

# week11-01-multinomial

April 2, 2024

## 1 George McNinch Math 87 - Spring 2024

## 2 Week 11

## 3 Multinomial distributions

The *binomial distribution* appears when modeling a sequence of independent events each having exactly two outcomes. One can instead consider analogous questions for events having some fixed number of outcomes  $m > 2$ . These are known as *multinomial distributions*.

## 4 Example

Consider a nature preserve which has 3 species – A, B, C – of birds.

Among the total population of birds in the preserve, 40% are of species A, 35% are of species B and 25% are of species C.

Each day you travel through the preserve and record the species of the first 10 birds you see. We assume that our observations are *independent* and that the probability of seeing a bird of a particular species is given by the corresponding proportion of the population.

```
[15]: import numpy as np
import math

# lets make a list of all sequences of `a`, `b`, `c` of length n

def seqs(symbols,n):
    if n == 0:
        return [[]]
    else:
        ls = seqs(symbols,n-1)    # recursively get all the sequences of length  $n-1$ 
        result = [ [s] + l for s in symbols for l in ls]
        return result

## for example, here are all the sequences of length 3
seqs(['a','b','c','d'],2)
```

```
[15]: [['a', 'a'],
       ['a', 'b'],
       ['a', 'c'],
       ['a', 'd'],
       ['b', 'a'],
       ['b', 'b'],
       ['b', 'c'],
       ['b', 'd'],
       ['c', 'a'],
       ['c', 'b'],
       ['c', 'c'],
       ['c', 'd'],
       ['d', 'a'],
       ['d', 'b'],
       ['d', 'c'],
       ['d', 'd']]
```

```
[4]: def count(ls,symbol):
      # count the number of occurrences of `symbol` in the sequence ls
      f = [ x for x in ls if x == symbol ]
      return len(f)

count(['a','a','b'], 'a')
```

```
[4]: 2
```

```
[5]: ## for an arrangement
      ## { 'a': na, 'b': nb, 'c': nc }
      ## we count the number of sequences of the symbols ['a','b','c']
      ## of length `n = na + nb + nc`
      ## having 'na' occurrences of 'a',
      ## 'nb' occurrences of 'b'
      ## 'nc' occurrences of 'c'

      def multinomial(arrange):
          symbols = arrange.keys()
          n = sum([arrange[s] for s in symbols ])
          seq = seqs(symbols,n)
          results = [ ls for ls in seq if all([count(ls,s) == arrange[s] for s in
↪symbols ])]
          return len(results)

      [ multinomial({'x': 1, 'y': 1, 'z': 3}), multinomial({'x': 3,'y':0,'z':2}) ]
```

```
[5]: [20, 10]
```

```
[6]: import sympy
(x,y,z) = sympy.symbols('x y z')
f = sympy.Poly((x + y + z)**5)
f
```

```
[6]: Poly(x5 + 5x4y + 5x4z + 10x3y2 + 20x3yz + 10x3z2 + 10x2y3 + 30x2y2z + 30x2yz2 + 10x2z3 + 5xy4 + 20xy3z + 30xy2z2 + 10xz4 + 20xyz3 + 5x2z3 + 10xz2y + 10xz2z + 5y4 + 20y3z + 10y2z2 + 5yz4 + 10yz3z + 5z5)
```

```
[7]: # the sum of all trinomial coefficients should give 3**n
# for example when n = 5

sum([multinomial({'a': a, 'b': b, 'c': 5-a-b}) for a in range(5+1) for b in
↪range(0,5-a+1)]) == 3**5
```

```
[7]: True
```

In fact, there is a *formula* for the multinomial coefficients, similar to that for the binomial coefficients.

For an arrangement { 'a': na, 'b': nb, 'c': nc }, the number of sequences of length n = na + nb + nc having exactly na 'a's, exactly nb b's and exactly nc c's is given by

$$\binom{n}{na, nb, nc} = \frac{n!}{na! \cdot nb! \cdot nc!}$$

This is a **closed form** expression for these coefficients.

Let's define a python function implementing this *closed form*, and let's check that it agrees with the python function we just defined.

```
[19]: def multinomial_alt(arrange):
n = sum([arrange[s] for s in arrange.keys()])
den = np.prod([math.factorial(arrange[s]) for s in arrange.keys()])
return math.factorial(n)/den
```

```
[19]: [(20, 20.0), (10, 10.0)]
```

```
[20]: [ (multinomial(a), multinomial_alt(a)) for a in [{ 'x': 1, 'y': 1, 'z': 3},
{ 'x': 3, 'y': 0, 'z': 2} ]
```

```
[20]: [(20, 20.0), (10, 10.0)]
```

```
[25]: ## again, for m symbols the multinomial coefficients should sum to `m**n`

## so for 4 symbols and 5 choices, we should get `4**5`:
sum([multinomial_alt({'a': a, 'b': b, 'c': c, 'd': 5-a-b-c}) for a in
↪range(5+1) for b in range(0,5-a+1) for c in range(0,5-a-b+1)]) == 4**5
```

```
[25]: True
```

We can now use the multinomial coefficients to describe the probabilities for our bird population.

Remember that  $p_a = .40$  denotes the probability of seeing a bird of type a,  $p_b = .35$  the probability of seeing a b bird, and  $p_c = .25$  the probability of seeing a c bird.

```
[21]: bird_probs = {'a': .4, 'b': .35, 'c': .25 }

def probArrangement(arrange,bird_probs=bird_probs):
    # for a sequence of `n` bird sightings,
    # compute the probability of seeing exactly `na` `a`-birds,
    # exactly `nb` `b`-birds, and exactly `n - na - nb` `c`-birds
    return multinomial_alt(arrange)*np.prod([bird_probs[c]**arrange[c] for c in
    ↪arrange.keys()])
```

```
[22]: a = { 'a': 2, 'b': 2, 'c': 1 }
      probArrangement(a)
```

```
[22]: 0.147000000000000002
```

```
[29]: birdprob10 = {(a,b,10-a-b): probArrangement({'a':a,'b':b,'c':10-a-b}) for a in
    ↪range(10+1) for b in range(10-a+1) }
      birdprob10
```

```
[29]: {(0, 0, 10): 9.5367431640625e-07,
      (0, 1, 9): 1.33514404296875e-05,
      (0, 2, 8): 8.411407470703124e-05,
      (0, 3, 7): 0.0003140258789062499,
      (0, 4, 6): 0.0007693634033203124,
      (0, 5, 5): 0.0012925305175781245,
      (0, 6, 4): 0.001507952270507812,
      (0, 7, 3): 0.0012063618164062494,
      (0, 8, 2): 0.0006333399536132809,
      (0, 9, 1): 0.0001970390966796874,
      (0, 10, 0): 2.7585473535156234e-05,
      (1, 0, 9): 1.52587890625e-05,
      (1, 1, 8): 0.00019226074218749997,
      (1, 2, 7): 0.0010766601562499998,
      (1, 3, 6): 0.003517089843749999,
      (1, 4, 5): 0.007385888671874999,
      (1, 5, 4): 0.010340244140624998,
      (1, 6, 3): 0.009650894531249997,
      (1, 7, 2): 0.005790536718749998,
      (1, 8, 1): 0.002026687851562499,
      (1, 9, 0): 0.00031526255468749983,
      (2, 0, 8): 0.00010986328125000002,
      (2, 1, 7): 0.0012304687500000002,
      (2, 2, 6): 0.006029296875000001,
```

```

(2, 3, 5): 0.01688203125,
(2, 4, 4): 0.029543554687499998,
(2, 5, 3): 0.03308878125,
(2, 6, 2): 0.023162146874999998,
(2, 7, 1): 0.009264858749999999,
(2, 8, 0): 0.0016213502812499993,
(3, 0, 7): 0.0004687500000000001,
(3, 1, 6): 0.004593750000000001,
(3, 2, 5): 0.019293750000000005,
(3, 3, 4): 0.045018749999999996,
(3, 4, 3): 0.06302625,
(3, 5, 2): 0.05294205,
(3, 6, 1): 0.024706289999999995,
(3, 7, 0): 0.004941257999999999,
(4, 0, 6): 0.0013125000000000003,
(4, 1, 5): 0.011025000000000002,
(4, 2, 4): 0.038587500000000004,
(4, 3, 3): 0.07203,
(4, 4, 2): 0.07563149999999999,
(4, 5, 1): 0.04235363999999999,
(4, 6, 0): 0.009882516,
(5, 0, 5): 0.0025200000000000005,
(5, 1, 4): 0.017640000000000003,
(5, 2, 3): 0.049392000000000005,
(5, 3, 2): 0.069148800000000001,
(5, 4, 1): 0.04840416,
(5, 5, 0): 0.013553164799999998,
(6, 0, 4): 0.0033600000000000014,
(6, 1, 3): 0.018816000000000006,
(6, 2, 2): 0.0395136,
(6, 3, 1): 0.036879360000000001,
(6, 4, 0): 0.012907776000000003,
(7, 0, 3): 0.003072000000000001,
(7, 1, 2): 0.012902400000000003,
(7, 2, 1): 0.018063360000000004,
(7, 3, 0): 0.008429568,
(8, 0, 2): 0.001843200000000001,
(8, 1, 1): 0.005160960000000003,
(8, 2, 0): 0.0036126720000000012,
(9, 0, 1): 0.0006553600000000003,
(9, 1, 0): 0.0009175040000000004,
(10, 0, 0): 0.00010485760000000006}

```

```
[30]: # these probabilities should sum to 1 (!)
```

```
sum(birdprob10.values())
```

[30]: 1.0000000000000002

```
[34]: # what is the most likely arrangement??

s=[ (k,birdprob10[k]) for k in birdprob10.keys()]
s.sort(key=lambda x: -x[1])

## top 5
s[:5]
```

```
[34]: [((4, 4, 2), 0.07563149999999999),
      ((4, 3, 3), 0.07203),
      ((5, 3, 2), 0.06914880000000001),
      ((3, 4, 3), 0.06302625),
      ((3, 5, 2), 0.05294205)]
```

```
[36]: ## bottom 5
s[-5:]
```

```
[36]: [((0, 2, 8), 8.411407470703124e-05),
      ((0, 10, 0), 2.7585473535156234e-05),
      ((1, 0, 9), 1.52587890625e-05),
      ((0, 1, 9), 1.33514404296875e-05),
      ((0, 0, 10), 9.5367431640625e-07)]
```

```
[39]: # probability of seeing ten `a` birds
birdprob10[(10,0,0)]
```

[39]: 0.00010485760000000006

```
[ ]:
```