learning, there is no explicit *training* or *testing* phase. Rather, in each round, the online learning algorithm predicts a value $\widetilde{y}_t$, learns the true value y_t, and uses the same example as a training example to help improve the prediction mechanism.

REFERENCES

[1] Hazan, Elad. "Introduction to Online Convex Optimization." Chapter 1, Sections 1.1-1.2.