

Re: Estimating the mean of the cumulative hypergeometric?

Re: Estimating the mean of the cumulative hypergeometric?

Source: <http://sci.tech-archive.net/Archive/sci.math/2009-01/msg01705.html>

- *From:* petertwocakes <petertwocakes@xxxxxxxxxxxxxxxx>
 - *Date:* Thu, 15 Jan 2009 06:07:09 -0800 (PST)
-

On 15 Jan, 09:45, Ray Vickson <RGVick...@xxxxxxx> wrote:

On Jan 14, 11:56 pm, petertwocakes <petertwoca...@xxxxxxxxxxxxxxxx> wrote:

On 15 Jan, 03:47, Matt <matt271829-n...@xxxxxxxxxxxx> wrote:

On Jan 14, 6:00 pm, petertwocakes <petertwoca...@xxxxxxxxxxxxxxxx> wrote:

On 13 Jan, 20:19, Matt <matt271829-n...@xxxxxxxxxxxx> wrote:

On Jan 13, 3:43 pm, petertwocakes <petertwoca...@xxxxxxxxxxxxxxxx> wrote:

This is my routine for calculating the cumulative Binomial distribution, with

Re: Estimating the mean of the cumulative hypergeometric?

replacement,
given n and
p:

```
double q =  
1-p, c =  
pow(q,n); //  
i.e. when  
k=0  
double cp =  
c; //  
cumulative  
probability  
for(double  
k=1; k<=n;  
k++)  
{  
    c =  
((n-(k-1))/k)*c*(p/q);  
    cp += c;  
  
}
```

It's fast, and
(for me)
clear to
understand,
and as it's
iterative,
the cost of
calculating
the
cumulative
probability
is little
more than
for
calculating
a specific
probability.
So... I
REALLY
thought I
would be
able to

Re: Estimating the mean of the cumulative hypergeometric?

modify this
to work
with no-
replacement,
after all the
only
difference is
that p & q are
changing as
a result of N
and m being
reduced.
($m =$
number
successes in
 N ; $k =$
number
successes in
sample(n))
Surely, in
the loop
above, if I
could
recalculate
 p & q by how
 N and m
are
changing,
then this
would give
the same
result as the
hypergeometric??

I don't see that this is an
efficient way to approach it.
 p and q are
changing, sure, but they're
changing in a way that
depends on the
history of the previous
selections, and you don't
want to be tracking
all those possibilities
individually.

Re: Estimating the mean of the cumulative hypergeometric?

Your loop code for the hypergeometric should not be that different from the binomial. You keep a running count of the product of the binomial coefficients involving k : this is what I was alluding to earlier. Each iteration involves multiplying by two ratios, something like $c = c * (m - k) / (k + 1) * (n - k) / (N - m - n + k + 1)$, depending on exactly how you implement it. Of course, you can pre-calculate $N - m - n + 1$. You do also have the overhead of initialising this multiplier to $(N - m)C(n)$ and calculating the denominator Nc_n .

Thanks Matt, I've got your's and Ray's iterative method working now.. It's so fast compared to the brute force method of summing across all k s that the overhead you mention isn't a problem. The only catch is that you have to ensure that none of the early values of k from k_0 onward are zero, as naturally the rest of the series will be locked at zero when that happens. I solved this by doing a binary search for the first non-zero value of k .

Not that it matters if you've got it working to your satisfaction, but I don't follow what you mean here. I assume you don't actually mean zero values "of" k (since there's only one). If you mean out-of-range values of k then there's no need to do a binary search; you can

Re: Estimating the mean of the cumulative hypergeometric?

calculate the valid range easily using max and min. If you mean vanishingly small probabilities for certain values of k then I don't see how a binary search can profitably be used when you are calculating the probabilities recursively. Or maybe in the binary search phase you aren't calculating them recursively? Is that really worth doing? Doesn't seem so to me... but I could be missing something.

"If you mean vanishingly small probabilities for certain values of k"

Yes.

The essence of the forward loop (stripping away all checks pre-calcs and initialisation for clarity) is:

```
pk = MyHypergeometricFunction(N,m,n,0);
(...directly calculate the initial hypergeometric value for when k=0)
cumulative_pk = pk;
rk = m * (n/(N-m-n+0+1)); // initial rk;
start = 0;
for(k=start; k<=maxK; k++)
{
    pk = rk*pk;
    rk = ((m-K)/(K+1)) * ((n-K)/(N-m-n+K+1));
    cumulative_pk += pk;
}
}
```

What I meant is that if the first value of pk I calculate (when k=0) is zero, then because in the loop $pk = rk*pk$, pk will remain at zero.

So, why don't you start near the mean, iterating up using $p(k+1)=r(k)*p(k)$ for $d \geq k_0$ and iterating down using $p(k-1) = p(k)/r(k-1)$ for $k \leq k_0$, stopping when they become small (starting with $p(k_0) = 1$). You can even bound the error made by stopping, since the $r(k)$ decrease in

Re: Estimating the mean of the cumulative hypergeometric?

k (being a product of two positive, decreasing factors). So, if you stop at $k_1 \gg k_0$ the dropped terms are smaller than $p(k_1)*r(k_1)$, $p(k_1)*r(k_1)^2, \dots$, so the remainder is bounded above by a geometric series. It is a finite series (it stops at $k = k_{\max}$), but we would get an even looser bound by taking an infinite series, getting $\sum_{k=k_1+1 \dots k_{\max}} p(k) \leq r(k_1)*p(k_1)/(1-r(k_1))$. Similarly, you can easily bound the error in the sum when you drop the $p(k)$ for $k < k_2 \ll k_0$ (because the factors $1/r(k-1)$ are < 1 and decrease as k decreases). So, in computing $\sum(p)$, you get a lower bound by just dropping the terms for $k < k_1$ or $k > k_1$, and you get an upper bound on the sum by estimating the "tails" via geometric series, as indicated above. Therefore, when you compute $S = \sum(p)$ [starting with $p(k_0) = 1$] you have a lower bound S_0 just by dropping terms and an upper bound S_1 by estimating the tails using geometric series. Therefore, you can say for sure (except for roundoff error, of course) that when you re-scale using $p(k) \leftarrow p(k)/S$, you have that $p(k)/S_1 < \text{'true probability } p(k) < p(k)/S_0$. If S_0 and S_1 differ by less than 1 part in a trillion, say, then you know you have a computation of 'true probability $p(k)$ ' that is accurate to maybe 12 digits or more. You can do better by taking more terms before stopping; if the problem is not too large you can go all the way to the end and take all terms with no missing remainders; then your computation would be "exact" (except for roundoff). In a very large problem some of the $p(k)$ for large or small k might be getting near the underflow point, and in any case might not contribute at all to the computed value of $\sum(p)$; in that type of case, stopping makes sense, and as I said already, you can easily bound (or estimate) the error. This up and down procedure is MUCH, MUCH better than one way iteration starting from $k = 0$, say (at least in a large problem). It is very slightly harder to translate into working code, but not by much, and the payoff is worth it.

R.G. Vickson

With the size of numbers I'm working with where n and k are often in the thousands; using double precision, if I directly calculate p_k (using the hypergeometric formula) it will often return zero for $k=500+$.

So the problem becomes: How do I find the first value of k ("start") that would return a non-zero result?

Well, knowing that it will be somewhere between zero and the mean, using binary search it typically only takes 4 or 5 direct hypergeometric calculations to find the smallest value of k that gives a non-zero result.

(I then have to calculate the first multiplier and set the `cumulative_pk`)

Yes, those 4 or 5 are a nuisance (grateful for any suggestions!), but it's nothing compared to doing it thousands of times (when n is large) using the brute-force method.

Re: Estimating the mean of the cumulative hypergeometric?

It's definitely working because I'm cross checking the results against the brute-force method, and the <http://stattrek.com/Tables/Hypergeometric.aspx> calculator. But maybe it could be more efficient still.

You do also have the overhead of initialising this multiplier to $(N - m)C(n)$ and calculating the denominator Nn

As you can see, I'm explicitly calculating the hypergeometric, but it comes to the same thing when $k=0$ doesn't it? (MyHypergeometricFunction () is also very fast because it's calculating the binomials recursively, but not fast enough that I want to do thousands of times)

you can calculate the valid range easily using max and min.

I don't understand this at all, could you elaborate?
(I'm worried this is key to the initialization bottleneck)

Thanks

Steve

I'm a bit embarrassed, because I can't understand any of that I'm afraid.

I don't know what 'scaling' means in this context (I thought it meant some kind of normalizing to 1.0??)

From your earlier post, I skipped the bit about
let $S = \sum(p)$, then re-scale as $p(k) \leftarrow p(k)/S$.

because once I had it working it seemed to make no difference.
Maybe if I understood what that meant, and what it's doing, I would follow you latest post.

So, why don't you start near the mean

Re: Estimating the mean of the cumulative hypergeometric?

From my position of not understanding the scaling, bounds & error

things you mentioned, I'm afraid that looks like a chicken & egg question; I don't know how to start near the mean. Sure I can work out a single probability for when k is close to the mean, but then I wouldn't know the $\text{sum}(P)$ at that point. (Because the whole point of the algorithm is to calculate $\text{sum}(P)$).

OK, I printed out your post out and re-read over many times before replying. I still don't understand the technicalities of it, but is this the general idea behind it: (as usual, make allowances for my limited math vocabulary)

- " Choose a starting value for k near the mean, called k_{start} , ensuring it's less than k_{max} .
- " Calculate $p(k_{\text{start}})$, then iterate backwards toward $k=0$, keeping a sum of $p(k)$ called $\text{sum}(p)$
- " Once you start getting values for p that are very close to zero, STOP.
- " Now go back to $k_{\text{start}} + 1$. Calculate $p(k_{\text{start}} + 1)$
- " Iterate forwards this time towards k_{max} , adding the $p(k)$ each time to $\text{sum}(p)$
- " Once you start getting values for p that are very close to zero, OR you hit k_{max} , then STOP.
- " If k_{max} is much greater than $n/2$, it would be quicker to start near the other end and reverse the directions of all the steps above.

That seems like it should work, I'll code it now, but I've still no idea what 'scaling' means.

Thanks

Steve

.