18.8 Distributions

Now that we have learned how to work with probability in both the discrete and the continuous setting, let us get to know some of the common distributions encountered. Depending on the area of machine learning, we may need to be familiar with vastly more of these, or for some areas of deep learning potentially none at all. This is, however, a good basic list to be familiar with. Let us first import some common libraries.

```
%matplotlib inline
from d2l import mxnet as d2l
from IPython import display
from math import erf, factorial
import numpy as np
```

18.8.1 Bernoulli

This is the simplest random variable usually encountered. This random variable encodes a coin flip which comes up 1 with probability p and 0 with probability $1-p$. If we have a random variable X with this distribution, we will write

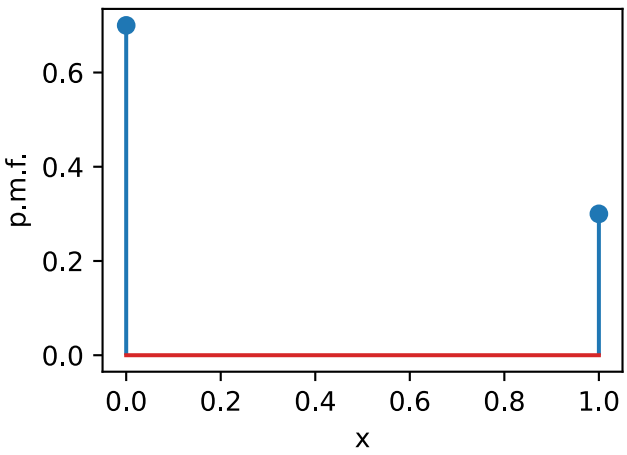
$$X \sim \text{Bernoulli}(p). \tag{18.8.1}$$

The cumulative distribution function is

$$F(x) = \begin{cases} 0 & x < 0, \\ 1-p & 0 \leq x < 1, \\ 1 & x >= 1. \end{cases} \tag{18.8.2}$$

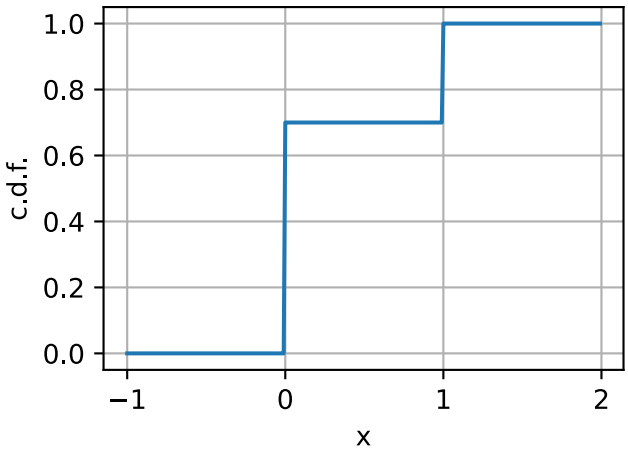
The probability mass function is plotted below.

```
p = 0.3

d2l.set_figsize()
d2l.plt.stem([0, 1], [1 - p, p], use_line_collection=True)
d2l.plt.xlabel('x')
d2l.plt.ylabel('p.m.f.')
d2l.plt.show()
```



Now, let us plot the cumulative distribution function (18.8.2).

```
x = np.arange(-1, 2, 0.01)

def F(x):
    return 0 if x < 0 else 1 if x > 1 else 1 - p

d2l.plot(x, np.array([F(y) for y in x]), 'x', 'c.d.f.')
```



If $X \sim \text{Bernoulli}(p)$, then:

- $\mu_X = p$,
- $\sigma_X^2 = p(1-p)$.

We can sample an array of arbitrary shape from a Bernoulli random variable as follows.

```
1*(np.random.rand(10, 10) < p)
```

```
array([[1, 0, 0, 1, 0, 0, 0, 0, 1, 0],
       [1, 0, 1, 0, 1, 0, 0, 0, 1, 0],
       [0, 1, 1, 0, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 1, 1, 1],
       [0, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 1, 0, 1, 1, 0, 0, 0, 0, 1],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 1, 0, 1, 1, 0, 0]])
```

18.8.2 Discrete Uniform

The next commonly encountered random variable is a discrete uniform. For our discussion here, we will assume that it is supported on the integers $\{1, 2, \ldots, n\}$, however any other set of values can be freely chosen. The meaning of the word *uniform* in this context is that every possible value is equally likely. The probability for each value $i \in \{1, 2, 3, \ldots, n\}$ is $p_i = \frac{1}{n}$. We will denote a random variable X with this distribution as

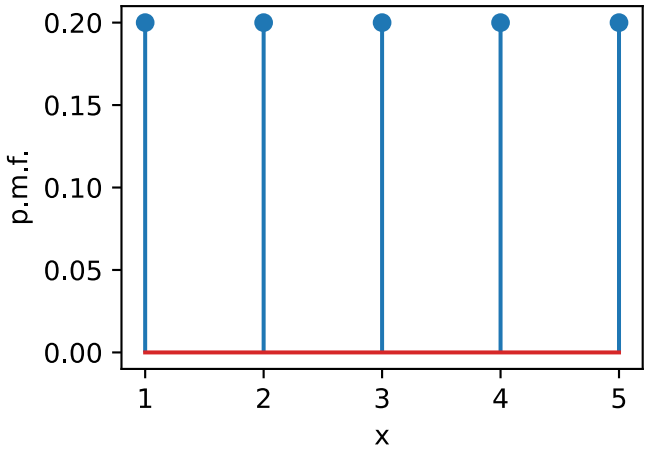
$$X \sim U(n). \tag{18.8.3}$$

The cumulative distribution function is

$$F(x) = \begin{cases} 0 & x < 1, \\ \frac{k}{n} & k \leq x < k+1 \text{ with } 1 \leq k < n, \\ 1 & x >= n. \end{cases} \tag{18.8.4}$$

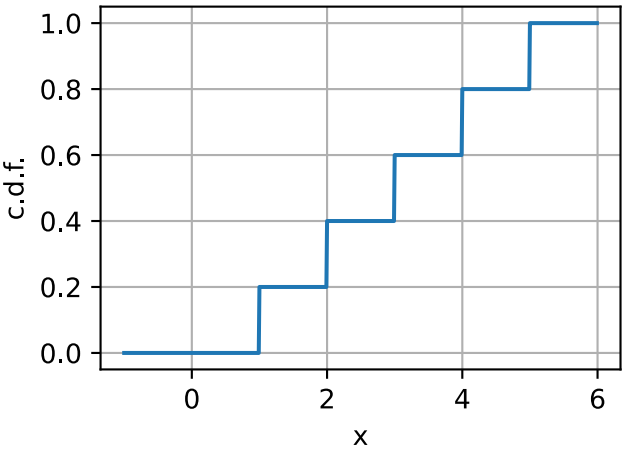
Let us first plot the probability mass function.

```
n = 5

d2l.plt.stem([i+1 for i in range(n)], n*[1 / n], use_line_collection=True)
d2l.plt.xlabel('x')
d2l.plt.ylabel('p.m.f.')
d2l.plt.show()
```



Now, let us plot the cumulative distribution function (18.8.4).

```
x = np.arange(-1, 6, 0.01)

def F(x):
    return 0 if x < 1 else 1 if x > n else np.floor(x) / n

d2l.plot(x, np.array([F(y) for y in x]), 'x', 'c.d.f.')
```



If $X \sim U(n)$, then:

- $\mu_X = \frac{1+n}{2}$,
- $\sigma_X^2 = \frac{n^2-1}{12}$.

We can sample an array of arbitrary shape from a discrete uniform random variable as follows.

```
np.random.randint(1, n, size=(10, 10))
```

```
array([[3, 2, 4, 3, 2, 3, 3, 3, 2, 2],
       [4, 3, 3, 2, 4, 2, 2, 1, 2, 4],
       [1, 3, 4, 1, 4, 4, 1, 3, 2, 1],
       [2, 1, 2, 1, 4, 1, 2, 4, 1, 3],
       [2, 3, 1, 4, 1, 4, 2, 4, 4, 4],
       [4, 3, 3, 4, 2, 2, 1, 1, 3, 1],
       [3, 2, 3, 1, 3, 3, 3, 1, 3, 4],
       [1, 3, 4, 1, 4, 1, 4, 2, 2, 3],
       [4, 2, 3, 1, 4, 4, 1, 2, 3, 1],
       [4, 2, 3, 3, 2, 1, 3, 4, 4, 1]])
```

18.8.3 Continuous Uniform

Next, let us discuss the continuous uniform distribution. The idea behind this random variable is that if we increase the n in the discrete uniform distribution, and then scale it to fit within the interval $[a, b]$, we will approach a continuous random variable that just picks an arbitrary value in $[a, b]$ all with equal probability. We will denote this distribution as

$$X \sim U(a, b). \tag{18.8.5}$$

The probability density function is

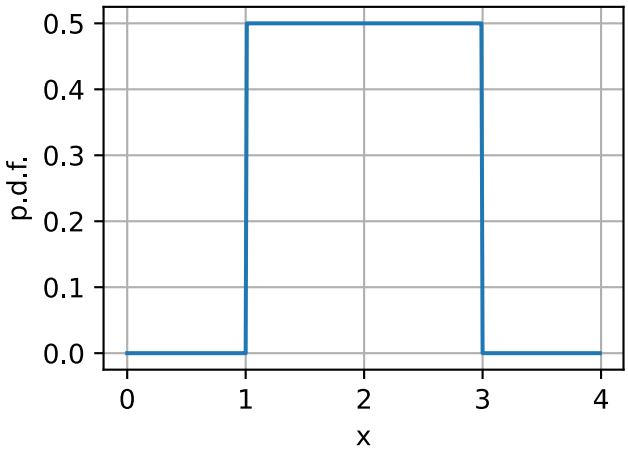
$$p(x) = \begin{cases} \frac{1}{b-a} & x \in [a, b], \\ 0 & x \notin [a, b]. \end{cases} \tag{18.8.6}$$

The cumulative distribution function is

$$F(x) = \begin{cases} 0 & x < a, \\ \frac{x-a}{b-a} & x \in [a, b], \\ 1 & x >= b. \end{cases} \tag{18.8.7}$$

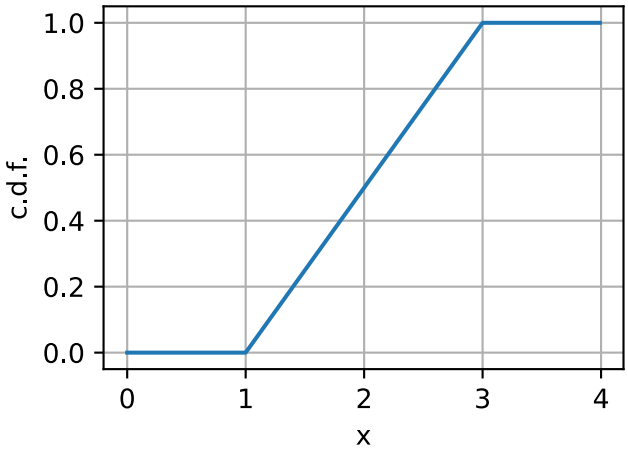
Let us first plot the probability density function (18.8.6).

```
a, b = 1, 3

x = np.arange(0, 4, 0.01)
p = (x > a)*(x < b)/(b - a)

d2l.plot(x, p, 'x', 'p.d.f.')
```



Now, let us plot the cumulative distribution function (18.8.7).

```
def F(x):
    return 0 if x < a else 1 if x > b else (x - a) / (b - a)

d2l.plot(x, np.array([F(y) for y in x]), 'x', 'c.d.f.')
```



If $X \sim U(a, b)$, then:

- $\mu_X = \frac{a+b}{2}$,
- $\sigma_X^2 = \frac{(b-a)^2}{12}$.

We can sample an array of arbitrary shape from a uniform random variable as follows. Note that it by default samples from a $U(0,1)$, so if we want a different range we need to scale it.

```
(b - a) * np.random.rand(10, 10) + a
```

```
array([[1.16638971, 2.60829226, 1.15735583, 1.36750491, 2.04113244,
        1.72380061, 1.05654174, 2.08335801, 2.01479292, 1.52882922],
       [1.89983863, 1.46675017, 1.35574418, 2.2992871 , 2.51580963,
        2.53631594, 2.65751412, 1.18817725, 1.91833913, 1.56705755],
       [1.06557466, 1.40951186, 1.0622393 , 2.85806504, 2.3121155 ,
        2.31982627, 2.18009419, 1.74715042, 1.40808742, 1.7431962 ],
       [1.61104473, 2.43062662, 2.77990591, 2.03972287, 1.48719032,
        2.61264074, 1.73899875, 2.43444439, 2.27143442, 1.79603527],
       [1.91166964, 1.11673866, 2.80828133, 1.28907311, 2.83441297,
        2.65732271, 1.67734004, 1.02790143, 1.72794931, 1.95525304],
       [1.62155918, 1.87954283, 2.69818375, 1.85903782, 1.83673141,
        1.31117936, 1.15945099, 2.51898714, 2.13606725, 2.55268084],
       [1.86327176, 1.46574516, 1.11909121, 2.00312759, 2.93744247,
        1.4939852 , 2.43412936, 2.16806221, 2.61033778, 2.04561783],
       [2.70323408, 1.10783845, 2.26832736, 2.51491486, 1.1850613 ,
        2.43226205, 2.07965973, 2.55486896, 2.97074634, 1.92170188],
       [1.68736522, 2.70051563, 2.80058273, 2.41516343, 1.54603804,
        2.17228632, 2.4443905 , 2.42810564, 2.40509585, 2.09178648],
       [1.96977242, 2.14372728, 1.67140827, 2.67258417, 1.07308583,
        2.25937872, 1.96890949, 2.43525078, 1.66113014, 2.87820763]])
```

18.8.4 Binomial

Let us make things a little more complex and examine the *binomial* random variable. This random variable originates from performing a sequence of n independent experiments, each of which has probability p of succeeding, and asking how many successes we expect to see.

Let us express this mathematically. Each experiment is an independent random variable X_i where we will use 1 to encode success, and 0 to encode failure. Since each is an independent coin flip which is successful with probability p, we can say that $X_i \sim \text{Bernoulli}(p)$. Then, the binomial random variable is

$$X = \sum_{i=1}^{n} X_i. \tag{18.8.8}$$

In this case, we will write

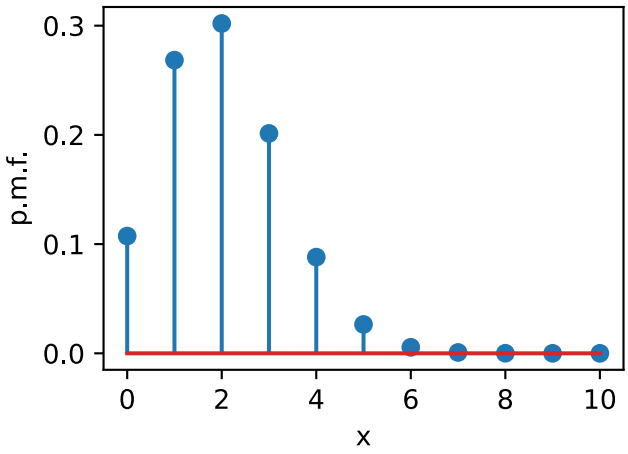
$$X \sim \text{Binomial}(n, p). \tag{18.8.9}$$

To get the cumulative distribution function, we need to notice that getting exactly k successes can occur in $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ ways each of which has a probability of $p^k(1-p)^{n-k}$ of occurring. Thus the cumulative distribution function is

$$F(x) = \begin{cases} 0 & x < 0, \\ \sum_{m \leq k} \binom{n}{m} p^m (1-p)^{n-m} & k \leq x < k+1 \text{ with } 0 \leq k < n, \\ 1 & x >= n. \end{cases} \tag{18.8.10}$$

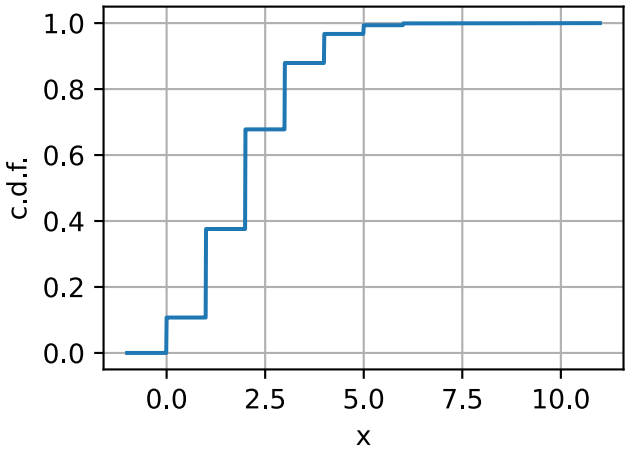
Let us first plot the probability mass function.

```
n, p = 10, 0.2

# Compute binomial coefficient
def binom(n, k):
    comb = 1
    for i in range(min(k, n - k)):
        comb = comb * (n - i) // (i + 1)
    return comb

pmf = np.array([p**i * (1-p)**(n - i) * binom(n, i) for i in range(n + 1)])

d2l.plt.stem([i for i in range(n + 1)], pmf, use_line_collection=True)
d2l.plt.xlabel('x')
d2l.plt.ylabel('p.m.f.')
d2l.plt.show()
```



Now, let us plot the cumulative distribution function (18.8.10).

```
x = np.arange(-1, 11, 0.01)
cmf = np.cumsum(pmf)

def F(x):
    return 0 if x < 0 else 1 if x > n else cmf[int(x)]

d2l.plot(x, np.array([F(y) for y in x.tolist()]), 'x', 'c.d.f.')
```



While this result is not simple, the means and variances are. If $X \sim \text{Binomial}(n, p)$, then:

- $\mu_X = np$,
- $\sigma_X^2 = np(1-p)$.

This can be sampled as follows.

```
np.random.binomial(n, p, size=(10, 10))
```

```
array([[3, 1, 1, 2, 1, 2, 2, 0, 0, 5],
       [2, 1, 2, 1, 1, 3, 2, 0, 3, 2],
       [1, 4, 2, 1, 2, 4, 3, 2, 3, 2],
       [1, 2, 1, 3, 3, 1, 2, 2, 1, 4],
       [2, 1, 4, 1, 1, 3, 4, 2, 1, 4],
       [0, 4, 3, 1, 2, 1, 1, 6, 3, 1],
       [2, 3, 2, 2, 2, 2, 4, 5, 1, 1],
       [3, 1, 3, 1, 1, 0, 2, 4, 1, 0],
       [3, 3, 4, 5, 4, 3, 2, 1, 3, 1],
       [3, 3, 3, 3, 1, 4, 3, 2, 2, 3]])
```

18.8.5 Poisson

Let us now perform a thought experiment. We are standing at a bus stop and we want to know how many buses will arrive in the next minute. Let us start by considering $X^{(1)} \sim \text{Bernoulli}(p)$ which is simply the probability that a bus arrives in the one minute window. For bus stops far from an urban center, this might be a pretty good approximation. We may never see more than one bus in a minute.

However, if we are in a busy area, it is possible or even likely that two buses will arrive. We can model this by splitting our random variable into two parts for the first 30 seconds, or the second 30 seconds. In this case we can write

$$X^{(2)} \sim X_1^{(2)} + X_2^{(2)}, \tag{18.8.11}$$

where $X^{(2)}$ is the total sum, and $X_i^{(2)} \sim \text{Bernoulli}(p/2)$. The total distribution is then $X^{(2)} \sim \text{Binomial}(2, p/2)$.

Why stop here? Let us continue to split that minute into n parts. By the same reasoning as above, we see that

$$X^{(n)} \sim \text{Binomial}(n, p/n). \tag{18.8.12}$$

Consider these random variables. By the previous section, we know that (18.8.12) has mean $\mu_{X^{(n)}} = n(p/n) = p$, and variance $\sigma^2_{X^{(n)}} = n(p/n)(1-(p/n)) = p(1-p/n)$. If we take $n \to \infty$, we can see that these numbers stabilize to $\mu_{X^{(\infty)}} = p$, and variance $\sigma^2_{X^{(\infty)}} = p$. This indicates that there *could be* some random variable we can define in this infinite subdivision limit.

This should not come as too much of a surprise, since in the real world we can just count the number of bus arrivals, however it is nice to see that our mathematical model is well defined. This discussion can be made formal as the *law of rare events*.

Following through this reasoning carefully, we can arrive at the following model. We will say that $X \sim \text{Poisson}(\lambda)$ if it is a random variable which takes the values $\{0, 1, 2, \ldots\}$ with probability

$$p_k = \frac{\lambda^k e^{-\lambda}}{k!}. \tag{18.8.13}$$

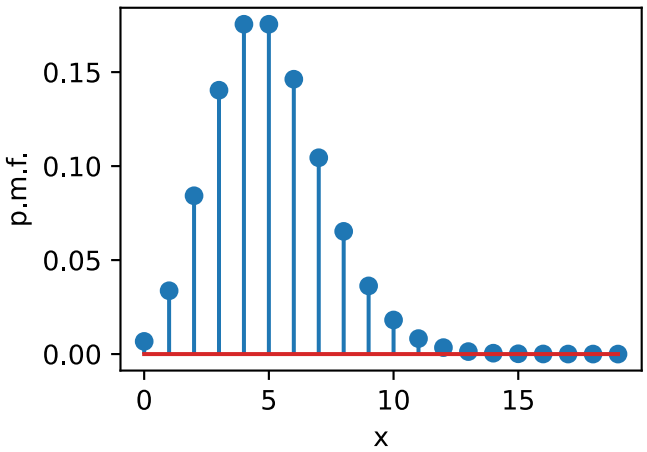
The value $\lambda > 0$ is known as the *rate* (or the *shape* parameter), and denotes the average number of arrivals we expect in one unit of time.

We may sum this probability mass function to get the cumulative distribution function.

$$F(x) = \begin{cases} 0 & x < 0, \\ e^{-\lambda} \sum_{m=0}^{k} \frac{\lambda^m}{m!} & k \leq x < k+1 \text{ with } 0 \leq k. \end{cases} \tag{18.8.14}$$

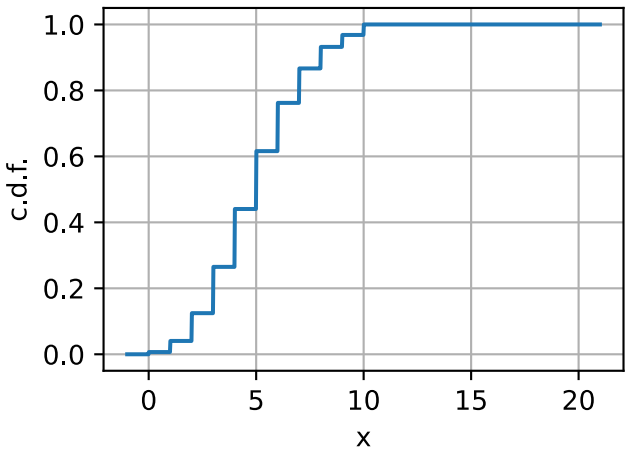
Let us first plot the probability mass function (18.8.13).

```
lam = 5.0

xs = [i for i in range(20)]
pmf = np.array([np.exp(-lam) * lam**k / factorial(k) for k in xs])

d2l.plt.stem(xs, pmf, use_line_collection=True)
d2l.plt.xlabel('x')
d2l.plt.ylabel('p.m.f.')
d2l.plt.show()
```



Now, let us plot the cumulative distribution function (18.8.14).

```
x = np.arange(-1, 21, 0.01)
cmf = np.cumsum(pmf)
def F(x):
    return 0 if x < 0 else 1 if x > n else cmf[int(x)]

d2l.plot(x, np.array([F(y) for y in x.tolist()]), 'x', 'c.d.f.')
```



As we saw above, the means and variances are particularly concise. If $X \sim \text{Poisson}(\lambda)$, then:

- $\mu_X = \lambda$,
- $\sigma_X^2 = \lambda$.

This can be sampled as follows.

```
np.random.poisson(lam, size=(10, 10))
```

```
array([[ 4,  2,  4,  5,  6,  4,  4,  4,  4,  9],
       [10,  5,  3,  6,  4,  5, 10,  6,  7,  6],
       [ 6,  4,  7,  7, 11,  3,  3,  7,  8,  9],
       [ 4,  4,  4,  5,  3,  3,  4,  2,  4,  6],
       [ 6,  3,  3,  5,  5,  5,  3,  1,  7,  3],
       [ 5,  7,  1,  6,  3,  3,  8,  3,  4,  3],
       [ 5,  7,  2,  6,  8,  2,  7,  3,  8,  6],
       [ 3,  7,  5,  6,  9,  8,  3,  2,  4,  8],
       [ 7,  3,  7,  3,  6,  5,  5,  3,  2,  1],
       [ 6,  6,  8,  4,  5,  5,  8,  8,  5,  4]])
```

18.8.6 Gaussian

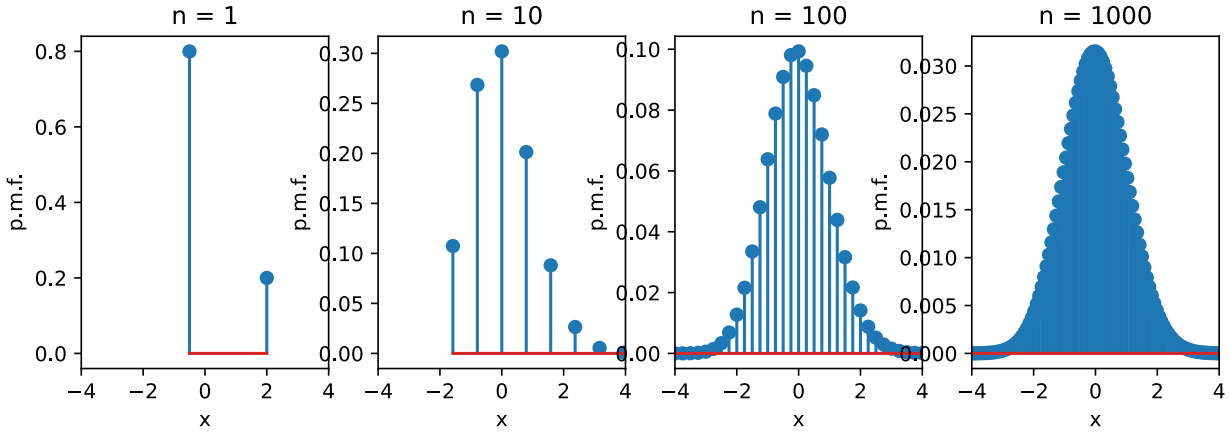
Now Let us try a different, but related experiment. Let us say we again are performing n independent Bernoulli(p) measurements X_i. The distribution of the sum of these is $X^{(n)} \sim \text{Binomial}(n, p)$. Rather than taking a limit as n increases and p decreases, Let us fix p, and then send $n \to \infty$. In this case $\mu_{X^{(n)}} = np \to \infty$ and $\sigma^2_{X^{(n)}} = np(1-p) \to \infty$, so there is no reason to think this limit should be well defined.

However, not all hope is lost! Let us just make the mean and variance be well behaved by defining

$$Y^{(n)} = \frac{X^{(n)} - \mu_{X^{(n)}}}{\sigma_{X^{(n)}}}. \tag{18.8.15}$$

This can be seen to have mean zero and variance one, and so it is plausible to believe that it will converge to some limiting distribution. If we plot what these distributions look like, we will become even more convinced that it will work.

```
p = 0.2
ns = [1, 10, 100, 1000]
d2l.plt.figure(figsize=(10, 3))
for i in range(4):
    n = ns[i]
    pmf = np.array([p**i * (1-p)**(n-i) * binom(n, i) for i in range(n + 1)])
    d2l.plt.subplot(1, 4, i + 1)
    d2l.plt.stem([(i - n*p)/np.sqrt(n*p*(1 - p)) for i in range(n + 1)], pmf,
                 use_line_collection=True)
    d2l.plt.xlim([-4, 4])
    d2l.plt.xlabel('x')
    d2l.plt.ylabel('p.m.f.')
    d2l.plt.title("n = {}".format(n))
d2l.plt.show()
```



One thing to note: compared to the Poisson case, we are now dividing by the standard deviation which means that we are squeezing the possible outcomes into smaller and smaller areas. This is an indication that our limit will no longer be discrete, but rather a continuous.

A derivation of what occurs is beyond the scope of this document, but the *central limit theorem* states that as $n \to \infty$, this will yield the Gaussian Distribution (or sometimes normal distribution). More explicitly, for any a, b:

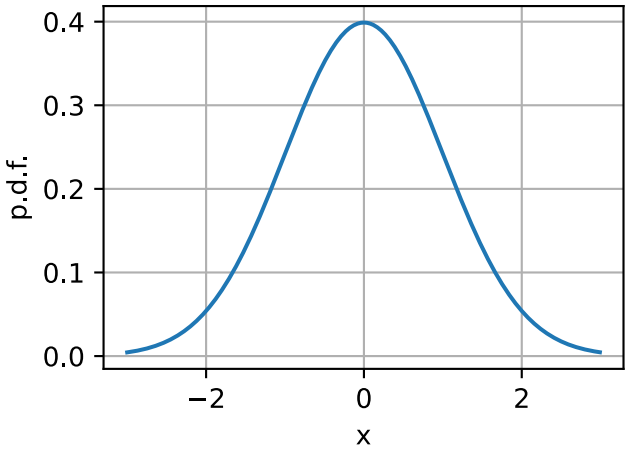
$$\lim_{n \to \infty} P(Y^{(n)} \in [a, b]) = P(\mathcal{N}(0, 1) \in [a, b]), \tag{18.8.16}$$

where we say a random variable is normally distributed with given mean μ and variance σ^2, written $X \sim \mathcal{N}(\mu, \sigma^2)$ if X has density

$$p_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{18.8.17}$$

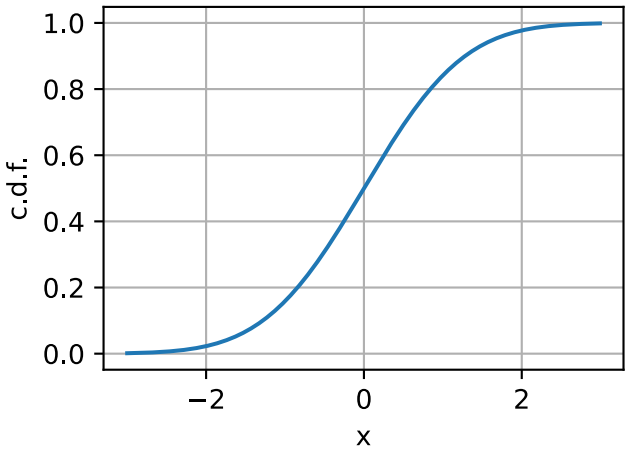
Let us first plot the probability density function (18.8.17).

```
mu, sigma = 0, 1

x = np.arange(-3, 3, 0.01)
p = 1 / np.sqrt(2 * np.pi * sigma**2) * np.exp(-(x - mu)**2 / (2 * sigma**2))

d2l.plot(x, p, 'x', 'p.d.f.')
```



Now, let us plot the cumulative distribution function. It is beyond the scope of this appendix, but the Gaussian c.d.f. does not have a closed-form formula in terms of more elementary functions. We will use `erf` which provides a way to compute this integral numerically.

```
def phi(x):
    return (1.0 + erf((x - mu) / (sigma * np.sqrt(2)))) / 2.0

d2l.plot(x, np.array([phi(y) for y in x.tolist()]), 'x', 'c.d.f.')
```



Keen-eyed readers will recognize some of these terms. Indeed, we encountered this integral in Section 18.5. Indeed we need exactly that computation to see that this $p_X(x)$ has total area one and is thus a valid density.

Our choice of working with coin flips made computations shorter, but nothing about that choice was fundamental. Indeed, if we take any collection of independent identically distributed random variables X_i, and form

$$X^{(N)} = \sum_{i=1}^{N} X_i. \tag{18.8.18}$$

Then

$$\frac{X^{(N)} - \mu_{X^{(N)}}}{\sigma_{X^{(N)}}} \tag{18.8.19}$$

will be approximately Gaussian. There are additional requirements needed to make it work, most commonly $E[X^4] < \infty$, but the philosophy is clear.

The central limit theorem is the reason that the Gaussian is fundamental to probability, statistics, and machine learning. Whenever we can say that something we measured is a sum of many small independent contributions, we can assume that the thing being measured will be close to Gaussian.

There are many more fascinating properties of Gaussians, and we would like to discuss one more here. The Gaussian is what is known as a *maximum entropy distribution*. We will get into entropy more deeply in Section 18.11, however all we need to know at this point is that it is a measure of randomness. In a rigorous mathematical sense, we can think of the Gaussian as the *most* random choice of random variable with fixed mean and variance. Thus, if we know that our random variable has some mean and variance, the Gaussian is in a sense the most conservative choice of distribution we can make.

To close the section, Let us recall that if $X \sim \mathcal{N}(\mu, \sigma^2)$, then:

- $\mu_X = \mu$,
- $\sigma_X^2 = \sigma^2$.

We can sample from the Gaussian (or standard normal) distribution as shown below.

```
np.random.normal(mu, sigma, size=(10, 10))
```

```
array([[-1.50540385, -0.15668106, -0.97883369, -0.38838738, -0.04440771,
        -0.70100778, -0.65743761,  1.29423745,  0.38279992,  3.25431916],
       [-0.06796545, -0.23545584, -0.71052995, -0.26348241, -1.01795227,
         1.38722161, -0.83838333, -0.2293768 ,  0.01806315, -0.43954804],
       [ 0.57646145, -0.96840955, -0.90086057,  0.55618025, -0.38907222,
         1.94156447, -1.38845265,  0.82532408, -0.04857338,  0.7083775 ],
       [-1.26561284, -0.8167302 ,  0.96821643,  0.53847379,  1.12030463,
         1.75629459, -0.41271404,  0.52296585,  0.61560488,  0.6504033 ],
       [-0.69995263,  0.02918155, -0.24331134,  0.93332187,  0.26428711,
         0.15676619,  0.18465117,  1.08895264,  0.77780953,  0.19227336],
       [-0.73358456,  0.11239037,  1.80201349, -0.14732248,  0.7262625 ,
        -0.90678858,  0.15750501, -0.85597781, -0.26388355, -0.29809398],
       [ 0.1132453 , -0.25272724, -0.46792835, -1.40147327,  0.09938934,
```

(continues on next page)

(continued from previous page)

```
        1.0642668 , -1.68660006,  0.3481269 , -0.83257487, -0.92267175],
       [ 0.28282213, -0.99287916, -2.33438988,  0.72532162, -0.23973005,
        -1.15449566,  0.05941123, -0.85476641, -1.57083736, -1.32798648],
       [ 1.20486519, -0.22900869,  0.26370763,  0.43254133, -0.135596  ,
         0.42526098,  0.16543189,  1.14033453,  0.62693223,  0.6667187 ],
       [ 0.25819661,  1.31339443, -0.9203143 , -0.1173653 ,  0.08648907,
        -2.31887364,  0.28883866,  0.98434658, -0.13177449, -0.32659498]])
```

18.8.7 Exponential Family

One shared property for all the distributions listed above is that they all belong to which is known as the *exponential family*. The exponential family is a set of distributions whose density can be expressed in the following form:

$$p(\mathbf{x}|\eta) = h(\mathbf{x}) \cdot \exp\big(\eta^\top \cdot T(x) - A(\eta)\big) \tag{18.8.20}$$

As this definition can be a little subtle, let us examine it closely.

First, $h(\mathbf{x})$ is known as the *underlying measure* or the *base measure*. This can be viewed as an original choice of measure we are modifying with our exponential weight.

Second, we have the vector $\eta = (\eta_1, \eta_2, ..., \eta_l) \in \mathbb{R}^l$ called the *natural parameters* or *canonical parameters*. These define how the base measure will be modified. The natural parameters enter into the new measure by taking the dot product of these parameters against some function $T(\cdot)$ of $\mathbf{x} = (x_1, x_2, ..., x_n) \in \mathbb{R}^n$ and exponentiated. $T(\mathbf{x}) = (T_1(\mathbf{x}), T_2(\mathbf{x}), ..., T_l(\mathbf{x}))$ is called the *sufficient statistics* for η. This name is used since the information represented by $T(\mathbf{x})$ is sufficient to calculate the probability density and no other information from the sample $\mathbf{x}$'s are required.

Third, we have $A(\eta)$, which is referred to as the *cumulant function*, which ensures that the above distribution (18.8.20) integrates to one, i.e.,

$$A(\eta) = \log\left[\int h(\mathbf{x}) \cdot \exp\big(\eta^\top \cdot T(x)\big) dx\right]. \tag{18.8.21}$$

To be concrete, let us consider the Gaussian. Assuming that $\mathbf{x}$ is an univariate variable, we saw that it had a density of

$$\begin{aligned} p(x|\mu,\sigma) &= \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left\{\frac{-(x-\mu)^2}{2\sigma^2}\right\} \\ &= \frac{1}{\sqrt{2\pi}} \cdot \exp\left\{\frac{\mu}{\sigma^2}x - \frac{1}{2\sigma^2}x^2 - \left(\frac{1}{2\sigma^2}\mu^2 + \log(\sigma)\right)\right\}. \end{aligned} \tag{18.8.22}$$

This matches the definition of the exponential family with:

- *underlying measure*: $h(x) = \frac{1}{\sqrt{2\pi}}$,
- *natural parameters*: $\eta = \begin{bmatrix}\eta_1\\ \eta_2\end{bmatrix} = \begin{bmatrix}\frac{\mu}{\sigma^2}\\ \frac{1}{2\sigma^2}\end{bmatrix}$,
- *sufficient statistics*: $T(x) = \begin{bmatrix}x\\ -x^2\end{bmatrix}$, and
- *cumulant function*: $A(\eta) = \frac{1}{2\sigma^2}\mu^2 + \log(\sigma) = \frac{\eta_1^2}{4\eta_2} - \frac{1}{2}\log(2\eta_2)$.

It is worth noting that the exact choice of each of above terms is somewhat arbitrary. Indeed, the important feature is that the distribution can be expressed in this form, not the exact form itself.

As we allude to in Section 3.4.5, a widely used technique is to assume that the final output $\mathbf{y}$ follows an exponential family distribution. The exponential family is a common and powerful family of distributions encountered frequently in machine learning.

Summary

- Bernoulli random variables can be used to model events with a yes/no outcome.
- Discrete uniform distributions model selects from a finite set of possibilities.
- Continuous uniform distributions select from an interval.
- Binomial distributions model a series of Bernoulli random variables, and count the number of successes.
- Poisson random variables model the arrival of rare events.
- Gaussian random variables model the result of adding a large number of independent random variables together.
- All the above distributions belong to exponential family.

Exercises

1. What is the standard deviation of a random variable that is the difference $X - Y$ of two independent binomial random variables $X, Y \sim \text{Binomial}(16, 1/2)$.
2. If we take a Poisson random variable $X \sim \text{Poisson}(\lambda)$ and consider $(X - \lambda)/\sqrt{\lambda}$ as $\lambda \to \infty$, we can show that this becomes approximately Gaussian. Why does this make sense?
3. What is the probability mass function for a sum of two discrete uniform random variables on n elements?

Discussions[259]

18.9 Naive Bayes

Throughout the previous sections, we learned about the theory of probability and random variables. To put this theory to work, let us introduce the *naive Bayes* classifier. This uses nothing but probabilistic fundamentals to allow us to perform classification of digits.

Learning is all about making assumptions. If we want to classify a new data point that we have never seen before we have to make some assumptions about which data points are similar to each other. The naive Bayes classifier, a popular and remarkably clear algorithm, assumes all features are independent from each other to simplify the computation. In this section, we will apply this model to recognize characters in images.

[259] https://discuss.d2l.ai/t/417