3 Exploring probability distributions and conditional probabilities

This chapter covers

- Probability distributions
- Probability computations
- Conditional probabilities

In statistics and especially probability theory, understanding the behavior of random variables is just the beginning of a fascinating journey into the worlds of uncertainty and prediction. As we dive deeper into this subject, we arrive at a crucial inflection point: the exploration of probability distributions and conditional probability computations.

This chapter builds on the foundation established in chapter 2, where we introduced the concept of random variables and their properties. With this groundwork in place, we now embark on an in-depth exploration of four common probability distributions: normal, binomial, uniform, and Poisson.

Random variables serve as the bedrock on which probability distributions are constructed, allowing us to model and analyze the likelihood of various outcomes in a systematic manner. By examining specific probability distributions such as the normal distribution, which describes continuous phenomena like heights and

weights, and the binomial distribution, which deals with discrete events like coin flips, we gain a deeper appreciation for the diverse ways in which uncertainty manifests itself in the real world.

Furthermore, this chapter not only focuses on describing probability distributions but also explores the practical aspect of probability computations, including a final discussion around understanding and applying conditional probabilities. We will therefore study the interplay between prior and future events, thereby offering a more nuanced understanding of uncertainty and inference. When you reach the end of this journey, you will not only have a solid grasp of common probability distributions, but also the tools and techniques to navigate complex probabilistic scenarios with confidence and clarity.

3.1 Probability distributions

Probability distributions are foundational concepts in probability theory and statistics; they provide systematic methods and graphical techniques to quantify or illustrate the likelihood of different outcomes or events in the haze of uncertain and random phenomena. Understanding probability distributions is imperative because they form the backbone of statistical inference and decision-making processes in various fields, including finance, engineering, biology, and the social sciences. Mastery of probability distributions makes it possible to not only analyze and model complex real-world phenomena but also assess risk, forecast outcomes, and optimize strategies.

Our plan is to explore the four probability distributions—normal, binomial, uniform, and Poisson—that are most common when practicing data science. Each of these probability distributions assumes a unique set of properties and is therefore "designed" to handle specific, and different, types of random variables and phenomena. Our goal is to provide the twin benefits of theoretical understanding mixed with practical applications so as to establish a solid footing for effective probability analysis.

3.1.1 Normal distribution

The normal distribution—sometimes called a Gaussian distribution in honor of the mathematician Carl Friedrich Gauss, who contributed significantly to its study and application—is best characterized by its bell-like shape. We caught a glimpse of the normal distribution in chapter 2 when introducing continuous random variables. Many real-world phenomena follow the normal distribution, including heights and weights of adults, IQ scores, measurement errors, test scores, weights of products, residuals in regression analysis (see chapter 4), and daily stock returns.

PROPERTIES

The probability density function (PDF) of the normal distribution—that is, the *relative* likelihood of observing a continuous random variable at a particular value—is given by the following equation:

$$(x|\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \times e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where

- $f(x|\mu, \sigma)$ represents the PDF of a given value x, mean μ, and standard deviation σ.
- μ (mu) is the mean, or average, of the distribution.
- σ (sigma) is the standard deviation, which measures the spread or dispersion of the distribution.
- e is the base of the natural logarithm, approximately equal to 2.72.
- π is the mathematical constant pi, approximately equal to 3.14.

Remember an important takeaway from the previous chapter: *the PDF isn't defined by a single equation; rather, it varies depending on the specific distribution under consideration.* So, this specific probability density equation applies exclusively to the normal distribution.

Let's examine, by visual inspection, a 2 × 2 grid of normal distributions where the mean is held constant at 0 but the standard deviation (abbreviated Std Dev) equals a range of values between 5 and 20 (see figure 3.1).

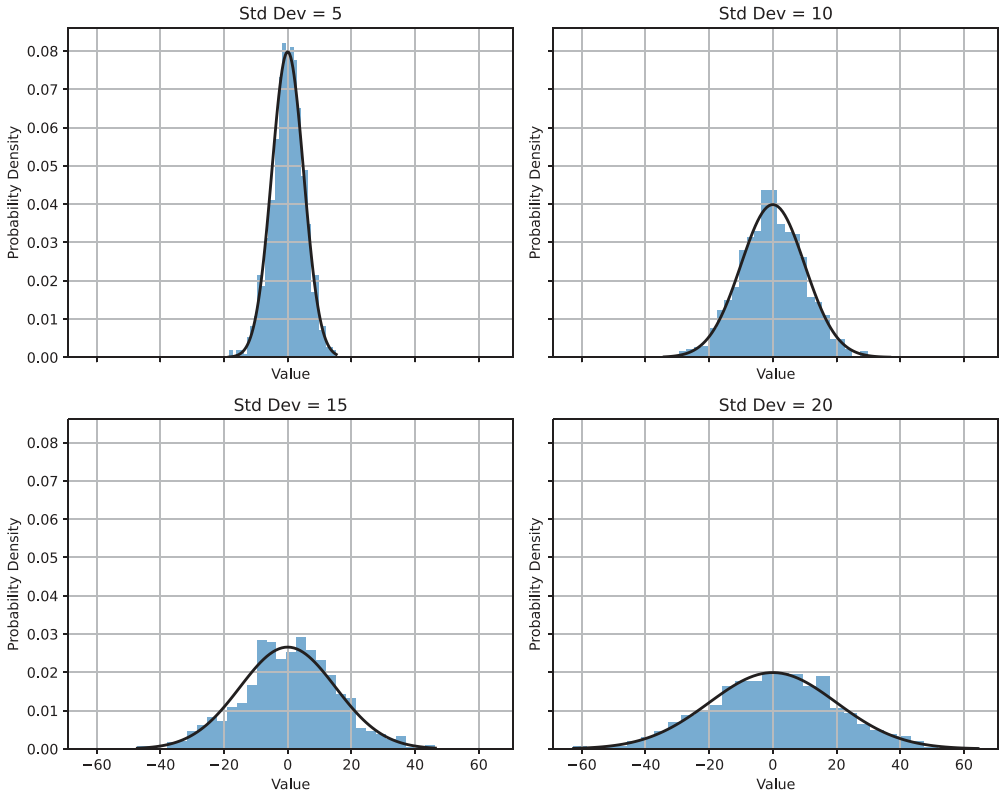

Figure 3.1 A 2 × 2 grid of normal distributions where each plot shares the same *x*-axis and *y*-axis scales. The mean equals 0 throughout, and the standard deviation equals 5, 10, 15, or 20. Increases in the standard deviation translate to greater dispersion from the mean, flatter distributions, and wider tails. In other words, the PDF becomes more dispersed across a larger range of values, and the distribution is broader and less peaked around the mean. The bell-like shape applies throughout, however, where the values are distributed symmetrically around the mean.

This grid of histograms reveals several important features about the normal distribution:

- The distribution is symmetrical around its center, or mean. This is to say that the probability of observing a value greater than the mean is equal to 0.5, and the probability of observing another value less than the mean is also equal to 0.5.
- Due to its symmetrical and unimodal nature, the normal distribution takes on a bell-shaped curve, with a single peak at the mean. This shape indicates that values closer to the mean are more likely to be observed than values further away.
- Assuming a normal distribution, a pair of parameters—the mean and standard deviation—are key. The mean is equal to the sum of all values in a numeric data series divided by the record count; when the same data is normally distributed, the mean is at the center of the distribution. The standard deviation, meanwhile, shapes the distribution by determining its spread around the mean. Although the normal distribution inherently assumes a bell-like shape, the standard deviation indicates the extent to which the distribution is thin or flat. The standard deviation can be estimated by dividing 0.399—which represents the maximum probability density function from a standard normal distribution, where the mean is 0 and the standard deviation is 1—by the maximum PDF associated with the normal distribution under observation. So, for instance, 0.399 divided by .080 returns an estimated standard deviation equal to 4.99 (see the upper-left histogram, where the actual standard deviation is equal to 5).
- The PDF achieves its maximum when the value for x is the mean, regardless of the standard deviation.

When working with the normal distribution, it's important to always remember what is commonly called the *68–95–99 rule*. This rule, also known as the *three-sigma rule* (for three standard deviations), states that approximately 68% of the data in the normal distribution is within one standard deviation of the mean, that approximately 95% of the data is within two standard deviations of the mean, and up to 99.7% of the data is within three standard deviations of the mean. It's important to be aware of this rule because it provides valuable, and fairly precise, insights into normally distributed data—helping analysts quickly assess how likely an outcome is, estimate the probability of extreme values, and set reasonable expectations around what constitutes normal variation. These characteristics make the normal distribution a popular assumption in many fields, as it offers predictable guidelines for interpreting data variability.

Let's repeatedly test the 68–95–99 rule by generating 10,000 random samples from a normal distribution with mean `0` and standard deviation `1` using the `np.random.normal()` method from the `numpy` library:

```
>>> mean = 0
>>> std_dev = 1

>>> import numpy as np
>>> samples = np.random.normal(mean, std_dev, 10000)
```

In the next line of Python code,

- `np.abs(samples - mean)` calculates the absolute differences between each sample in the `samples` array and the mean. We want the absolute differences, rather than a mix of positive and negative differences, to prevent the final calculations from canceling each other.
- `< std_dev` checks whether each absolute difference is less than the standard deviation; it creates a Boolean array where `True` indicates the absolute difference is less than the standard deviation and `False` otherwise.
- `np.sum(…)` sums the Boolean array where `True` is equivalent to 1 and `False` is equivalent to 0, thereby providing a count of samples within one standard deviation of the mean.
- `within_1_std` is a variable that stores the results.

In summary, the following line of code calculates the number of samples from 10,000 random samples that are within one standard deviation of the mean:

```
>>> within_1_std = np.sum(np.abs(samples - mean) < std_dev)
```

The next two lines of code calculate the number of samples that are within two or three standard deviations of the mean.

```
>>> within_2_std = np.sum(np.abs(samples - mean) < 2 * std_dev)
>>> within_3_std = np.sum(np.abs(samples - mean) < 3 * std_dev)
```

And from the next snippet of code, we get the percentage of samples within one, two, or three standard deviations of the mean. Percentages are obtained by dividing the sample counts by `100`; results are then printed as formatted strings:

```
>>> print(f'Percent within 1 standard deviation: {within_1_std / 100}%')
Percent within 1 standard deviation: 66.89%

>>> print(f'Percent within 2 standard deviations: {within_2_std / 100}%')
Percent within 2 standard deviations: 95.13%

>>> print(f'Percent within 3 standard deviations: {within_3_std / 100}%')
Percent within 3 standard deviations: 99.69%
```

These results are obviously consistent with the 68–95–99 rule. But let's run the same code two more times and then compare the results:

```
>>> mean = 0
>>> std_dev = 1

>>> samples = np.random.normal(mean, std_dev, 10000)

>>> within_1_std = np.sum(np.abs(samples - mean) < std_dev)
>>> within_2_std = np.sum(np.abs(samples - mean) < 2 * std_dev)
>>> within_3_std = np.sum(np.abs(samples - mean) < 3 * std_dev)

>>> print(f'Percent within 1 standard deviation: {within_1_std / 100}%')
>>> print(f'Percent within 2 standard deviations: {within_2_std / 100}%')
```

```
>>> print(f'Percent within 3 standard deviations: {within_3_std / 100}%')
Percent within 1 standard deviation: 68.3%
Percent within 2 standard deviations: 95.45%
Percent within 3 standard deviations: 99.75%

>>> mean = 0
>>> std_dev = 1

>>> samples = np.random.normal(mean, std_dev, 10000)

>>> within_1_std = np.sum(np.abs(samples - mean) < std_dev)
>>> within_2_std = np.sum(np.abs(samples - mean) < 2 * std_dev)
>>> within_3_std = np.sum(np.abs(samples - mean) < 3 * std_dev)

>>> print(f'Percent within 1 standard deviation: {within_1_std / 100}%')
>>> print(f'Percent within 2 standard deviations: {within_2_std / 100}%')
>>> print(f'Percent within 3 standard deviations: {within_3_std / 100}%')
Percent within 1 standard deviation: 68.31%
Percent within 2 standard deviations: 95.46%
Percent within 3 standard deviations: 99.78%
```

The results may never be exactly alike, no matter how many times we run and rerun the same code. But they won't ever deviate in any meaningful way from the 68–95–99 rule.

PROBABILITY DENSITY FUNCTION

To effectively compute (and explain) probabilities from the normal distribution, it's essential to have a visual reference. Rather than reusing one or more of the normal distributions from our 2×2 grid, we'll instead draw a standard normal distribution, where the mean equals 0 and the standard deviation equals 1 (see figure 3.2). In the standard normal distribution, the raw data has been transformed, or standardized, so that the values along the x axis represent the number of standard deviations they are below or above the mean.

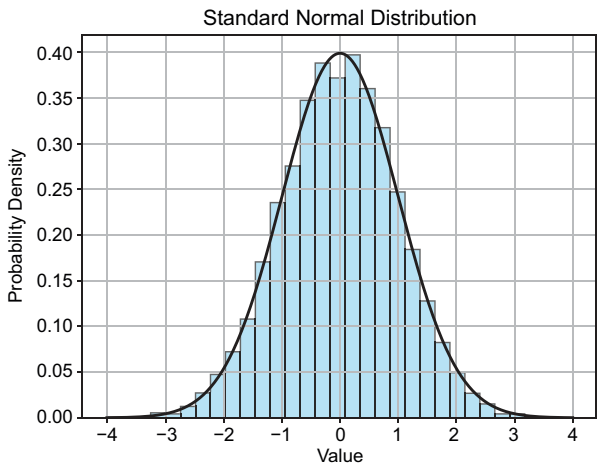

Figure 3.2 A standard normal distribution is a normal distribution where the mean is equal to 0, the standard deviation is equal to 1, and the values have been standardized from their raw form. The PDF typically peaks at or around 0.399.

We want to find the PDF of a transformed continuous random variable from the standard normal distribution. When the mean equals 0 and the standard deviation equals 1, the random variable is actually denoted as z rather than x (why will be clear in a bit).

There are two ways of going about this in Python. One way is to plug values for `mu` (the mean), `sigma` (the standard deviation), and `z` (the continuous random variable that has since been standardized from its raw form) into an arithmetic operation based on the PDF.

The first step is to initialize these three variables:

```
>>> mu = 0
>>> sigma = 1
>>> z = 1
```

The second step is to write and execute a snippet of code that calculates the PDF for `z` when the mean equals 0 and the standard deviation equals 1:

```
>>> pdf_value = ((1 / (sigma * math.sqrt(2 * math.pi))) * \
>>>              math.exp(-((z - mu) ** 2) / (2 * sigma ** 2)))
```

And the third and final step is to print the results:

```
>>> print(f'PDF at {z} = {pdf_value}')
PDF at 1 = 0.24197072451914337
```

Another (easier) way is to pass the same variables to the `norm.pdf()` method from the `norm` module in the `scipy` library:

```
>>> mu = 0
>>> sigma = 1
>>> z = 1

>>> from scipy.stats import norm
>>> pdf_value = norm.pdf(z, loc = mu, scale = sigma)
>>> print(f'PDF at {z} = {pdf_value}')
PDF at 1 = 0.24197072451914337
```

A PDF equal to 24% is relatively high when you consider that the maximum PDF for the standard normal distribution is just less than 40%. To be clear, the PDF does *not* represent the probability of observing a continuous random variable at a particular standardized value. So, it would be incorrect to conclude that when the mean is equal to 0 and the standard deviation is equal to 1, the probability of observing a continuous random variable standardized to 1 is approximately 24%. Rather, it would be correct to infer that when z equals 1, we can expect a higher density of probabilities at that value compared to other values for z in the same distribution. However, there are methods we can use to get a true probability.

COMPUTING PROBABILITIES

One way—maybe the old-school way—is to use a z-score table. If the raw data hasn't yet been transformed, we first need to convert the values to their corresponding

z-scores. A *z-score*, also known as a *standard score*, is a statistical measure that indicates how many standard deviations a particular value is from the mean. It is calculated using the following formula:

$$z = \frac{x - \mu}{\sigma}$$

where

- z is the z-score.
- x is an individual data point.
- μ is the mean of the distribution.
- σ is the standard deviation of the distribution.

A positive z-score indicates that the data point, or a particular value, is above the mean, whereas a negative z-score indicates that it is below the mean. The magnitude of the z-score reflects how many standard deviations the value is from the mean—a larger absolute value means the point is farther from the average. By converting raw data to standardized scores, we can then compare values from different data sets or even different variables from the same data set that were originally on different scales. Standardized scores are commonly used in hypothesis testing, statistical analysis, and data normalization. From a data series that follows the standard normal distribution, z-scores are used to determine probabilities.

If we set z to equal 1, it signifies that a specific raw value, on standardization, corresponds to a z-score of 1, thereby indicating that it is one standard deviation above the mean of the distribution. With the z-score determined, we can then obtain the probability associated with it from a z-score table (see figure 3.3).

z	.00	.01	.02	.03	.04	.05	.06	.07	.08	.09
0.0	.5000	.5040	.5080	.5120	.5160	.5199	.5239	.5279	.5319	.5359
0.1	.5398	.5438	.5478	.5517	.5557	.5596	.5636	.5675	.5714	.5753
0.2	.5793	.5832	.5871	.5910	.5948	.5987	.6026	.6064	.6103	.6141
0.3	.6179	.6217	.6255	.6293	.6331	.6368	.6406	.6443	.6480	.6517
0.4	.6554	.6591	.6628	.6664	.6700	.6736	.6772	.6808	.6844	.6879
0.5	.6915	.6950	.6985	.7019	.7054	.7088	.7123	.7157	.7190	.7224
0.6	.7257	.7291	.7324	.7357	.7389	.7422	.7454	.7486	.7517	.7549
0.7	.7580	.7611	.7642	.7673	.7704	.7734	.7764	.7794	.7823	.7852
0.8	.7881	.7910	.7939	.7967	.7995	.8023	.8051	.8078	.8106	.8133
0.9	.8159	.8186	.8212	.8238	.8264	.8289	.8315	.8340	.8365	.8389
1.0	.8413	.8438	.8461	.8485	.8508	.8531	.8554	.8577	.8599	.8621
1.1	.8643	.8665	.8686	.8708	.8729	.8749	.8770	.8790	.8810	.8830
1.2	.8849	.8869	.8888	.8907	.8925	.8944	.8962	.8980	.8997	.9015
1.3	.9032	.9049	.9066	.9082	.9099	.9115	.9131	.9147	.9162	.9177
1.4	.9192	.9207	.9222	.9236	.9251	.9265	.9279	.9292	.9306	.9319
1.5	.9332	.9345	.9357	.9370	.9382	.9394	.9406	.9418	.9429	.9441
1.6	.9452	.9463	.9474	.9484	.9495	.9505	.9515	.9525	.9535	.9545
1.7	.9554	.9564	.9573	.9582	.9591	.9599	.9608	.9616	.9625	.9633
1.8	.9641	.9649	.9656	.9664	.9671	.9678	.9686	.9693	.9699	.9706
1.9	.9713	.9719	.9726	.9732	.9738	.9744	.9750	.9756	.9761	.9767
2.0	.9772	.9778	.9783	.9788	.9793	.9798	.9803	.9808	.9812	.9817

Figure 3.3 The top of a typical z-score table. The probability (or area to the right of a particular value that has been standardized) is found where the integer and remaining fractional parts of the value intersect.

The probability is where the first two digits of the z-score, located in the far-left column of the table, intersect with the next two digits of the z-score, located along the top row. A z-score of 1, for instance, is the equivalent of 1.000. The corresponding probability is therefore equal to .8413, or 84.13%, which is where 1.0 and .00 intersect. This figure also represents the area to the left of the z-score in the standard normal distribution, where the area left of the mean equals 0.5 and the area between the mean and one standard deviation above it equals .3413—these probabilities (or areas) when added together sum to 84.13%.

To further explain, imagine a statistics class of 30 students taking their final exam. The exams are graded and the scores standardized. One student achieved a score one standard deviation above the class mean. That student thus scored in the 84th percentile, or performed better than 25 of the 30 students in the class—derived by multiplying 30 students times .8413 and rounding the product down to the nearest whole number.

Finally, we can use the z-score table to get the probability of a particular value falling within some range along the standard normal distribution. For instance, if we were curious about the probability of some value being equal to or greater than 1.000 but less than 1.500, we could subtract their respective probabilities (.9332 − .8413) and apply the difference (.0919) to get the probability (roughly 9%).

Another way is to write a short snippet of Python code. We initialize a z-score value of `1`; pass that to the `norm.cdf()` method, which calculates the cumulative probability (or area) under the standard normal distribution curve up to the value of one standard deviation above the mean and stores the result in a variable called `first_probability`; and then print the results in the form of a percentage by multiplying the fractional output by `100`:

```
>>> z = 1
>>> first_probability = norm.cdf(z)
>>> print(f'Probability (area): {first_probability * 100}%')
Probability (area): 84.1344746068543%
```

We can even use Python to get the probability (or area) between a pair of standardized scores, like 1.000 and 1.500.

First we make a second call to the `norm.cdf()` method, this time with `z` initialized to `1.5`. `norm.cdf()` calculates the cumulative probability (or area) up to 1.5 standard deviations above the distribution mean and stores the results in a variable called `second_probability`:

```
>>> z = 1.5
>>> second_probability = norm.cdf(z)
>>> print(f'Probability (area): {second_probability * 100}%')
Probability (area): 93.3192798731142%
```

Then we subtract `first_probability` from `second_probability` to get the probability (or area) of some value falling between the standardized scores of 1 and 1.5:

```
>>> range_probability = second_probability - first_probability
>>> print(f'Probability (area) between '
```

```
      f'1.0 and 1.5: {range_probability * 100}%')
Probability (area) between 1.0 and 1.5: 9.184805266259898%
```

From continuous random variables and the normal distribution, we now shift our focus to discrete outcomes and the binomial distribution. At first glance, this so-called shift may appear to be substantial, but in reality the normal distribution is the result of many continuous trials of the binomial distribution.

KEY TAKEAWAYS

The normal distribution, often referred to as the Gaussian distribution, is a continuous probability distribution characterized by its symmetric bell-shaped curve centered around the mean. This distribution is unique due to its symmetry and is entirely defined by its mean and standard deviation. The PDF of a normal distribution describes the relative likelihood of different outcomes, providing insights into the distribution's shape and spread without yielding actual probabilities. To compute actual probabilities from a *standard* normal distribution, we must use the cumulative distribution function, which can be done manually using a z-table or programmatically using Python methods. Demonstrating these computations both manually and programmatically is crucial for a thorough understanding of the underlying mechanics and ensures that we can accurately interpret and implement statistical analyses in various contexts.

3.1.2 Binomial distribution

A binomial experiment is a statistical experiment in which a binomial random variable represents the number of successes in a fixed number of repeatable trials, where the probability of success is consistent throughout. The binomial distribution is a probability distribution that describes the likelihood of observing a specific number of successes from a series of trials where each trial has only two potential outcomes: success or failure. The following conditions must prevail:

- The trials are independent. That means each trial is unaffected by previous trials and does not then influence the outcomes of subsequent trials. Independence between trials is a fundamental assumption in many statistical analyses and probability models, including the binomial distribution. For instance, when flipping a fair coin multiple times, each flip is considered an independent trial because the outcome of one flip has no bearing on the outcome of any other flip.
- Each trial can be classified as either success or failure, where p equals the probability of success and $1 - p$ equals the probability of failure.
- The number of trials is fixed and determined in advance.
- The probability of success is consistent across trials. Thus, the trials must not only be independent, but they must also be alike in order to hold the probability of success constant.

Quality control inspections that result in pass or fail, medical tests that return positive or negative results, surveys featuring binary responses—these are just a few examples of binomial distributions in everyday life. Binomial distributions are important to understand because they provide a framework for quantifying probabilities of binary

outcomes in repeated trials across various domains, including manufacturing, healthcare, and market research.

PROPERTIES

The probability mass function (PMF) is a function that assigns probabilities to discrete random variables that represent the likelihood of each possible outcome. In the context of binomial distributions, the PMF calculates the probability of observing a specific number of successes (or failures) in a fixed number of independent trials, each with two possible outcomes.

The PMF for a binomial distribution is given by the following equation:

$$p\,(x = k) = \binom{n}{k} \times p^k \times (1 - p)^{n-k}$$

where

- $p(x = k)$ represents the probability of observing k successes in n trials.
- n over k is the binomial coefficient, also known as "n choose k," which represents the number of ways to choose k successes from n trials.
- p is the probability of success on any individual trial.
- $1 - p$ represents the probability of failure on any individual trial.
- n is the total number of trials.
- k is the number of successes.

The PMF for a binomial distribution is a property that characterizes the distribution and allows us to determine the probabilities associated with different numbers of successes. By evaluating the PMF at different values of k, we can obtain the probability distribution for the binomial random variable, thereby providing insights into the likelihood of various outcomes in a binomial experiment.

Let's examine two pairs of plotted binomial distributions (see figures 3.4 and 3.5), where

- The number of trials equals 20 throughout.
- Therefore, the number of potential successes ranges from 0 to 20.
- The probability of success iterates from 0.20 and 0.50 in the first two plots to 0.75 and 0.90 in the next pair.

We initialize `n`, `k`, and `p` like so: the `np.arrange()` method creates an array of evenly spaced values starting at `0` and ending at `n` (inclusive), where each value is incremented by `1`. The second argument passed to the `np.arrange()` method, `n + 1`, ensures that the endpoint `n` is included in the array.

For now, `p` equals a list containing just two probabilities of success, because we intend to apply different methods to create each pair of plots:

```
>>> n = 20
>>> k = np.arange(0, n + 1)
>>> p = [0.20, 0.50]
```

These values are passed to a snippet of Python code that mirrors the PMF equation; it specifically calculates the PMF for a binomial distribution for each possible number of successes `j` in `n` trials, given the parameters `n` and `p[i]` and the array `k`.

Here's the complete breakdown of the code:

- `pmf = [...for j in k]` is a list comprehension that iterates over each value `j` in the array `k`, which represents the possible number of successes in `n` trials.
- `np.math.comb(n, j)` calculates the binomial coefficient, which represents the number of ways to choose `j` successes from `n` trials.
- `(p[i] ** j)` calculates the probability of obtaining `j` successes, raised to the power of `j`, using the value `p[i]`, where `p` is assumed to be an array or list of probabilities indexed by `i`.
- `((1 - p[i]) ** (n - j))` calculates the probability of getting `n – j` failures, raised to the power of `n – j`, assuming failure probability `1 – p[i]` for each trial.

And here's the code:

```
>>> pmf = [(np.math.comb(n, j) * (p[i] ** j) * \
>>>         ((1 - p[i]) ** (n - j))) for j in k]
```

This, when combined with a chunk of Matplotlib code, returns the graphical output shown in figure 3.4. (Matplotlib is a popular Python plotting library that we began using in the prior chapter.)

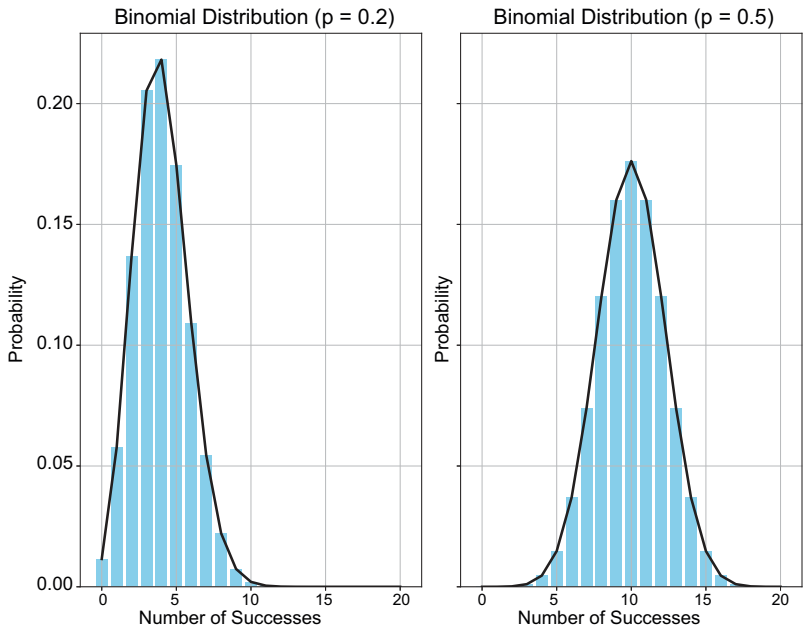

Figure 3.4 A first pair of binomial distributions where the probability of success equals 0.20 (on the left) and 0.50 (on the right), given 20 independent trials. Although the distributions are binomial, the data is nonetheless distributed normally.

For our next pair of plots, instead of supplying the number of trials, the count of potential successes, and a fresh list of success probabilities to a single arithmetic operation, we will make a call to the `binom.pmf()` method for this purpose. But first, we reinitialize `n` and `k` and assign new probabilities to `p`:

```
>>> n = 20
>>> k = np.arange(0, n + 1)
>>> p = [0.75, 0.90]
```

The `binom.pmf()` method, which requires the `binom` module from `scipy.stats`, twice calculates the PMF for a binomial distribution—in this instance, the number of trials again equals `20`, the number of potential successes therefore also equals `20`, and the probabilities of success now equal `0.75` and then `0.90`:

```
>>> from scipy.stats import binom
>>> pmf = binom.pmf(k, n, p[i])
```

When the `binom.pmf()` method is mixed with another chunk of Matplotlib code, we get the plots shown in figure 3.5.

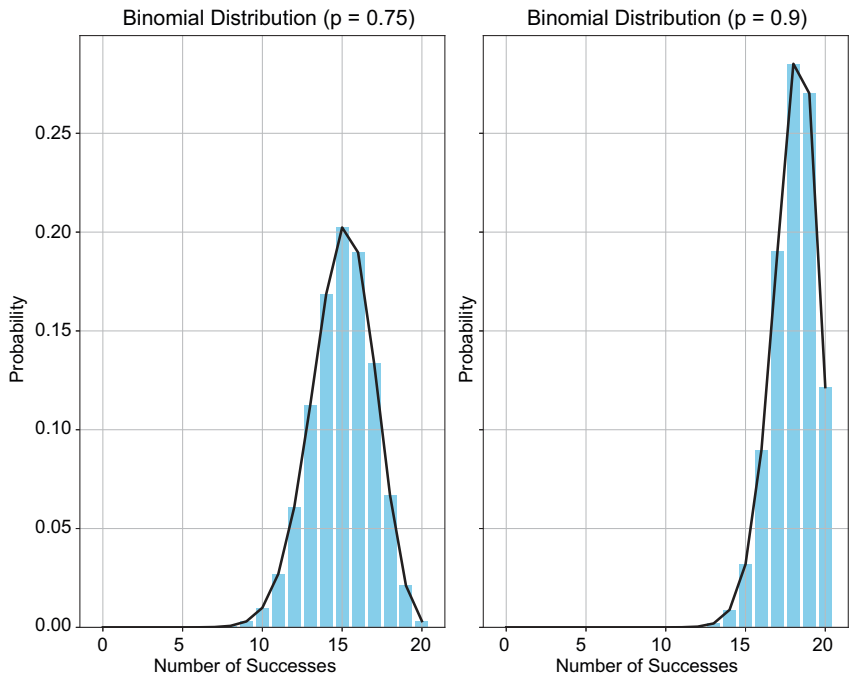

Figure 3.5 A second pair of binomial distributions where the probability of success equals 0.75 (on the left) and 0.90 (on the right), given 20 independent trials. Regardless of the probability of success, although the distribution mean shifts as a result, the binomial distributions maintain their normal distribution look.

The primary observation from examining these plots is that binomial outcomes exhibit characteristics resembling a normal distribution. Variations in the probability of success bring about corresponding changes in the distribution mean, but the distributions nevertheless consistently retain their symmetrical shapes, with binomial outcomes equally centered around the mean. Hence, although it may seem somewhat evident, it's worth emphasizing that the distribution mean is derived by multiplying the fixed trial count by the probability of success:

$$\mu = np$$

where

- μ is the binomial distribution mean.
- n is the number of independent trials.
- p is the probability of success.

So, for instance, when the number of independent trials equals 20 and the probability of success in each of them equals 0.20, the mean number of successes equals $20 \times 0.2 = 4$. This coincides with the point where the PMF reaches its maximum value.

The standard deviation of a binomial distribution is derived by taking the square root of the product of the distribution mean and the probability of failure:

$$\sigma = \sqrt{np\,(1-p)}$$

where

- σ represents the standard deviation.
- n is the number of independent trials.
- p is the probability of success.
- $1 - p$ is the inverse of p and therefore equals the number of failures.

Therefore, the standard deviation is relatively low when the probabilities of success are either low or high; alternatively, the standard deviation is relatively high, or higher, when the probability of success equals something like 0.50 or 0.75. Another glimpse at our histograms illustrates what we can derive mathematically: low and high probabilities of success, like 0.20 and 0.90, respectively, translate to thinner and taller binomial distributions, whereas "average" probabilities of success render just the opposite.

COMPUTING PROBABILITIES

Let's say we want to know the probability of getting heads 7 times from 20 flips of a fair coin. Or maybe we want to know the probability of discovering 7 defective products from a batch of 20. Although the probabilities of these outcomes might be very different, both are nevertheless examples of a binomial experiment.

There are at least two ways to derive the probability of a specific binomial outcome in Python. We'll demonstrate the first way by simulating 20 coin flips. So, we initialize `n` to equal `20`, `k` to equal `7`, and `p` to equal `0.5`:

```
>>> n = 20
>>> k = 7
>>> p = 0.5
```

Then we calculate the binomial coefficient by applying the formula for counting combinations without replacement:

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

The binomial coefficient and the number of combinations without replacement share the same formula because both calculate the number of ways to choose a specific number of objects from a fixed set without considering the order of selection. We replicate this formula in Python by passing our three parameters to the `math.factorial()` method:

```
>>> binomial_coefficient = (math.factorial(n) / \
>>>                         (math.factorial(k) * math.factorial(n - k)))
>>> print(binomial_coefficient)
77520.0
```

Finally, we obtain the probability of getting heads exactly 7 times from 20 coin flips by inputting this result into the PMF equation:

```
>>> probability = binomial_coefficient * (p ** k) * ((1 - p) ** (n - k))
>>> print(f'Probability: {probability * 100}%')
Probability: 7.39288330078125%
```

This result perfectly corresponds to the binomial distribution displayed on the right side of figure 3.4, where the probability of success was set at 0.5.

The more efficient way is to instead pass `n`, `k`, and `p` to the `binom.pmf()` method. Let's change `p` to equal `0.2` but keep `n` and `k` constant:

```
>>> n = 20
>>> k = 7
>>> p = 0.2
```

This time around, we want the probability of observing exactly 7 successes in 20 independent trials when the probability of success equals 0.2. The `binom.pmf()` method automatically calculates the binomial coefficient and subsequently applies the result to the PMF equation:

```
>>> probability = binom.pmf(k, n, p)
>>> print(f"Probability: {probability * 100}%")
Probability: 5.454985048652523%
```

This result is perfectly in sync with the binomial distribution displayed on the left side of figure 3.4, where the probability of success was established at 0.2.

Having explored the intricacies of binomial distributions and their application in modeling discrete random phenomena, we now direct our attention to uniform distributions as the next step in our exploration of probability analysis.

KEY TAKEAWAYS

The binomial distribution is a discrete probability distribution that models the number of successes in a fixed number of independent trials, each with the same probability of success. It is defined by two parameters: the number of trials and the probability of success in each trial. The binomial PMF describes the probability of obtaining a specific number of successes, providing insight into the distribution's shape and the likelihood of different outcomes. To compute actual probabilities from a binomial distribution, we typically use the binomial formula or use Python methods for computational efficiency. Once again, understanding how to calculate these probabilities both manually and programmatically is essential for gaining a solid grasp of the binomial distribution's behavior and applying it correctly in statistical analysis, especially in scenarios involving binary outcomes like pass/fail or yes/no events.

3.1.3 Discrete uniform distribution

The discrete uniform distribution is a probability distribution with a constant probability; that is, every value, or every discrete random variable, within a predefined range has an equal probability of occurring. When flipping a fair coin, for instance, heads and tails both have probabilities equal to 50%; or when rolling a fair six-sided die, each possible outcome has an equal probability of 1/6, or 16.7%. Consequently, the discrete uniform distribution is sometimes called a *rectangular distribution* because, when plotted, the distribution takes on a rectangular shape. It is typically denoted as $X \sim U(a, b)$, where a and b represent, respectively, the lower and upper bounds of possible values.

Thus:

- Each outcome within the range $[a, b]$ must have the same probability of occurrence.
- The distribution only applies to a finite set of values within the specified range.
- The sum of all probabilities within the range must therefore equal 1.
- Each outcome is independent of the others.

The discrete random distribution applies to a wide array of contexts, ranging from modeling coin flips, dice rolls, and card selections to generating random numbers for simulations, test and control assignments, and numerical experiments; creating random keys and nonces (numbers used once) for cryptographic applications; and randomly selecting items from a fixed set or population. This is why discrete uniform distributions, despite their relative simplicity, still deserve our attention.

PROPERTIES

The PMF of a discrete uniform distribution is defined as

$$f(x) = \frac{1}{b - a + 1}$$

where

- a represents the lower bound of the distribution.
- b represents the upper bound of the distribution.

- $f(x)$ is the PMF, representing the probability of obtaining a specific value x within the range $[a, b]$.

Any and all specific values must be discrete random variables, or a countable set of distinct values, typically integers or whole numbers. Otherwise, the PMF simply defines the lower and upper boundaries of possible values and assigns an equal probability of occurrence to all values equal to or greater than a and equal to or less than b. So, for instance, if a is equal to 1 and b is equal to 6, the denominator, which represents the range of possible outcomes, is equal to 6. By adding 1 to the difference between b and a, the PMF ensures that the range of possible outcomes is inclusive of the lower and upper bounds. When we assign a to 1 and b to 6 and embed the PMF equation in a chunk of Matplotlib code, we obtain a typical plot illustrating the discrete uniform distribution (see figure 3.6).

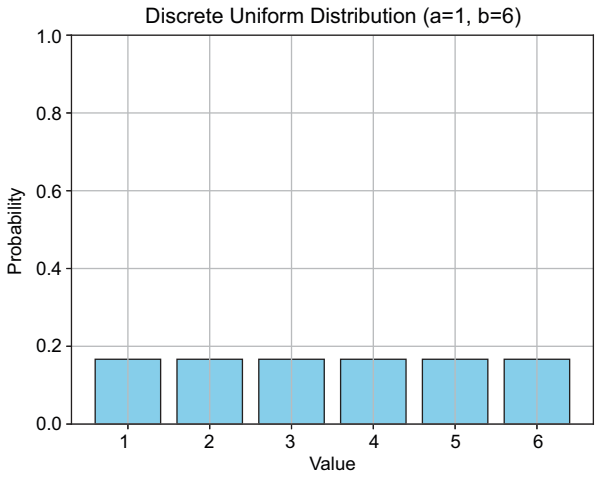

Figure 3.6 A typical discrete uniform distribution, where each discrete random variable has the same probability of occurrence. It doesn't matter what the lower and upper bounds are; the discrete uniform distribution will always assume this rectangular shape.

Why the discrete uniform distribution is sometimes referred to as the rectangular distribution is now evident. Regardless of the two values designated as a and b, respectively, the discrete uniform distribution consistently maintains this rectangular shape.

Continuous uniform distribution

There are actually two types of uniform distributions. In addition to the discrete uniform distribution, there is also a continuous uniform distribution, which represents a scenario where any value within a specified range has an equal probability of occurring. Unlike the discrete uniform distribution, which deals with countable outcomes (typically integers or whole numbers only), the continuous uniform distribution applies to an uncountably infinite set of values within the range (which might include fractional numbers). However, our sole focus on the discrete type of uniform distribution makes sense due to its simplicity in modeling scenarios with finite and equally probable outcomes, offering clarity and practicality in various real-world applications.

It's important to emphasize a couple of related points. First, probabilities do not guarantee specific outcomes; in other words, when rolling a fair six-sided die six times, it's improbable that each possible outcome will occur exactly once in those six attempts. When sample sizes are relatively low, actual outcomes will not necessarily align with theoretical distributions. And second, equal probabilities absolutely do not guarantee equal outcomes. We would anticipate convergence with larger sample sizes, such as 10,000 or more. However, even with such sample sizes, there is still no promise of achieving a distribution where each of, let's say, six discrete random variables is observed between 16% and 17% of the time.

COMPUTING PROBABILITIES

In Python, one approach to computing the probabilities of observing a range of discrete random variables is by writing an arithmetic operation based on the PMF. We start by defining the values for `a` and `b`, representing the lower and upper bounds of the range, respectively:

```
>>> a = 1
>>> b = 6
```

These assignments represent the parameters of the discrete uniform distribution.

Next, we write a line of code that evaluates the total number of possible outcomes within the predefined range, inclusive of both the lower and upper bounds, and stores the result in a variable called `total_outcomes`:

```
>>> total_outcomes = b - a + 1
```

Of course, we already know that when the lower and upper bounds are equal to `1` and `6`, there are six possible outcomes.

Then we create another variable, `pmf_values`, where

- `1/total_outcomes` computes the probability for each individual outcome, which is equal to the reciprocal of the total number of outcomes; this ensures that the probabilities sum to 1.
- `[1/total_outcomes] * total_outcomes` creates a list where the computed probability value is repeated `total_outcomes` times; this ensures that each outcome has the same probability value.

The list of six common probabilities is stored in the variable `pmf_values`:

```
>>> pmf_values = [1 / total_outcomes] * total_outcomes
```

And finally, we write a snippet of Python code that jointly iterates through each possible value within the specified range and the corresponding probability value stored in `pmf_values`. For each value, it prints the probability of that value occurring, formatted as a percentage.

Here's the breakdown:

- `zip(range(a, b + 1), pmf_values)` pairs each value from the range `a` to `b` with its corresponding probability value stored in the variable `pmf_values`.

- `for value, probability in...` iterates through each value and probability pair.
- `print(f'p(x = {value}) = {probability * 100}%')` prints each discrete random variable and its probability, formatted as a percentage. The probabilities are converted from decimals to percentages by multiplying the results by `100`.

And here's the snippet of code, followed by the results:

```
>>> for value, probability in zip(range(a, b + 1), pmf_values):
>>>     print(f'p(x = {value}) = {probability * 100}%')
p(x = 1) = 16.666666666666664%
p(x = 2) = 16.666666666666664%
p(x = 3) = 16.666666666666664%
p(x = 4) = 16.666666666666664%
p(x = 5) = 16.666666666666664%
p(x = 6) = 16.666666666666664%
```

But, of course, there is a Python method that simplifies this operation.

Once more, let's define the lower and upper bounds as `1` and `6`, respectively:

```
>>> a = 1
>>> b = 6
```

Then we create a discrete uniform distribution object, called `uniform_dist`, by making a call to the `stats.radint()` method from the `scipy` library.

```
>>> import scipy.stats as stats
>>> uniform_dist = stats.randint(a, b + 1)
```

The `stats.randint()` method constructs a discrete uniform distribution object, which generates random variates within a specified range. It's important to note that a *random variate*, distinct from a *random variable*, refers to a specified observed value of the random variable. The `stats.randint()` method requires two arguments: the lower and upper bounds, incremented by `1` to ensure inclusivity. The returned object facilitates various calculations, including PMFs.

Then we instruct Python to iterate over each value in the specified range, calculate the PMF for each value using `uniform_dist.pmf(value)`, and return the results as percentages rather than decimals. This operation requires the `uniform` module from `scipy.stats`:

```
>>> from scipy.stats import uniform
>>> for value in range(a, b + 1):
>>>     probability = uniform_dist.pmf(value)
>>>     print(f'p(x = {value}) = {probability * 100}%')
p(x = 1) = 16.666666666666664%
p(x = 2) = 16.666666666666664%
p(x = 3) = 16.666666666666664%
p(x = 4) = 16.666666666666664%
p(x = 5) = 16.666666666666664%
p(x = 6) = 16.666666666666664%
```

Just a different path to get the exact same results as before.

Last, but certainly not least, we have Poisson distributions—yet another fundamental concept in probability theory and statistics. Poisson distributions are used to model the frequency of events occurring within a fixed interval of time or space. This distribution is particularly useful in scenarios where events happen independently at a constant average rate and the probability of multiple occurrences within a short interval is low.

KEY TAKEAWAYS

The discrete uniform distribution is a simple yet pervasive probability distribution where each outcome in a finite set of equally likely outcomes has the same probability. Its straightforward nature makes it a fundamental concept in probability theory and statistics, and it's often used in real-world applications like games of chance, random sampling, and simulations. The equal likelihood of outcomes in a discrete uniform distribution provides a clear and intuitive framework for understanding randomness and fairness in various contexts.

3.1.4 Poisson distribution

The Poisson distribution is a discrete probability distribution used to model the likelihood of a specific number of events occurring within a fixed interval of time or space. Its shape varies based on the frequency of events, ranging from right-skewed to approaching normal as occurrences become more frequent. This means the following:

- Because Poisson distributions are inherently discrete, they are used exclusively to model countable events.
- This entails that only nonnegative integers are applicable, as countable events cannot, of course, be negative.
- These events are typically fixed within a predefined interval of time or space, such as the number of arrivals *within an hour* or the count of typographical errors *per page*.

The Poisson distribution is named after the French mathematician Simeon-Denis Poisson, who introduced it in the early 19th century. One of the earliest recorded applications of the Poisson distribution was by a statistician named Ladislaus Bortkiewicz, who used it to model the annual rate of accidental deaths in the Prussian army caused by horse kicks. Today, the Poisson distribution can be applied to various scenarios. It can model the annual occurrences of meteor showers, the diagnosis count of measles within a year, daily text message traffic, hourly customer purchases, accidents at a specific intersection during peak hours, or the hourly inbound call volume received by a customer service center.

PROPERTIES

The PMF of the Poisson distribution is given by the following equation:

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

where

- $P(X = k)$ is the probability of observing k events.
- e is the base of the natural logarithm, approximately equal to 2.72.
- λ (lambda) is the average rate of occurrence (also known as the rate parameter).
- k is the number of events.

It's often observed that the PMF for the Poisson distribution includes the mean (μ) rather than the rate parameter (λ). However, this discrepancy is inconsequential because in the Poisson distribution, both the mean and the variance (σ^2) are equal to the rate parameter, so

$$\mu = \sigma^2 = \lambda$$

In essence, then, the PMF for the Poisson distribution models the probability of observing a specific number of events within a fixed interval of time or space, given the average rate at which those events occur. It uses the frequency of recorded events to estimate the rate at which the same events should occur in a future interval of time or space. Yet the Poisson distribution is memoryless. That is to say, although the Poisson distribution does utilize past event frequencies to estimate future rates of occurrence, its memoryless property explicitly refers to the probability of an event happening at a *specific* time in the future. This property therefore implies that the *distribution* of future events remains the same regardless of when the previous like events occurred, given the overall average rate of occurrence.

Previously, we noted that the Poisson distribution is discrete and nonnegative. As the rate parameter increases, the distribution becomes more symmetric and eventually resembles a normal distribution. Let's look at this phenomenon (see figure 3.7).

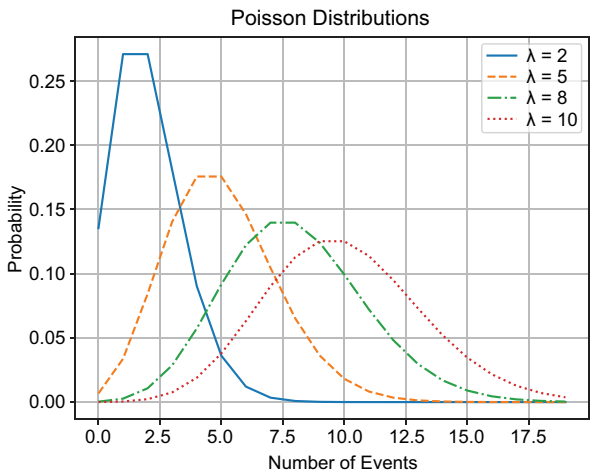

Figure 3.7 Four Poisson distributions, distinguished by their rate parameters, plotted in a single graph. As the rate parameter increases, the distribution mutates from right-skewed to normal.

We've embedded the PMF for the Poisson distribution in a chunk of Matplotlib code to create a single graph that plots four Poisson distributions distinguished only by their respective rate parameters. When the rate parameter is relatively low—that is, the rate at which an event occurs is relatively rare—the distribution takes on a right-skewed shape, where the tail on the right-hand side of the distribution is noticeably longer than the tail on the left-hand side. This is to say that, when the frequency of past occurrences is low, the PMF returns a distribution that estimates a high probability for a small number of events and low or zero probabilities for a larger number of events.

As the rate parameter (λ) increases, the Poisson distribution gradually becomes more symmetrical, resembling a normal distribution when λ reaches a sufficiently high value. This symmetry implies that the PMF then estimates nearly equal probabilities for the number of events to be either less than or greater than the given rate parameter (or the mean). Notably, the Poisson distribution can be seen as a limiting case of the binomial distribution when the number of trials is large and the probability of each event is small. This connection underscores why the Poisson distribution approximates a normal shape at high values of λ, much like how a binomial distribution also becomes approximately normal under similar conditions.

COMPUTING PROBABILITIES

Let's begin by setting two parameters: the rate parameter, denoted as lambda (`lam`), is set to `2`, and the number of events, denoted as `k`, is set to `4`. Essentially, this means the estimated number of events within a fixed period of time, based on historical rates of occurrence, is two, and we're interested in calculating the probability of observing four events:

```
>>> lam = 2
>>> k = 4
```

In Python, there are two methods for obtaining the probability. The first is to pass the values for `lam` and `k` to an arithmetic operation derived from the PMF for Poisson distributions, like so:

```
>>> probability_formula = ((np.exp(-lam) * lam ** k) / \
>>>                        math.factorial(k))
```

Then we print the probability—converted from a decimal to a percentage by multiplying the result by `100`—as a formatted string:

```
>>> print(f'Probability (using PMF formula): '
>>>       f'{probability_formula * 100}%')
Probability (using PMF formula): 9.02235221577418%
```

The second method is to pass the same values to `poisson.pmf()`, which automatically calls the PMF and returns the same result. This operation first requires the `poisson` module from `scipy.stats`:

```
>>> from scipy.stats import poisson
>>> probability_python = poisson.pmf(k, lam)
```

```
>>> print(f'Probability (using Python method): '
>>>       f'{probability_formula * 100}%')
Probability (using Python method): 9.02235221577418%
```

So, when the estimated number of events is two, the probability of actually observing four events is about 9%. We can revisit figure 3.7, particularly focusing on the Poisson distribution with a rate parameter of 2, for comparison. Our computed probability aligns closely with the graph.

Before moving on to probability computations, let's repeat these exercises, but with different values for lambda (`lam`) and `k`:

```
>>> lam = 10
>>> k = 8

>>> probability_formula = ((np.exp(-lam) * lam ** k) / \
>>>                        math.factorial(k))
>>> print(f'Probability (using PMF formula): '
>>>       f'{probability_formula * 100}%')
Probability (using PMF formula): 11.259903214901998%

>>> probability_python = poisson.pmf(k, lam)
>>> print(f'Probability (using Python method): '
>>>       f'{probability_formula * 100}%')
Probability (using Python method): 11.259903214901998%
```

Regardless of which method is used, we get a roughly 11% probability of observing 8 events when the estimated number of events is 10. These results, too, align well with the Poisson distribution from figure 3.7, where the parameter rate is 10.

KEY TAKEAWAYS

The Poisson distribution is unique in that it seamlessly integrates empirical data with theoretical probabilities, offering a practical tool for modeling real-world events that occur independently over a fixed interval. Unlike other probability distributions that are more theoretical, the Poisson distribution directly connects with observed data, making it invaluable for applications like traffic flow analysis, call center management, and biological processes. One of its defining features is the way its shape can dramatically change with variations in the rate parameter, shifting from a highly skewed form for low rates to a more symmetric shape as the rate increases. Notably, the Poisson distribution can be seen as an approximation of the binomial distribution when the number of trials is large and the probability of each event is small. This adaptability makes the Poisson distribution a versatile and essential concept in both theoretical and applied statistics.

3.2 Probability problems

Having explored several probability and counting rules in the prior chapter—mostly from a theoretical perspective—we now intend to apply them empirically, demonstrating along the way that solving real-world problems typically requires some combination

of these same rules. In chapter 2, we discovered that the probability of a (successful) event can be determined by dividing the number of favorable outcomes by the total number of possible outcomes. For instance, when rolling a six-sided die and aiming for an even number, out of the six possible outcomes, three would qualify as successes. Thus, the probability of success is calculated as three out of six, or 50%.

Then we demonstrated how to convert probabilities to odds—by dividing the number of potential successes by the number of potential failures or, alternatively, by dividing the probability of success by the probability of failure. So, when again rolling a six-sided die, the probability of getting a 4 is one out of six, or about 16.7%. This means the odds of getting a 4 are derived by dividing 1 by 5, or rather 0.167 by its inverse, 0.833. Either way, we get 0.2, or 20%, as a result.

Furthermore, we explored the multiplication and addition rules and discussed where and when each of these rules apply. According to the multiplication rule, the probability of two or more events occurring is equal to the product of their individual probabilities. So, if we roll a pair of six-sided dice, the probability of getting a pair of 4s is equal to the probability of rolling one 4 multiplied by the probability of rolling another 4, or 16.7% times 16.7%: less than 3%.

Although the multiplication rule applies to two or more independent events occurring simultaneously, the addition rule, by contrast, is used to calculate the probability of at least one of two mutually exclusive events happening. Consider once more the event of rolling a single six-sided die. The addition rule can be applied to find the probability of rolling either a 4 or a 5. The probability of rolling a 4 is 16.7%, and the probability of rolling a 5 is also 16.7%. Using the addition rule, we sum the probabilities of these two mutually exclusive events—16.7% plus 16.7%—to get the probability of rolling a 4 or a 5 on a single die: 33%.

We concluded our exploration of basic probabilities with an in-depth look into the intricacies of combinations and permutations. We isolated the key differentiator between the two: permutations account for arrangements where the order matters, and combinations do not. Furthermore, we provided step-by-step instructions for computing combinations and permutations, broken down by scenarios involving replacement and without replacement.

Our purpose here is not to belabor these concepts but rather to present them in compact formats for the twin purposes of easy reference and deeper appreciation and understanding. To achieve these goals, we'll construct a single table—a quick reference guide—that neatly organizes these concepts into their own rows, featuring concise definitions, formulas, and, where applicable, short snippets of Python code. Then we'll present a series of probability problems and demonstrate that solving them typically requires the combined use of multiple probability concepts and counting rules. The reference guide will be a valuable resource, eliminating the need to revisit chapter 2 and navigate through its content. But first, a discussion of the complement rule for probability is in order.

3.2.1 Complement rule for probability

Sometimes it's much easier to find the probability of an event *not* happening than it is to find the probability that the same event *will* happen. The complement rule is a fundamental principle in probability theory that states that the probability of the complement of an event is equal to 1 minus the probability of the event itself. In simpler terms, if we denote the event of interest as A, then the complement of A, typically denoted as A^C, represents all outcomes that are not in A. Mathematically, this can be expressed as $P(A^C) = 1 - P(A)$. By using the complement rule, we can often quickly and easily calculate the probability of an event not occurring by subtracting the probability of an event from 1. This rule is particularly useful when dealing with scenarios where it is easier to determine the probability of the complement, or inverse, rather than the event itself, allowing for more efficient probability computations and problem-solving strategies.

Let's demonstrate the complement rule with a pair of examples. We want to find the probability of getting a 4 on at least one of two dice rolls; thus, we need to consider the following three scenarios:

1. Getting a 4 on the first die and any number other than 4 on the second die
2. Getting a number other than 4 on the first die and a 4 on the second die
3. Getting a 4 on both dice

We can calculate the probabilities of these three scenarios by repeatedly applying the multiplication rule. With respect to the first and second scenarios, the probability of getting a 4 on one die is 1/6, and the probability of getting any number other than 4 on the other die is 5/6; therefore, the probability is equal to $1/6 \times 5/6$, or 5/36, or about 14%.

The probability of getting a 4 on both dice is $1/6 \times 1/6$, or 1/36, or less than 3%. Then we can apply the addition rule once to get the total probability: $5/36 + 5/36 + 1/36 = 11/36$. This is to say that, out of 36 total possible outcomes, 11 of them, or about 31%, result in getting a 4 at least once.

That was rather cumbersome. The complement of getting at least one 4 from two dice rolls is twice getting any number other than 4. This, by again applying the multiplication rule, is equal to $5/6 \times 5/6$, or 25/36, or about 69%; thus, the probability of getting the inverse is 1 minus this result, or about 31%.

We can get the exact same result with just a few lines of Python code. In the first line of code, the variable `probability_not_4` equals `5/6` and therefore represents the probability of rolling a six-sided die and getting some number other than 4:

```
>>> probability_not_4 = 5/6
```

In the next line of code, the variable `probability_not_4_both` equals the probability of getting any number other than 4 from a pair of dice rolls. The `pow()` method is used to compute the power of a number. It takes two arguments: the base and the

exponent. In this line of code, `pow()` raises the variable `probability_not_4` to the power of `2`. In other words, it essentially applies the multiplication rule to compute the probability of not getting a 4 from either die roll and assigns the result to `probability_not_4_both`:

```
>>> probability_not_4_both = pow(probability_not_4, 2)
```

Then we instruct Python to print the result as a percentage, rather than as a decimal, in the form of a printed string:

```
>>> print(f'Probability of NOT getting at least one 4: '
>>>       f'{probability_not_4_both * 100}%')
Probability of NOT getting at least one 4: 69.44444444444446%
```

Let's look at one more example. This time, we want the probability of getting heads (H) at least one time when flipping three coins, versus all tails (T). By applying the multiplication rule, we get a total of eight possible outcomes:

1. HHH
2. HHT
3. HTH
4. HTT
5. THH
6. THT
7. TTH
8. TTT

Of these eight outcomes, seven include at least one coin that is heads. So, the probability of getting at least one heads from three coin flips is 7/8, or almost 88%.

Not bad. However, combining the multiplication rule with the complement rule to derive this same outcome is still the most efficient alternative.

The complement of getting heads at least once is not getting heads at all, or getting tails on all three coin flips. We can use the multiplication rule to calculate the probability of getting all tails. Because each coin flip is independent, the probability of getting tails on a single coin flip is 1/2. Therefore, the probability of getting tails on all three coin flips is $(1/2)^3$, or 1/8, which is equal to about 12%. Subtracting this result from 1 gets us the inverse, or the probability of getting heads at least once, which of course is 88%.

3.2.2 Quick reference guide

Now that the complement rule for probability is in our rearview mirror, it's time to unveil the quick but comprehensive reference guide in table 3.1. It is a snapshot of the probabilities and counting rules covered so far in this chapter and chapter 2; it includes short definitions, formulas, and snippets of Python code, packed into a single table. So, if you're tackling, let's say, a problem involving permutations without replacement, and you require just the formula and a snippet of code, there's no need

to sift through lengthy explanations, now or in the future; instead, table 3.1 puts that information right at your fingertips.

Table 3.1 Quick reference guide for probabilities, odds, and counting rules

Theoretical probability Theoretical ratio of successes to outcomes	**Empirical probability** Ratio of observed success to observations
Formula	**Formula**
$P(\text{Event}) = \frac{\text{number of successes}}{\text{number of outcomes}}$	$P(\text{Event}) = \frac{\text{number of successes observed}}{\text{number of observations made}}$

Complement rule Inverse probability of an event occurring or the probability of an event not occurring
Formula
$P(A^{\text{C}}) = 1 - P(A)$

Odds Ratio of successes to failures or the probability of success to the probability of failure
Formulas
$\text{Odds} = \frac{\text{number of potential successes}}{\text{number of potential failures}}$ or $\text{Odds} = \frac{\text{probability of success}}{\text{probability of failure}}$

Multiplication rule Applies to simultaneous or sequential events	**Addition rule** Applies to mutually exclusive events
Formula	**Formula**
$n_1 \times n_2 \times \ldots n_k$	$n_1 + n_2 + \ldots n_k$

Permutations with replacement: order matters; repetition allowed	
Formula	**Python code**
n^r	`pow(n, r)`

Permutations without replacement: order matters; no repetition allowed	
Formula	**Python code**
$\frac{n!}{(n-r)!}$	`math.perm(n, r)`

Table 3.1 Quick reference guide for probabilities, odds, and counting rules *(continued)*

Combinations without replacement: order irrelevant; no repetition allowed	
Formula	**Python code**
$\frac{n!}{r!\,(n-r)!}$	`math.comb(n, r)`
Combinations with replacement: order irrelevant; repetition allowed	
Formula	**Python code**
$\frac{(n+r-1)!}{r!\,(n-1)!}$	`math.comb(n + r - 1, r)`

Keep in mind that *n* typically refers to the total number of items in a set, and *r* usually represents the number of items chosen or selected from that set.

3.2.3 Applied probability: Examples and solutions

The upcoming probability problems (and their solutions) feature scenarios involving rolling dice, flipping coins, and selecting cards from a standard deck. We will present several probability problems to showcase the combined application of the many probability and counting rules introduced thus far.

A typical die has six sides or faces, numbered 1 through 6. We always assume an equal probability of getting any of these outcomes when rolling a die, regardless of any previous results. Of course, when playing craps, backgammon, or many other games, two dice are rolled together. When that is the case, we write (3, 4) to represent getting 3 on the first die and 4 on the second, which is a different outcome from (4, 3), or getting 4 on the first die and 3 on the second die.

When a fair coin is flipped, we assume equal probabilities of getting heads (H) or tails (T), independent of previous coin flips. So, the probability of heads and the probability of tails always equal 0.5. Although many scenarios involve flipping just a single coin once, there are other scenarios where multiple coins are flipped simultaneously or a single coin is flipped multiple times in succession.

Selecting cards from a standard playing deck of 52 cards can potentially result in millions of combinations or permutations. The 52 cards are equally divided into four suits: hearts, diamonds, clubs, and spades. Within each suit, the cards are numbered between 1 and 10; plus, there is a King, a Queen, and a Jack for each suit. Cards numbered 1 are typically referred to as Aces. That all being said, let's get started.

PROBLEM 1

Problem: Two dice are rolled together. What is the probability of getting 1, 2, or 3 on the first die and 4, 5 or 6 on the second die?

Solution: We first use the multiplication rule to get the denominator, or the total number of possible outcomes. There are six possible outcomes when rolling a single

six-sided die, so there must be 6 × 6, or 36, possible outcomes when rolling a pair of six-sided dice together. We get the numerator by again applying the multiplication rule. There are three successful outcomes from the first die and three successful outcomes from the second die; so, there must be 3 × 3, or nine, possible successes. Therefore, the probability of getting 1, 2, or 3 on the first die and 4, 5, or 6 on the second die is 9/36, or 25%, when we convert the result from a fraction to a percentage.

PROBLEM 2

Problem: Two dice are rolled together. What are the odds of getting 1, 2, or 3 on the first die and 4, 5, or 6 on the second die?

Solution: We know that the probability of success is equal to 0.25; therefore, the probability of failure must be the inverse of that, or 0.75. So, the odds of getting 1, 2, or 3 on one die and 4, 5, or 6 on the other are 0.25/0.75, or 33%.

PROBLEM 3

Problem: Two dice are rolled together. What is the probability of getting a sum of 7 or a sum of 11?

Solution: Six possible outcomes sum to 7: (1, 6), (2, 5), (3, 4), (4, 3), (5, 2), and (6, 1). There are, of course, 36 total outcomes when rolling a pair of six-sided dice, so the probability of rolling a sum of 7 is equal to 6/36, or 16.7%. Only two possible outcomes sum to 11: (5, 6) and (6, 5). So, the probability that two dice sum to 11 is 2/36, or between 5% and 6%. Because these are mutually exclusive events, we then apply the addition rule to get the probability that two dice will sum to 7 or 11: 6/36 + 2/36 = 8/36, or 22.2%.

PROBLEM 4

Problem: Two dice are rolled together. What is the probability of getting a sum equal to or greater than 2 and equal to or less than 11?

Solution: Rather than calculating the probabilities for all sums between 2 and 11 and then again applying the addition rule to get the total probability, the quickest, easiest, and surest method is to utilize the complementary rule. This of course entails computing the probability of the complementary event—rolling a sum of 12—and then subtracting that probability from 1 to obtain the probability of the desired outcome. The likelihood of rolling double 6s, the only way a pair of dice can sum to 12, is 1 in 36, or approximately 3%. Consequently, the probability of obtaining any other outcome is the complement of that, which is 1 – 1/36, or equivalently, 36/36 – 1/36, resulting in a probability of about 97% when expressed as a percentage.

PROBLEM 5

Problem: A single coin is flipped five times. What is the probability of getting heads on all five flips?

Solution: The probability of getting heads from just one flip is 1/2. We then apply the multiplication rule to get the probability of getting the same result four more times. So, the probability of getting heads on five out of five coin flips is equal to $(1/2)^5$, or 1/2 multiplied by itself five times. This equals 1/32, or approximately 3%.

PROBLEM 6

Problem: A single coin is flipped five times. What is the probability of getting the following sequence: HTHHT?

Solution: The probability of getting heads on the first flip is 1/2. The probability of getting tails on the second flips is also 1/2, and so forth. So, the probability of getting this sequence, or any specific sequence from five flips, is equal to $(1/2)^5$, or roughly 3%.

PROBLEM 7

Problem: A single coin is about to be flipped five times. How many different sequences are there?

Solution: If any random sequence has a roughly 3% probability of occurring, it stands to reason that there must be 32 or 33 possible sequences. There are two possible outcomes from any coin flip. Because each flip is independent of every other flip, we can use the multiplication rule to get the exact number of possible sequences. So, two possible outcomes from five flips means there are 2^5 or 32 possible sequences of heads and tails.

PROBLEM 8

Problem: A single coin is flipped five times. What is the probability of getting exactly two heads and three tails, in any order?

Solution: We just learned there are 32 possible sequences of heads and tails, so that's our denominator. To find the numerator, we utilize the combinations-without-replacement counting rule. That's because the order of flips doesn't matter, but once we've achieved two heads, further replacement isn't allowed. We can therefore pass 5 and 2 to the `math.comb()` method; `math.comb()` assigns these two parameters as n and r, respectively, runs the combinations-without-replacement formula, and returns 10 as the numerator. Thus, there is a 10 in 32 chance of getting exactly two heads and three tails from five coin flips, or an approximate probability of 31%.

PROBLEM 9

Problem: A single card is to be selected from a standard deck. What is the probability of selecting a face card?

Solution: There are four suits and three face cards per suit, so 12 of the 52 cards are face cards. Thus, there is a 12/52 chance, or 23% probability, of selecting a face card.

PROBLEM 10

Problem: Two cards are to be selected from a standard deck. What is the probability of selecting two face cards if the first card is not returned to the deck?

Solution: We know the probability of selecting one face card. The probability of then selecting a second face card is not 12/52, but rather 11/51, because the first card was permanently removed from the deck. To find the probability of two consecutive successes, we apply the multiplication rule, where $12/52 \times 11/51 = 132/2{,}652$, which is roughly equal to 5%. So, there is about a 5% probability of consecutively selecting two face cards from a standard deck without replacement.

PROBLEM 11

Problem: Five cards are selected (or dealt) at random for a game of poker. How many different poker hands are possible?

Solution: This is another example of a combination-without-replacement problem. We therefore pass 52 and 5 to the `math.comb()` method, which returns a result of 2,598,960 possible hands.

PROBLEM 12

Problem: Now we're playing bridge; 13 cards are selected (or dealt) at random. How many different bridge hands are possible?

Solution: This time, we pass 52 and 13 to the `math.comb()` method, which returns a result of 635,013,559,600 possible hands.

PROBLEM 13

Problem: Five cards are to be selected from a standard deck. What is the number of ways to arrange the cards in a specific order, without replacement?

Solution: Because the order is now significant, this becomes a permutations-without-replacement problem. We therefore pass 52 as the total number of items (n) and 5 as the number of items to be selected (r) to Python's `math.perm()` method; `math.perm()` computes the number of ways to select and arrange 5 items from a set of 52 items without the possibility of replacement, which equals 311,875,200 possible permutations.

PROBLEM 14

Problem: Five cards are to be selected from a standard deck. What is the number of possible combinations if replacement is allowed?

Solution: In this instance, whenever a card is selected, it is then returned to the deck to potentially be selected again. This is therefore a combination-with-replacement problem, because the order doesn't matter. So, we pass 56 and 5 to the `math.comb()` method, which returns the combinations count by computing the number of ways to choose 5 items from 56 options. The result is 3,819,816 possible combinations.

The preceding series of probability problems aimed to offer a comprehensive set of examples with varying levels of complexity. The goal was to ensure a solid understanding of how to apply different probability concepts and counting rules in practical contexts, highlighting the need to combine these methods to solve complex probability problems.

As we conclude our exploration of probability computations, it's worth acknowledging that probabilities often extend beyond simple or straightforward events to more complex scenarios, where outcomes may depend on prior conditions or events, which leads us to conditional probabilities. Conditional probabilities allow us to calculate the likelihood of an event occurring given that another event has already occurred. This concept is critical in many real-world applications, from weather forecasting to medical diagnosis. By exploring conditional probabilities, we gain a deeper understanding of probabilistic reasoning and its practical implications in decision-making and problem-solving.

3.3 Conditional probabilities

Conditional probabilities refer to the likelihood of an event occurring given that another event has already happened or is known, or at least assumed, to be true. Rather than quantifying probabilities in isolation, we assess them within a context where additional information shapes our expectations. From a mathematical perspective, known or presumed information typically triggers a change in the denominator, representing the number of possibilities, without necessarily any modification to the numerator, representing the number of successes.

From a more practical perspective, it affects the size of the sample space: that is, the set of all possible outcomes or events that could occur in a given experiment or situation. Understanding conditional probabilities enables us to navigate uncertainty more effectively, as it allows for nuanced decision-making based on available and relevant information. By recognizing how the probability of an event changes with additional data points, we gain a more comprehensive understanding of the changing dynamics across various scenarios, from risk assessment to strategic planning. A few examples should help.

3.3.1 Examples

The following examples, deliberately chosen for their relative simplicity, are provided to reinforce our fundamental understanding of conditional probabilities before we explore their mathematical intricacies and other properties.

WEATHER FORECASTING

- *Without additional information:* The probability of rain tomorrow, based on historical record-keeping, might be 30%.
- *With additional information* (dark clouds gathering, rainy conditions in neighboring states): The probability of rain might increase to 70%.

The denominator, or sample space, is reduced from all possible weather conditions to cloudy and rainy conditions. By reducing the denominator but keeping the numerator constant, the probability of rain increases.

MEDICAL TESTING

- *Without additional information:* The probability of having a noninfectious disease might be 5%, based on general population statistics.
- *With additional information* (positive test result): The probability of having a noninfectious disease might increase to 80%.

In this case, the sample space changes from the entire population to only those individuals who test positive, significantly altering the probability.

TRAFFIC

- *Without additional information:* The probability of heavy traffic during rush hour might be 60%, based on prior commutes.

- *With additional information* (accident reported ahead): The probability of heavy traffic might increase to as high as 90%.

The denominator changes from all possible traffic conditions to only those affected by the reported accident.

BASKETBALL GAME

- *Without additional information:* The probability of a professional basketball team advancing in the playoffs might be 50%.
- *With additional information* (leading scorer/rebounder is injured): The probability of the team winning their current series and advancing to the next round of the playoffs might drop to 25%.

In this scenario, the sample space changes from all possible outcomes to only those where the team's leading scorer/rebounder is injured and therefore unable to play.

FINANCIAL INVESTMENT

- *Without additional information:* The probability of a stock price increasing in value might be 70%, based on recent market trends.
- *With additional information* (company reports strong earnings): The probability of the stock increasing might rise to 90%.

Here, the denominator is reduced from all possible market movements to only those influenced by the company's earnings report, leading to a higher probability of an increase (and likely more investments).

In each of these scenarios, the likelihood of one event has a legitimate and viable influence on the probability of another event; in other words, there is some sort of dependency between the two. But let's briefly discuss the inverse of this: when two events are instead independent of each other.

3.3.2 Conditional probabilities and independence

Imagine rolling a six-sided die and getting a 6 on one roll. Your intuition might suggest that the chance of rolling another 6 is less than the probability of rolling any other number. However, this outcome doesn't influence the probability of getting a 6, or any other outcome for that matter, on the next roll of the same die. These events are independent, meaning past results have no bearing on future ones. The roll of a single die always follows a discrete uniform distribution, regardless of previous outcomes.

Or consider five successive flips of the same fair coin, all of which resulted in heads. In no way should this prior information be used to predict the result of the next coin flip. The previous results do not change the probability of getting heads or tails on the next flip. That's because coin flips are independent events, where prior outcomes have no bearing on future outcomes. Although we should expect coin flips, or any other binomial experiment, to normalize after several flips, that's separate from the probability of getting heads (or tails) on the very next flip.

Likewise, if the queen of diamonds is selected from a standard deck of 52 cards and then immediately returned to the deck, in no way does that affect the probability of then selecting the same card when the next selection is made. The probability of again selecting the queen of diamonds is exactly the same, 1 in 52, as selecting any other card.

So, it's important to distinguish independent events from dependent events: conditional probabilities apply to the latter but not to the former. Even events that appear dependent require assessment to determine the extent of their interdependence. Rain in Oregon may or may not be a leading indicator of rain in California. Similarly, market trends in one sector of the economy may or may not influence future performance in other sectors. Now that we've clarified the contexts in which conditional probabilities are relevant and where they aren't, let's explore two methods for solving the same conditional probability problem.

3.3.3 Intuitive approach to conditional probability

Once more, conditional probability is the likelihood of an event occurring given that another event has already happened or is known to be true. It is given by the following equation:

$$P(A|B) = \frac{P(A \text{ and } B)}{P(B)}$$

where

- $P(A|B)$ denotes the conditional probability of event A occurring with the knowledge that event B has already occurred.
- $P(A \text{ and } B)$ is the probability of both events A and B occurring.
- $P(B)$ is the probability of event B occurring.

However, let's set aside the conditional probability formula *for the moment*; we'll first demonstrate how to solve a conditional probability problem not by plugging numbers into a formula but rather through a more intuitive method.

We'll start by constructing a contingency table containing the data for our analysis. A *contingency table*—sometimes referred to as a *cross-tabulation* or a *two-way table*—is a tabular representation of two categorical variables. It organizes the data into rows and columns, where each cell in the table represents the frequency or count of occurrences for a particular combination of classes from the two variables. More specifically,

- Each row in the contingency table corresponds to a distinct class of one categorical variable.
- Each column corresponds to a distinct class of the other categorical variable.
- Each cell contains a count or frequency of occurrences that align to a combination of classes from each categorical variable.

The following snippet of Python code creates a categorical variable called `test_results` containing illustrative data that initializes our contingency table.

test_results contains two classes, Negative and Positive, which represent the two possible outcomes of a medical test:

```
>>> test_results = {'Negative': [860, 10],
>>>         'Positive': [90, 40]}
```

The values assigned to each class represent counts or frequencies. The Negative class contains 860 instances where the test result was negative and 10 instances where the test result was positive. Similarly, the Positive class contains 90 instances where the test result was negative and 40 instances where the test result was positive. Negative and Positive are intended to be our column labels.

Our next line of code initializes a new categorical variable called symptoms, which consists of two classes. In this context, No signifies the absence of symptoms, and Yes signifies their presence:

```
>>> symptoms = ['No', 'Yes']
```

No and Yes will serve as our row labels.

The last piece of code constructs the contingency table and displays it in the Python console. This requires the pd.DataFrame() method from the pandas library. The pandas library contains several methods for creating and manipulating data structures; pd.DataFrame() creates a two-dimensional, labeled data structure similar to a table in a database or an Excel spreadsheet. We pass the categorical variables test_results (as the data) and symptoms (as the index) to pd.DataFrame() to create an object called contingency_table:

```
>>> import pandas as pd
>>> contingency_table = pd.DataFrame(test_results, index = symptoms)
>>> print(contingency_table)
     Negative  Positive
No        860        90
Yes        10        40
```

Now that we have our contingency table, we can write other snippets of code to extract values from it and perform arithmetic operations, such as conditional probability calculations. To demonstrate, the following line of code uses the .iloc attribute to extract the value from contingency_table where the second row and second column intersect (Python indices begin at 0 rather than 1):

```
>>> contingency_table.iloc[1, 1]
40
```

We then want to find the probability of getting a positive test result, knowing in advance the presence of symptoms, which can be expressed this way:

$$P(\text{positive test result} \mid \text{presence of symptoms})$$

This is derived by dividing the number of instances where the test result was positive and there was the presence of symptoms (A and B) by the sum of those instances and

the number of instances where the test result was negative and there was the presence of symptoms (B):

```
>>> P = (contingency_table.iloc[1, 1] / \
>>>         (contingency_table.iloc[1, 1] + contingency_table.iloc[1, 0]))
>>> print(f'Conditional probability: {P * 100}%')
Conditional probability: 80.0%
```

This is equivalent to the quotient between 40 and 50 converted from a decimal to a percentage.

From the same contingency table, we can also find the reverse—that is, the probability that the symptoms are correct given a positive test result—simply by switching A and B. It can be expressed this way:

$$P(\text{presence of symptoms} \mid \text{positive test result})$$

The numerator remains unchanged. To get the denominator, we extract the two values from the second column and add them together:

```
>>> P = (contingency_table.iloc[1, 1] / \
>>>         (contingency_table.iloc[0, 1] + contingency_table.iloc[1, 1]))
>>> print(f'Conditional probability: {P * 100}%')
Conditional probability: 30.76923076923077%
```

In both instances, we computed higher, and yet more accurate, probabilities by reducing the sample space. This triggers very different action plans than if there had been no additional information in advance.

3.3.4 Formulaic approach to conditional probability

We get the exact same results by instead plugging numbers into the conditional probability formula. Here's how.

STEP 1: DEFINE THE EVENTS

- A represents a positive test result.
- B represents the presence of symptoms.

STEP 2: FIND $P(A$ AND $B)$

- $P(A) = 130$ (total number of positive test results).
- $P(B) = 50$ (total number of instances with the presence of symptoms).
- $P(A \text{ and } B) = 40$ (total number of instances where both A and B occur.

Thus,

$$P(A \text{ and } B) = \frac{40}{1{,}000} = 0.04$$

STEP 3: FIND $P(B)$

- There are 50 instances with the presence of symptoms and 1,000 observations, so

$$P(B) = \frac{50}{1{,}000} = 0.05$$

STEP 4: CALCULATE *P*(*A* | *B*)

- $P(A \text{ and } B) = 0.04$.
- $P(B) = 0.05$.

Thus

$$P(A|B) = \frac{P(A \text{ and } B)}{P(B)} = \frac{0.04}{0.05} = 0.8$$

Just like the intuitive approach, the formulaic approach returns an 80% probability of a positive test result given the presence of symptoms.

To calculate the probability of the symptoms being correct given a positive test result, we switch A and B and divide the same numerator—after all, the probability of A and B is the same as the probability of B and A—by $P(A)$ rather than by $P(B)$:

$$P(B|A) = \frac{P(B \text{ and } A)}{P(A)} = \frac{0.04}{0.13} = 0.31$$

This again returns a 31% probability (rounded) of the symptoms being correct given a positive test result.

The intuitive approach is likely easier to understand and to carry forward than the formulaic approach. But as demonstrated here, both methods return the same results, so you should apply whichever works best for you.

We've packed a wealth of knowledge on probabilities and counting rules into just two chapters. These learnings will prove invaluable as we move forward into more specific subject matters, starting with linear regression models.

Summary

- The normal distribution is a probability distribution symmetric around the mean so that, when plotted, it assumes a bell-like shape. It is defined by two parameters: the mean, which, of course, determines the center of the distribution, and the standard deviation, which determines the spread.
- A standard normal distribution is a special case of the normal distribution with a mean of 0 and a standard deviation of 1. The raw data has been transformed, or standardized, so that the values represent the number of standard deviations they are below or above the mean.
- The binomial distribution describes the probability of a specific number of successes in a fixed number of independent trials, where each trial has but two outcomes and the probabilities of those outcomes remain constant throughout. It is characterized by two parameters: the number of trials and the probability of success. Regardless of those parameters, binomial outcomes are normally distributed.
- The discrete uniform distribution is a probability distribution where all outcomes in a finite set of values have an equal probability of occurring. It is characterized by two parameters: the number of possible outcomes and the range of values in the set.

- The Poisson distribution is a probability distribution that describes the number of events occurring in a fixed interval of time or space, given a known average rate of occurrence and assuming independence between events. It is characterized by a single parameter: the rate of occurrence.
- When computing probabilities, it's essential to use a combination of various probabilities and counting rules to accurately assess the likelihood of events occurring.
- In some scenarios, it can be simpler to compute the inverse, or complement, of an event's occurrence, thereby providing an alternative approach to understanding its probability.
- Conditional probabilities represent the likelihood of an event occurring given that another event has already occurred, thereby allowing for a deeper understanding of how events relate to each other. They are calculated by adjusting the probability of the event of interest based on additional information provided by the occurrence of another event. In other words, the sample space is reduced by decreasing the denominator and maybe the numerator as well, which can significantly influence the dynamics of subsequent decision-making and risk assessment.