PS 8 – Binomial & Poisson distribution

George McNinch

2024-04-05

1. For a whole number $N \ge 1$, recall the following identity:

$$(X+Y)^N = \sum_{j=0}^{N} \binom{N}{j} X^j Y^{N-j}$$

where X, Y are *variables*.

a. Explain why the identity $(X+Y)^N = (Y+X)^N$ implies that

$$\binom{N}{j} = \binom{N}{N-j}$$

for each $0 \le j \le N$.

Solution:

For variables S, T, $\binom{N}{j}$ is the coefficient of $S^j T^{N-j}$ in the expression $(S+T)^N$

Setting $S = X$ and $T = Y$, we see that $\binom{N}{j}$ is the coefficient of $X^j Y^{N-j}$ in $(X+Y)^N$.

On the other hand, setting $S = Y$ and $T = X$, we see that $\binom{N}{N-j}$ is the coefficient of $X^j Y^{N-j}$ in $(Y+X)^N = (X+Y)^N$.

Since equality of polynomials implies equality of their coefficients, we can conclude that $\binom{N}{j} = \binom{N}{N-j}$.

---

b. Explain why the identity

$$\begin{aligned}(X+Y)^N &= (X+Y)(X+Y)^{N-1} \\ &= X(X+Y)^{N-1} + Y(X+Y)^{N-1}\end{aligned}$$

implies that

$$\binom{N}{j} = \binom{N-1}{j} + \binom{N-1}{j-1}$$

for each $0 \le j \le N-1$.

SOLUTION:

On the one hand we have

$$(X+Y)^N = \sum_{j=0}^{N} \binom{N}{j} X^j Y^{N-j}.$$

Thus also

$$(X+Y)^{N-1} = \sum_{j=0}^{N-1} \binom{N-1}{j} X^j Y^{N-1-j}.$$

Now,we of course have

$$\begin{aligned}
(X+Y)^N &= (X+Y)(X+Y)^{N-1} \\
&= (X+Y) \sum_{j=0}^{N-1} \binom{N-1}{j} X^j Y^{N-1-j} \\
&= X \sum_{j=0}^{N-1} \binom{N-1}{j} X^j Y^{N-1-j} + Y \sum_{j=0}^{N-1} \binom{N-1}{j} X^j Y^{N-1-j} \\
&= \sum_{j=0}^{N-1} \binom{N-1}{j} X^{j+1} Y^{N-1-j} + \sum_{j=0}^{N-1} \binom{N-1}{j} X^j Y^{N-j}
\end{aligned}$$

In the *first* sum, re-index with $i = j+1$ so that $j = i-1$; we get

$$\begin{aligned}
&= \sum_{i=1}^{N} \binom{N-1}{i-1} X^i Y^{N-i} + \sum_{j=0}^{N-1} \binom{N-1}{j} X^j Y^{N-j} \\
&= \sum_{j=1}^{N} \binom{N-1}{j-1} X^j Y^{N-j} + j \sum_{j=0}^{N-1} \binom{N-1}{j} X^j Y^{N-j} \qquad = (*)
\end{aligned}$$

Thus we see that the coefficient of $X^j Y^{N-j}$ in $(*)$ is

$$\binom{N-1}{j-1} + \binom{N-1}{j}.$$

Since $(*) = (X+Y)^N$, this permits us to conclude that

$$\binom{N}{j} = \binom{N-1}{j-1} + \binom{N-1}{j}$$

as required.

Hint: In each case, observe what the indicated identity says about the coefficient of $X^j Y^{N-j}$ in the given expression(s).

2. Using the identities in 1., one can argue inductively that

$$\binom{N}{j} = \frac{N!}{j! \cdot (N-j)!}$$

(in fact, this formula has been used in the `notebook`!)

Use this identity to compute the following limits:

$$\lim_{N\to\infty} \frac{1}{N^3} \binom{2N}{3} \qquad \text{and} \qquad \lim_{N\to\infty} \frac{1}{e^N} \binom{N}{N-4}.$$

SOLUTION:

$$\lim_{N\to\infty} \frac{1}{N^3}\binom{2N}{3} = \lim_{N\to\infty} \frac{1}{N^3} \cdot \frac{2N\cdot(2N-1)(2N-2)}{3!}$$
$$= \lim_{N\to\infty} \frac{2\cdot(2-\frac{1}{N})(2-\frac{2}{N})}{3!} = (*)$$

Since $\lim_{N\to\infty}(2-\frac{1}{N}) = \lim_{N\to\infty}(2-\frac{2}{N}) = 2$, we see that

$$(*) = \frac{2^3}{3!} = \frac{8}{6} = \frac{4}{3}.$$

$$\lim_{N\to\infty} \frac{1}{e^N}\binom{N}{N-4} = \lim_{N\to\infty} \frac{1}{e^N}\binom{N}{4}$$
$$= \lim_{N\to\infty} \frac{1}{e^N}\frac{N(N-1)(N-2)(N-3)}{4!} = (\clubsuit)$$

We now observe that, according to *l'hopitals rule*, we have

$$\lim_{N\to\infty} \frac{F(N)}{e^N} = 0$$

for *any* polynomial function F.

Since $F(N) = N(N-1)(N-2)(N-3)$ is a (4th degree) polynomial in N, conclude that $(\clubsuit) = 0$.

3. Suppose you have estimated the probability that your pet songbird will sing during a one-hour time period is `0.35`.

   For the following, you should use the *binomial distribution*.

   a. Indicate an expression for the probability that the bird will sing during a ten-minute time period.

   SOLUTION:

   (You don't really have to use the *binomial distribution*; sorry!)

   If the bird sings with probability p in an hour, the probability that the probability that the bird sings during a 10 minute period is $p/6$.

   b. Indicate an expression for the probability that the bird will not sing during a twenty-minute time period.

   SOLUTION:

   (You don't really have to use the *binomial distribution*; sorry!)

   If the bird sings with probability p in an hour, the probability that the probability that the bird sings during a 20 minute period is $p/3$.

4. Suppose that the probability that an automobile accident occurs during a 24 hour period in a certain stretch of freeway is given by the number p, $0 < p < 1$.

   Assume that the random variable X describing the *number of automobile accidents* is given by the Poisson distribution.

   Thus the probability that there are k accidents is given by

$$P(X = k) = e^{-p} \cdot \frac{p^k}{k!}$$

   Give an expression for the probability that there no more than 3 accidents in a 24-hour period.

SOLUTION:

$$\begin{aligned} P(X \leq 3) &= \sum_{i=0}^{3} P(X = i) \\ &= \sum_{i=0}^{3} e^{-p} \frac{p^i}{i!} \\ &= e^{-p} \left(1 + p + \frac{p^2}{2} + \frac{p^3}{6}\right) \end{aligned}$$

---

5. *Jane's Fish Tank Emporium* (*JFTE*) yet again

   Recall that in a [notebook] last week, we discussed the operation of *JFTE* by considering the question: what is the optimal ordering strategy for fish tanks: ordering *on-demand*?, or putting in place a *standing order*?

   [...]

   Edit the `JFTE` class so that its constructor uses the Poisson distribution to simulate arrival.

   [...]

   Run the simulation 10 times with both strategy functions, as was done in the notebook. Discuss similarities/differences between the results obtained in the notebook.

SOLUTION:

As in the notes, we create the arrival function:

```
import numpy as np
import math as math

def poisson(p,m):
   return (1.*p**m/ math.factorial(m))*np.exp(-p)

from numpy.random import default_rng
rng=default_rng()

def arrival(p=1./7,M = 10,rng=default_rng()):
  qq = list(map(lambda m:poisson(p,m),range(M)))
  qq = qq + [1-sum(qq,0)]

 return rng.choice(list(range(M+1)),p=qq)
```

We make a new class `JFTE_poisson` by copying the `JFTE` class, and replacing the the arrival function in the `__init__` method:

```
class JFTE_poisson():
    def __init__(self,N,prob=1./7):
        self.customers = [ arrival(prob) for n in range(N) ]

        self.num_days = N
        self.reset()

    def reset(self):
        self.stock = 1
        self.sales = 0
        self.lost_sales = 0
        self.storage_days = 0
        self.max_stock = 1

    def add_stock(self):
        self.stock = self.stock + 1
        if self.stock > self.max_stock:
            self.max_stock = self.stock

    def sale(self):
        self.stock = self.stock - 1
        self.sales = self.sales + 1

    def result(self):
        return { 'number_days': self.num_days,
                 'weeks': self.num_days/7.0,
                 'sales': self.sales,
                 'lost_sales': self.lost_sales,
                 'storage_days': self.storage_days,
                 'max_stock': self.max_stock
               }
```

We keep the old `strategy` functions `stand_order` and `order_on_demand`:

```
def stand_order(J,dow=6):
    ## dow = arrival day-of-week for standing order; should be in [0,1,2,3,4,5,6]
    ## we'll assume that the first day of the ``days`` list is dow=0.

    N = J.num_days
    J.reset()

    # loop through the days
    for i in range(N):
        c = J.customers[i]            ## c is 1 if there is a customer on day i, 0 otherwise

        if dow == np.mod(i,7):        ## add stock on the dow for order arrival
            J.add_stock()

        if c>0 and J.stock == 0:
            J.lost_sales = J.lost_sales + 1   ## lost sale if no stock

        if c>0 and J.stock > 0:               ## sale if adequate stock
            J.sale()
```

```
        J.storage_days = J.storage_days + J.stock    ## accumulate total storage costs

    return J.result()

def order_on_demand(J):
    J.reset()
    order_wait = np.inf                         ## order_wait represents wait-time
                                                ## until next order arrival

    ## loop through the customers
    for c in J.customers:
        if c>0 and J.stock==0:                  ## record lost sale if no stock
            J.lost_sales = J.lost_sales + 1

        if c>0 and J.stock>0:                   ## record sale if adequate stock
            J.sale()

        J.storage_days += J.stock               ## accumulate storage days

        if  J.stock==0 and order_wait == np.inf:  ## reorder if stock is empty and no current order
            order_wait = 5

        if order_wait == 0:                     ## stock arrives
            J.add_stock()
            order_wait = np.inf

        if order_wait>0:                        ## decrement arrival time for in-transit orders
            order_wait -= 1

    return J.result()
```

We now create the trials:

```
import pandas as pd

def make_trials(trial_weeks = 2*52, num_trials = 10):
    return [ JFTE(7*trial_weeks) for _ in range(num_trials) ]

def report_trials(strategy,trials):

    results = [ strategy(t) for t in trials ]

    details = ['weeks', 'sales', 'lost_sales', 'storage_days', 'max_stock']

    sd = {i: [r[i] for r in results ] for i in details}

    return pd.DataFrame(sd)

## make a list of 10 trials. Each trial has length 2 years
ten_trials = make_trials()
```

We report on the results of the `standing order` strategy:

```
stand_results = report_trials(stand_order,ten_trials)
print(stand_results)
=>
   weeks  sales  lost_sales  storage_days  max_stock
0  104.0     93           3          6413         17
```

```
1  104.0     98          2          4060         13
2  104.0    101          2          3785         13
3  104.0     99          0          5862         13
4  104.0     91          0          8886         20
5  104.0     78          0         13714         32
6  104.0     97          6          6582         19
7  104.0     99          1          3391         10
8  104.0    101          6          1500          7
9  104.0    103          5          2013          7
```

And we note the `mean()`s:

```
stand_results.mean()
=>
weeks            104.0
sales             96.0
lost_sales         2.5
storage_days    5620.6
max_stock         15.1
dtype: float64
```

Similarly, we report on the `on_demand` strategy:

```
demand_results = report_trials(order_on_demand, ten_trials)
demand_results
=>
     weeks   sales   lost_sales  storage_days    max_stock
0    104.0      62           34           356            1
1    104.0      56           44           393            1
2    104.0      61           42           362            1
3    104.0      56           43           394            1
4    104.0      58           33           380            1
5    104.0      51           27           422            1
6    104.0      62           41           356            1
7    104.0      61           39           362            1
8    104.0      63           44           350            1
9    104.0      61           47           362            1
```

And we note the `mean()` again:

```
weeks           104.0
sales            59.1
lost_sales       39.4
storage_days    373.7
max_stock         1.0
dtype: float64
```

In neither the case of `standing order` nor in `demand_results` did using the poisson distribution have a *large* affect on the outcomes.

In the case of a standing order, The (mean of the) `lost_sales` and `max_stock` arguably decreased slightly.