CHAPTER 1

Generative Models for Discrete Data

In molecular biology, many situations involve counting events: how many codons use a certain spelling, how many reads of DNA match a reference, how many CG digrams are observed in a DNA sequence. These counts give us *discrete* variables, as opposed to quantities such as mass and intensity that are measured on *continuous* scales.

If we know the rules that the mechanisms under study follow, even if the outcomes are random, we can generate the probabilities of any events we are interested in by computations and standard probability laws. This is a *top-down* approach based upon **deduction** and our knowledge of how to manipulate probabilities. In Chapter 2, you will see how to combine this with data-driven (*bottom-up*) statistical modeling.

1.1 Goals for this chapter

In this chapter we will:

- Learn how to obtain the probabilities of all possible outcomes from a given model and see how we can compare the theoretical frequencies with those observed in real data.
- Explore a complete example of how to use the Poisson distribution to analyze data on epitope detection.
- See how we can experiment with the most useful generative models for discrete data: Poisson, binomial, multinomial.
- Use the R functions for computing probabilities and counting rare events.
- Generate random numbers from specified distributions.

1.2 A real example

Let's dive into a real example, where we know the probability model for the process. We are told that mutations along the genome of HIV (human immunodeficiency virus) occur at random with a rate of 5×10^{-4} per nucleotide per replication cycle. This means

that after one cycle, the number of mutations in a genome of about $10^4 = 10{,}000$ nucleotides will follow a *Poisson distribution*[1] with rate 5. What does that tell us?

[1] We will give more details later about this type of probability distribution.

This **probability model** predicts that the number of mutations over one replication cycle will be close to 5, and that the variability of this estimate is $\sqrt{5}$ (the standard error). We now have baseline reference values for both the number of mutations we expect to see in a typical HIV strain and its variability.

In fact, we can deduce even more detailed information. If we want to know how often 3 mutations could occur under the Poisson(5) model, we can use an R function to generate the probability of seeing $x = 3$ events, taking the value of the *rate parameter* of the Poisson distribution, called lambda (λ), to be 5.

Greek letters such as λ and μ often denote important parameters that characterize the probability distributions we use.

```
dpois(x = 3, lambda = 5)
## [1] 0.1403739
```

This says the chance of seeing exactly three events is around 0.14, or about 1 in 7.

If we want to generate the probabilities of all values from 0 to 12, we do not need to write a loop. We can simply set the first argument to be the **vector** of these 13 values, using R's sequence operator, the colon " : ". We can see the probabilities by plotting them (Figure 1.1). As with this figure, most figures in the margins of this book are created by the code shown in the text.

Note how the output from R is formatted: the first line begins with the first item in the vector, hence the [1], and the second line begins with the 9th item, hence the [9]. This helps you keep track of elements in long vectors. The term *vector* is R parlance for an ordered list of elements of the same type (in this case, numbers).

```
0:12
##  [1]  0  1  2  3  4  5  6  7  8  9 10 11 12
dpois(x = 0:12, lambda = 5)
##  [1] 0.0067 0.0337 0.0842 0.1404 0.1755 0.1755 0.1462 0.1044
##  [9] 0.0653 0.0363 0.0181 0.0082 0.0034
barplot(dpois(0:12, 5), names.arg = 0:12, col = "red")
```

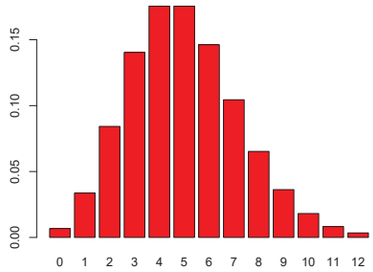

Figure 1.1: Probabilities of seeing 0,1,2,...,12 mutations, as modeled by the Poisson(5) distribution. The plot shows that we will often see four or five mutations but rarely as many as 12. The distribution continues to higher numbers (13, . . .), but the probabilities will be successively smaller, and here we don't visualize them.

Mathematical theory tells us that the Poisson probability of seeing the value x is given by the formula $e^{-\lambda}\lambda^x/x!$. In this book, we'll discuss theory from time to time, but give preference to displaying concrete numeric examples and visualizations like Figure 1.1.

The Poisson distribution is a good model for rare events such as mutations. Other useful probability models for **discrete events** are the Bernoulli, binomial and multinomial distributions. We will explore these models in this chapter.

1.3 Using discrete probability models

Think of a **categorical variable** as having different alternative values. These are the levels, similar to the different alternatives at a gene locus: alleles.

A point mutation can either occur or not; it is a binary event. The two possible outcomes (yes, no) are called the **levels** of the categorical variable.

Not all events are binary. For example, the genotypes in a diploid organism can take three levels (AA, Aa, aa).

Sometimes the number of levels in a categorical variable is very large; examples include the number of different types of bacteria in a biological sample (hundreds or thousands) and the number of codons formed of three nucleotides (64 levels).

When we measure a categorical variable on a sample, we often want to tally the frequencies of the different levels in a vector of counts. R has a special encoding for categorical variables and calls them **factors**.[2] Here we capture the different blood genotypes for 19 subjects in a vector that we tabulate.

[2] R makes sure that the factor variable will not accept other, "illegal" values, and this is useful for keeping your calculations safe.

```
genotype = c("AA","AO","BB","AO","OO","AO","AA","BO","BO",
             "AO","BB","AO","BO","AB","OO","AB","BB","AO","AO")
table(genotype)
```

```
## genotype
## AA AB AO BB BO OO
##  2  2  7  3  3  2
```

`c()` is a basic function. It collates elements of the same type into a vector. In this code, the elements of `genotype` are character strings.

On creating a factor, R automatically detects the levels. You can access the levels with the `levels` function.

```
genotypeF = factor(genotype)
levels(genotypeF)
```

```
## [1] "AA" "AB" "AO" "BB" "BO" "OO"
```

```
table(genotypeF)
```

```
## genotypeF
## AA AB AO BB BO OO
##  2  2  7  3  3  2
```

It is not obvious from the output of the `table` function that the input was a factor; however, if there had been another level with no instances, the table would also have contained that level, with a zero count.

► **Question 1.1** What if you want to create a factor that has some levels not yet in your data? ◄

► **Solution 1.1** Look at the manual page of the `factor` function. □

If the order in which the data are observed doesn't matter, we call the random variable **exchangeable**. In that case, all the information available in the factor is summarized by the counts of the factor levels. We then say that the vector of frequencies is **sufficient** to capture all the relevant information in the data, thus providing an effective way of compressing the data.

1.3.1 Bernoulli trials

Tossing a coin has two possible outcomes. This simple experiment, called a Bernoulli trial, is modeled using a so-called Bernoulli random variable. Understanding this building block will take you surprisingly far. We can use it to build more complex models.

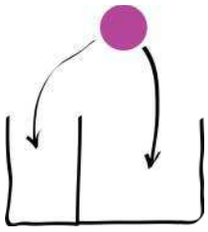

Figure 1.2: Two possible events with unequal probabilities. We model this by a **Bernoulli distribution** with probability parameter $p = \frac{2}{3}$.

Let's try a few experiments to see what some of these random variables look like. We use special R functions tailored to generate outcomes for each type of distribution. They all start with the letter `r`, which stands for random, followed by a specification of the model, here `rbinom`, where `binom` is the abbreviation used for binomial.

Suppose we want to simulate a sequence of 15 fair coin tosses. To get the outcome of 15 Bernoulli trials with a probability of success equal to 0.5 (a fair coin), we write

```
rbinom(15, prob = 0.5, size = 1)
##  [1] 1 1 0 0 1 1 0 0 0 1 0 1 0 1 0
```

[3] For R functions, parameters are also called *arguments*.

We use the `rbinom` function with a specific set of **parameters**:[3] the first parameter is the number of trials we want to observe; here we chose 15. We designate by `prob` the probability of success. By `size=1` we declare that each individual trial consists of just one single coin toss.

► **Question 1.2** Repeat this function call a number of times. Why isn't the answer always the same? ◄

[4] We call such events **complementary**.

Success and failure can have unequal probabilities in a Bernoulli trial, as long as the probabilities sum to one.[4] To simulate 12 trials of throwing a ball into the two boxes shown in Figure 1.2, with probability of falling in the right-hand box $\frac{2}{3}$ and in the left-hand box $\frac{1}{3}$, we write

```
rbinom(12, prob = 2/3, size = 1)
##  [1] 1 1 1 1 0 0 1 1 0 0 1 0
```

The 1 indicates success, meaning that the ball fell in the right-hand box; 0 means the ball fell in the left-hand box.

1.3.2 Binomial success counts

[5] The exchangeability property.

If we only care how many balls go in the right-hand box, then the order of the throws doesn't matter,[5] and we can get this number by just taking the sum of the cells in the output vector. Therefore, instead of the binary vector we saw above, we only need to report a single number. In R, we can do this using one call to the `rbinom` function with the parameter `size` set to 12.

Two outcomes and a size of 1 or more make this experiment a binomial trial. If the size is 1, then this is the special case of the Bernoulli trial.

```
rbinom(1, prob = 2/3, size = 12)
## [1] 5
```

This output tells us how many of the 12 balls fell into the right-hand box (the outcome that has probability $\frac{2}{3}$).

We use a random two-box model when we have only two possible outcomes, such as heads or tails, success or failure, CpG or non-CpG, M or F, Y = pyrimidine or R = purine, diseased or healthy, true or false. We only need to specify the probability, p, of "success" because "failure" (the *complementary* event) will occur with probability $1 - p$. When looking at the results of several such trials, if they are exchangeable,[6] we record only the number of successes. Therefore, `SSSSSFSSSSFFFSF` is summarized as (#Successes=10, #Failures=5), or as $x = 10$, $n = 15$.

[6] One situation in which trials are exchangeable is if they are independent of each other.

The number of successes in 15 Bernoulli trials with a probability of success of 0.3 is called a **binomial** random variable or a random variable that follows the $B(15, 0.3)$

distribution. To generate samples, we use a call to the `rbinom` function with the number of trials set to 15.

```
set.seed(235569515)
rbinom(1, prob = 0.3, size = 15)
## [1] 5
```

What does `set.seed` do here?

▶ **Question 1.3** Repeat this function call 10 times. What seems to be the most common outcome? ◀

▶ **Solution 1.3** The most frequent value is 4. In fact, the theoretical proportion of times that we expect 4 to appear is the value of the probability that $X = 4$ if X follows $B(15, 0.3)$. □

The complete **probability mass distribution** is available by typing

```
probabilities = dbinom(0:15, prob = 0.3, size = 15)
round(probabilities, 2)
##  [1] 0.00 0.03 0.09 0.17 0.22 0.21 0.15 0.08 0.03 0.01 0.00 0.00
## [13] 0.00 0.00 0.00 0.00
```

The function `round` keeps the number of printed decimal digits down to two.

We can produce a barplot of this distribution, shown in Figure 1.3.

```
barplot(probabilities, names.arg = 0:15, col = "red")
```

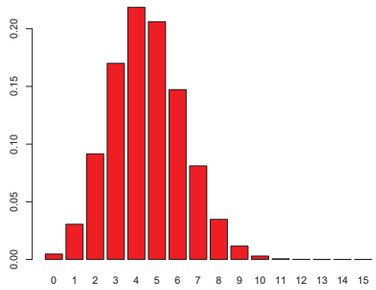

Figure 1.3: Theoretical distribution of $B(15, 0.3)$. The highest bar is at $x = 4$. We have chosen to represent theoretical values in red throughout.

The number of trials is the number we input to R as the `size` parameter and is often written n, while the probability of success is p. Mathematical theory tells us that for X distributed as a binomial distribution with parameters (n, p), written $X \sim B(n, p)$, the probability of seeing $X = k$ successes is

$$\begin{aligned} P(X = k) &= \frac{n \times (n-1) \ldots (n-k+1)}{k \times (k-1) \ldots 1} \, p^k \, (1-p)^{n-k} \\ &= \frac{n!}{(n-k)!k!} \, p^k \, (1-p)^{n-k} \\ &= \binom{n}{k} p^k \, (1-p)^{n-k}. \end{aligned}$$

The special notation $\binom{n}{k}$, called the binomial coefficient and read "n choose k", is a shortcut for $\frac{n!}{(n-k)!k!}$.

▶ **Question 1.4** What is the output of the formula for $k = 3, p = \frac{2}{3}, n = 4$? ◀

1.3.3 Poisson distributions

When the probability of success p is small and the number of trials n is large, the binomial distribution $B(n, p)$ can be faithfully approximated by a simpler distribution, the **Poisson distribution** with rate parameter $\lambda = np$. We used this fact, and this distribution, in the HIV example (Figure 1.1).

▶ **Question 1.5** What is the probability mass distribution of observing `0:12` mutations in a genome of $n = 10^4$ nucleotides, when the probability of mutation is $p = 5 \times 10^{-4}$ per nucleotide? Is it similar when modeled by the binomial $B(n, p)$ distribution and by the Poisson($\lambda = np$) distribution? ◀

Note that, unlike the binomial distribution, the Poisson does not depend on two separate parameters n and p, but only on their product np. As in the case of the binomial distribution, we also have a mathematical formula for computing Poisson probabilities:[7]

$$P(X = k) = \frac{\lambda^k \, e^{-\lambda}}{k!}.$$

For instance, let's take $\lambda = 5$ and compute $P(X = 3)$.

```
5^3 * exp(-5) / factorial(3)
## [1] 0.1403739
```

which we can compare with what we computed in Section 1.2 using `dpois`.

► **Task** Simulate a mutation process along 10,000 positions with a mutation rate of 5×10^{-4} and count the number of mutations. Repeat this many times and plot the distribution with the `barplot` function (see Figure 1.5). ◄

```
rbinom(1, prob = 5e-4, size = 10000)
## [1] 6
simulations = rbinom(n = 300000, prob = 5e-4, size = 10000)
barplot(table(simulations), col = "lavender")
```

Now we are ready to use probability calculations in a case study.

[7] This formula appeared briefly in Section 1.2.

Figure 1.4: Simeon Poisson, after whom the Poisson distribution is named (this is why it always has a capital letter, except in our R code). Image credit: Wikicommons.

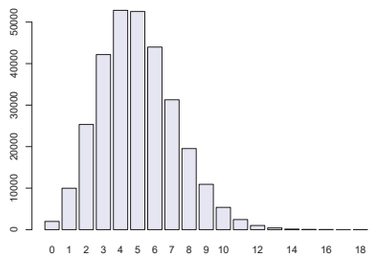

Figure 1.5: Simulated distribution of B(10,000, 10^{-4}) for 300,000 simulations.

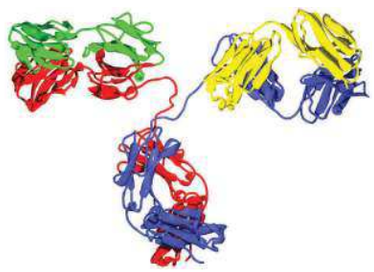

Figure 1.6: Structure of an IgG2 antibody showing several immunoglobulin domains in color. Image credit: Wikicommons.

1.3.4 A generative model for epitope detection

When testing certain pharmaceutical compounds, it is important to detect proteins that provoke an allergic reaction. The molecular sites that are responsible for such reactions are called **epitopes**. The technical definition of an epitope is:

> A specific portion of a macromolecular antigen to which an antibody binds. In the case of a protein antigen recognized by a T cell, the epitope or determinant is the peptide portion or site that binds to a major histocompatibility complex (MHC) molecule for recognition by the T-cell receptor (TCR).

And in case you're not so familar with immunology: an antibody (as schematized in Figure 1.6) is a type of protein made by certain white blood cells in response to a foreign substance in the body, which is called an antigen.

An antibody binds (with more or less specificity) to its antigen. The purpose of the binding is to help destroy the antigen. Antibodies can work in several ways, depending on the nature of the antigen. Some antibodies destroy antigens directly. Others help recruit white blood cells to destroy the antigen. An epitope, also known as antigenic determinant, is the part of an antigen that is recognized by the immune system, specifically by antibodies, B cells or T cells.

ELISA error model with known parameters

ELISA[8] assays are used to detect specific epitopes at different positions along a protein. Suppose the following facts hold for an ELISA assay we are using:

[8] **E**nzyme-**l**inked **i**mmuno**s**orbent **a**ssay – see *ELISA* on Wikipedia.

- The baseline noise level per position, or more precisely the false positive rate, is 1%. This is the probability of declaring a hit – we think we have an epitope – when there is none. We write this as P(declare epitope | no epitope).
- The protein is tested at 100 different positions, supposed to be independent.

We are going to examine a collection of 50 patient samples.

We read the vertical bar in expressions such as $X|Y$ as "given" or "conditional on". Thus, "X happens **conditional on** Y being the case".

One patient's data

The data for one patient's assay look like this:

```
##   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
##  [30] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [59] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [88] 0 0 0 0 0 0 0 0 0 0 0 0
```

where the 1 signifies a hit (and thus the potential for an allergic reaction), and the zeros signify no reaction at that position.

► **Task** Verify by simulation that the sum of 50 independent Bernoulli variables with $p = 0.01$ is – to good enough approximation – the same as a Poisson(0.5) random variable. ◄

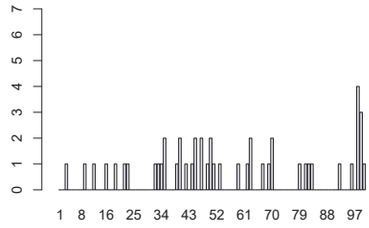

Figure 1.7: Plot of typical data from our generative model for the background, i.e., for the false positive hits: 100 positions along the protein, at each position the count is drawn from a Poisson(0.5) random variable.

Results from the 50 assays

We're going to study the data for all 50 patients tallied at each of the 100 positions. If there are no allergic reactions, the false positive rate of 1% means that for a single patient, each individual position has a probability of 1 in 100 of being a 1. So, after tallying 50 patients, we expect at any given position the sum of the 50 observed 0/1 variables to have a Poisson distribution with parameter 0.5. A typical set of false positives across the 100 positions may look like Figure 1.7.

Now suppose we see actual data as shown in Figure 1.8, loaded as an R object `e100` from the data file `e100.RData`.

```
load("../data/e100.RData")
barplot(e100, ylim = c(0, 7), width = 0.7, xlim = c(-0.5, 100.5),
  names.arg = seq(along = e100), col = "darkolivegreen")
```

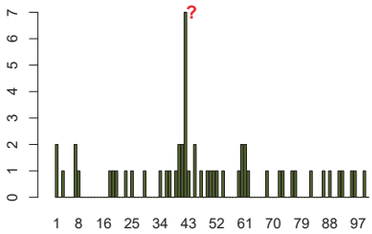

Figure 1.8: Output of the ELISA array results for 50 patients at the 100 positions.

The spike in Figure 1.8 is striking. *What are the chances of seeing a value as large as 7, if no epitope is present?*

If we look for the probability of seeing a number as big as 7 (or larger) when considering one Poisson(0.5) random variable, the answer can be calculated in closed form as

$$P(X \geqslant 7) = \sum_{k=7}^{\infty} P(X = k).$$

This is, of course, the same as $1 - P(X \leqslant 6)$. The probability $P(X \leqslant 6)$ is the so-called **cumulative distribution function** at 6, and R has the function `ppois` for computing it, which we can use in either of the following two ways.[9]

[9] Besides the convenience of not having to do the subtraction from 1, the second of these computations also tends to be more accurate when the probability is small. This has to do with limitations of floating point arithmetic.

```
1 - ppois(6, 0.5)
## [1] 1.00238e-06
ppois(6, 0.5, lower.tail = FALSE)
## [1] 1.00238e-06
```

► **Task** Check the manual page of `ppois` for the meaning of `lower.tail`. ◄

We denote this number by ϵ, the Greek letter epsilon.[10] We have shown that the probability of seeing a count as large as 7, assuming no epitope reactions, is

$$\epsilon = P(X \geqslant 7) = 1 - P(X \leqslant 6) \simeq 10^{-6}. \tag{1.1}$$

[10] Mathematicians often call small numbers (and children) epsilons.

Extreme value analysis for the Poisson distribution

Stop! The above calculation is *not* the correct computation in this case.

► **Question 1.6** Can you spot the flaw in our reasoning if we want to compute the probability that we observe these data if there is no epitope? ◄

► **Solution 1.6** We looked at all 100 positions, looked for the largest value and found that it was 7. Due to this initial selection from all positions, a value as large as 7 is more likely to occur than if we looked at only one position from the start. □

So instead of asking what the chances are of seeing a Poisson(0.5) as large as 7, we should ask ourselves, what are the chances that the maximum of 100 Poisson(0.5) trials is as large as 7? We will use **extreme value analysis** here.[11] We order the data values $x_1, x_2, \ldots, x_{100}$ and rename them $x_{(1)}, x_{(2)}, x_{(3)}, \ldots, x_{(100)}$, so that $x_{(1)}$ denotes the smallest and $x_{(100)}$ the largest of the counts over the 100 positions. Together, $x_{(1)}, \ldots, x_{(100)}$ are called the **rank statistic** of this sample of 100 values.

[11] Meaning that we're interested in the behavior of the very large or very small values of a random distribution, for instance the maximum or the minimum. This approach allows us to compute the probability of **rare events**.

The maximum value being as large as 7 is the *complementary event* of having all 100 counts be smaller than or equal to 6. Two complementary events have probabilities that sum to 1. Because the positions are supposed to be independent, we can now do the computation:

$$\begin{aligned} P(x_{(100)} \geqslant 7) &= 1 - P(x_{(100)} \leqslant 6) \\ &= 1 - P(x_{(1)} \leqslant 6) \times P(x_{(2)} \leqslant 6) \times \cdots \times P(x_{(100)} \leqslant 6) \\ &= 1 - P(x_1 \leqslant 6) \times P(x_2 \leqslant 6) \times \cdots \times P(x_{100} \leqslant 6) \\ &= 1 - \prod_{i=1}^{100} P(x_i \leqslant 6). \end{aligned}$$

The notation $\prod$ is just a compact way of writing the product of a series of terms, analogous to $\sum$ for sums.

Because we suppose these 100 events are independent, we can use our result from (1.1) above:

$$\prod_{i=1}^{100} P(x_i \leqslant 6) = (P(x_i \leqslant 6))^{100} = (1-\epsilon)^{100}.$$

Actually calculating the numbers

We could just let R compute the value of this number, $(1-\epsilon)^{100}$. For those interested in how such calculations can be simplified through approximation, we give some details here. These can be skipped on a first reading.

We recall from above that $\epsilon \simeq 10^{-6}$ is much smaller than 1. To compute the value of $(1-\epsilon)^{100}$ approximately, we can use the binomial theorem and drop all "higher order" terms of ϵ, i.e., all terms with $\epsilon^2, \epsilon^3, \ldots$, because they are negligibly small compared to the remaining ("leading") terms:

$$(1-\epsilon)^n = \sum_{k=0}^{n} \binom{n}{k} 1^{n-k} (-\epsilon)^k = 1 - n\epsilon + \binom{n}{2}\epsilon^2 - \binom{n}{3}\epsilon^3 + \ldots \simeq 1 - n\epsilon \simeq 1 - 10^{-4}.$$

Another, equivalent, route uses the approximation $e^{-\epsilon} \simeq 1-\epsilon$, which is the same as $\log(1-\epsilon) \simeq -\epsilon$. Hence

$$(1-\epsilon)^{100} = e^{\log\left((1-\epsilon)^{100}\right)} = e^{100\log(1-\epsilon)} \simeq e^{-100\epsilon} \simeq e^{-10^{-4}} \simeq 1 - 10^{-4}.$$

Thus the correct probability of seeing a number of hits as large as or larger than 7 in the 100 positions, if there is no epitope, is about 100 times the probability we wrongly calculated previously.

Both computed probabilities 10^{-6} and 10^{-4} are smaller than standard significance thresholds (say, 0.05, 0.01 or 0.001). The decision to reject the null hypothesis of no epitope would have been the same. However, if one has to stand up in court and defend the p-value to eight significant digits, as in some forensic court cases,[12] that is another matter. The adjusted p-value that takes into account the multiplicity of the test is the one that should be reported, and we will return to this important issue in Chapter 6.

[12] This occurred in the examination of the forensic evidence in the O.J. Simpson case.

Computing probabilities by simulation

In the case we just saw, the theoretical probability calculation was simple and we could figure out the result by an explicit calculation. In practice, things tend to be more complicated, and we do better to compute our probabilities using the **Monte Carlo** method: a computer simulation based on our generative model that finds the probabilities of the events we're interested in. The intuition here is to generate Figure 1.7 again and again, and observe how often the biggest spike is 7 or larger.

We generate 100,000 instances of picking the maximum from 100 Poisson distributed numbers.

```
maxes = replicate(100000, {
  max(rpois(100, 0.5))
})
table(maxes)
```

```
## maxes
##     1     2     3     4     5     6     7     9
##     7 23028 60840 14364  1604   141    15     1
```

We'll often use Monte Carlo simulations such as these instead of analytical calculations.

In 16 of 100,000 trials, the maximum was 7 or larger. This gives the following approximation for $P(X_{\max} \geqslant 7)$.

```
mean( maxes >= 7 )
## [1] 0.00016
```

The expression `maxes >= 7` evaluates into a logical vector of the same length as `maxes`, but with values of `TRUE` and `FALSE`. Applying the function `mean` to it converts that vector into 0s and 1s, and the result of the computation is the fraction of 1s, which is the same as the fraction of `TRUE`s.

which more or less agrees with our theoretical calculation. We already see one of the potential limitations of Monte Carlo simulations: the "granularity" of the simulation result is determined by the inverse of the number of simulations (100,000) and so will be around 10^{-5}. Any estimated probability cannot be more precise than this granularity, and indeed the precision of our estimate will be a few multiples of that. Everything we have done up to now is possible only because we know the false positive rate per position, we know the number of patients assayed and the length of the protein, we suppose we have identically distributed independent draws from the model, and there are no unknown parameters. This is an example of **probability or generative modeling**: all the parameters are known and the mathematical theory allows us to work by **deduction** in a *top-down* fashion.

We postulated the Poisson distribution for the noise, pretending we knew all the parameters, and were able to conclude through mathematical deduction.

If instead we are in the more realistic situation of knowing the number of patients and the length of the proteins, but don't know the distribution of the data, then we have to use **statistical modeling**. This approach will be developed in Chapter 2. We will see that if we have only the data to start with, we first need to **fit** a reasonable distribution to describe it. However, before we get to this harder problem, let's extend our knowledge of discrete distributions to more than binary success-or-failure outcomes.

1.4 Multinomial distributions: the case of DNA

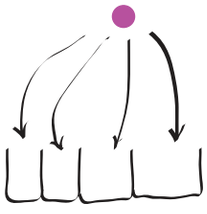

Figure 1.9: The boxes represent four outcomes or levels of a discrete *categorical* variable. The box on the right represents the most likely outcome.

More than two outcomes. When modeling four possible outcomes, as for instance the boxes in Figure 1.9 or when studying counts of the four nucleotides `A`, `C`, `G` and `T`, we need to extend the binomial model.

Recall that when using the binomial, we can consider unequal probabilities for the two outcomes by assigning a probability $p = P(1) = p_1$ to the outcome 1 and $1-p = P(0) = p_0$ to the outcome 0. When there are more than two possible outcomes, say `A`, `C`, `G` and `T`, we can think of throwing balls into boxes of different sizes corresponding to different probabilities, and we can label these probabilities p_A, p_C, p_G, p_T. Just as in the binomial case, the sum of the probabilities of all possible outcomes is 1, that is, $p_A + p_C + p_G + p_T = 1$.

You are secretly meeting a continuous distribution here, the *uniform distribution*: `runif`.

► **Task** Experiment with the random number generator that generates all possible numbers between 0 and 1 through the function called `runif`. Use it to generate a random variable with four levels (A, C, G, T) where $p_A = \frac{1}{8}, p_C = \frac{3}{8}, p_G = \frac{3}{8}, p_T = \frac{1}{8}$. ◄

Mathematical formulation. Multinomial distributions are the most important models for tallying counts, and R uses a general formula to compute the probability of a