Pooled testing

Alexander V. Alekseyenko

5/26/2020

Suppose we have to screen N subjects with K testing capacity (K<=N). Also suppose for each subject 1,. . . , N we have (prior) estimates of them being positive, p__1, . . . , p__N. Likewise, assume that up to 30 subjects can be pooled together without loss of sensitivity. See: https://www.thelancet.com/journals/laninf/article/PIIS1473-3099(20)30362-5/fulltext.

Probability that a pool of M subjects contains at least one positive is $\mathcal{P}_M(p_1, \ldots, p_M) = 1 - \prod_i = 1^M(1-p_i)$. Note that $1-(1-min(p_1,\ldots,p_M))^M \leq \mathcal{P}_M(p_1,\ldots,p_M) \leq 1-(1-max(p_1,\ldots,p_M))^M$.

```
pool_positive = function(probs){
  1 - prod(1-probs)
}
```

Assuming that each sample in a pool is to be re-tested if the entire pool tests positive. The expected number of unit tests to screen M (M>1) samples is $\mathcal{E}_M(p_1,\ldots,p_M) =$
$(1-\mathcal{P}_M(p_1,\ldots,p_M)) + \mathcal{P}_M(p_1,\ldots,p_M)(1+M) =$
$1 + M\mathcal{P}_M(p_1,\ldots,p_M)$.

```
expected_tests = function(probs){
  1+pool_positive(probs)*length(probs)
}
```

Note other strategies a possible here. For example, if a pool tests positive it can be sub-pooled into several smaller size pools. This may provide additional efficiency, but may have larger time requirements, so this is not considered here right now.

Example:

Compute the probabilities and the number of tests.

```
ps = seq(from=0.005, to=0.27, by=0.01)
ns = 2:20
res= c()
for(p in ps){
  res = rbind(res,
              c(p, 1, p, 1))
  for(n in ns){
    res = rbind(res,
                c(p, n, pool_positive(rep(p,n)), expected_tests(rep(p,n))))
  }
}
res = as.data.frame(res)
colnames(res) = c("prob_positive", "pool_size", "pool_positive", "expected_tests")
```

Capacity gain is defined as the ratio between the samples needed to be tested and the expected number of tests using the pooling strategy.

```r
res$best = F
res$capacity_gain = res$pool_size/res$expected_tests
for(p in ps){
  res[res$prob_positive ==p,]$best =
    (res[res$prob_positive ==p, ]$capacity_gain == max(subset(res, prob_positive==p)$capacity_gain))
}
```

```r
library(ggplot2)
library(reshape2)
```

```
## Warning: package 'reshape2' was built under R version 3.6.2
```

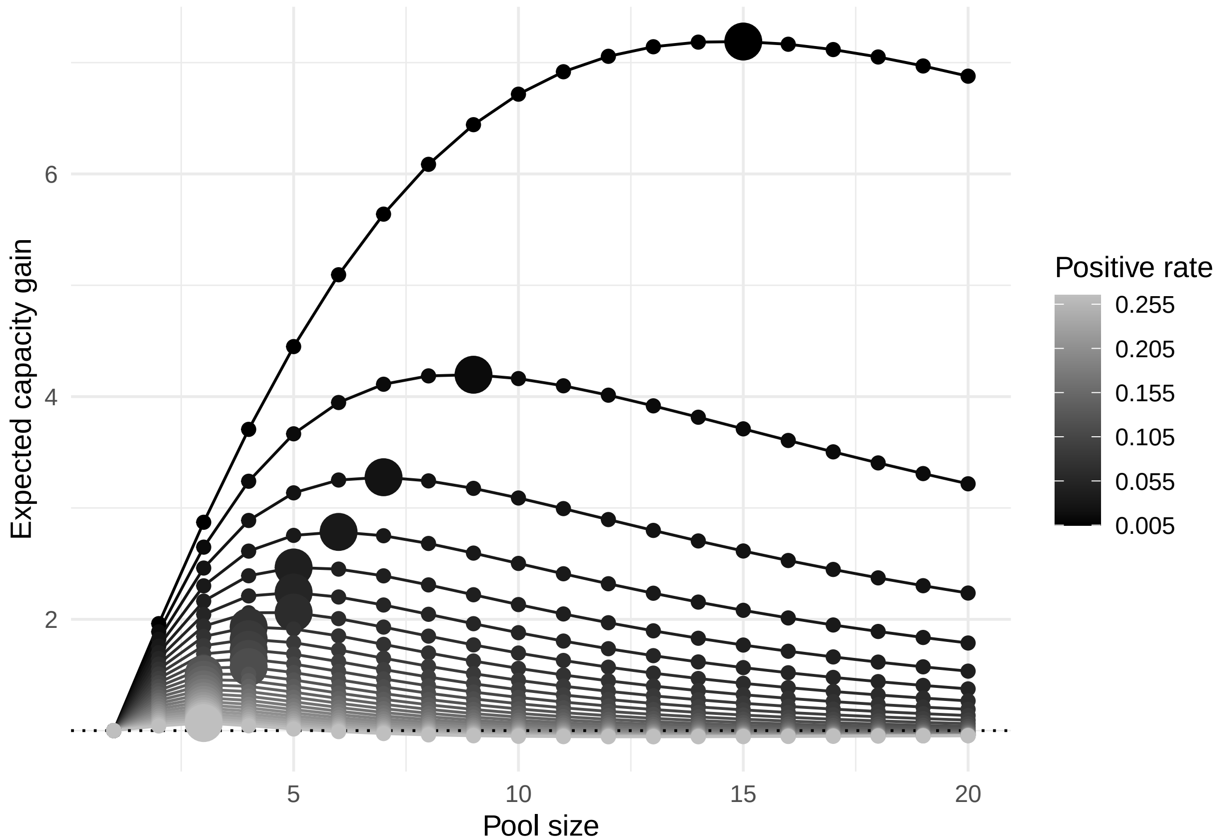
```r
head(res)
```

```
##   prob_positive pool_size pool_positive expected_tests  best capacity_gain
## 1         0.005         1    0.00500000       1.000000 FALSE      1.000000
## 2         0.005         2    0.00997500       1.019950 FALSE      1.960880
## 3         0.005         3    0.01492512       1.044775 FALSE      2.871431
## 4         0.005         4    0.01985050       1.079402 FALSE      3.705756
## 5         0.005         5    0.02475125       1.123756 FALSE      4.449364
## 6         0.005         6    0.02962749       1.177765 FALSE      5.094395
```

```r
cgplot =
ggplot(res, aes(y=capacity_gain, x = pool_size, group=prob_positive, color = prob_positive)) +
  theme_minimal() +
  geom_line() +
  geom_hline(yintercept=1, lty="dotted") +
  geom_point(aes(size=best)) +
  scale_color_gradient(name="Positive rate",
                       breaks = ps[seq(1,27,by=5)],
                       low = "black", high="grey75") +
  scale_size_discrete(guide="none") +
  ylab("Expected capacity gain") +
  xlab("Pool size")
```

```
## Warning: Using size for a discrete variable is not advised.
```

```r
print(cgplot)
```




```r
pdf("../results/cgPool.pdf", width=6, height=5)
print(cgplot)
dev.off()
```

```
## pdf
##   2
```

```r
library(knitr)
kable(subset(res, best))
```

	prob_positive	pool_size	pool_positive	expected_tests	best	capacity_gain
15	0.005	15	0.0724310	2.086466	TRUE	7.189192
29	0.015	9	0.1271772	2.144595	TRUE	4.196597
47	0.025	7	0.1624084	2.136859	TRUE	3.275836
66	0.035	6	0.1924603	2.154762	TRUE	2.784530
85	0.045	5	0.2056409	2.028205	TRUE	2.465235
105	0.055	5	0.2463685	2.231842	TRUE	2.240302
125	0.065	5	0.2854082	2.427041	TRUE	2.060122
144	0.075	4	0.2679059	2.071623	TRUE	1.930853
164	0.085	4	0.2990543	2.196217	TRUE	1.821314
184	0.095	4	0.3291980	2.316792	TRUE	1.726525
204	0.105	4	0.3583589	2.433436	TRUE	1.643766
224	0.115	4	0.3865586	2.546234	TRUE	1.570947
243	0.125	3	0.3300781	1.990234	TRUE	1.507360
263	0.135	3	0.3527854	2.058356	TRUE	1.457474
283	0.145	3	0.3749736	2.124921	TRUE	1.411817
303	0.155	3	0.3966489	2.189947	TRUE	1.369896

	prob_positive	pool_size	pool_positive	expected_tests	best	capacity_gain
323	0.165	3	0.4178171	2.253451	TRUE	1.331291
343	0.175	3	0.4384844	2.315453	TRUE	1.295643
363	0.185	3	0.4586566	2.375970	TRUE	1.262642
383	0.195	3	0.4783399	2.435020	TRUE	1.232023
403	0.205	3	0.4975401	2.492620	TRUE	1.203553
423	0.215	3	0.5162634	2.548790	TRUE	1.177029
443	0.225	3	0.5345156	2.603547	TRUE	1.152274
463	0.235	3	0.5523029	2.656909	TRUE	1.129132
483	0.245	3	0.5696311	2.708893	TRUE	1.107463
503	0.255	3	0.5865064	2.759519	TRUE	1.087146
523	0.265	3	0.6029346	2.808804	TRUE	1.068070

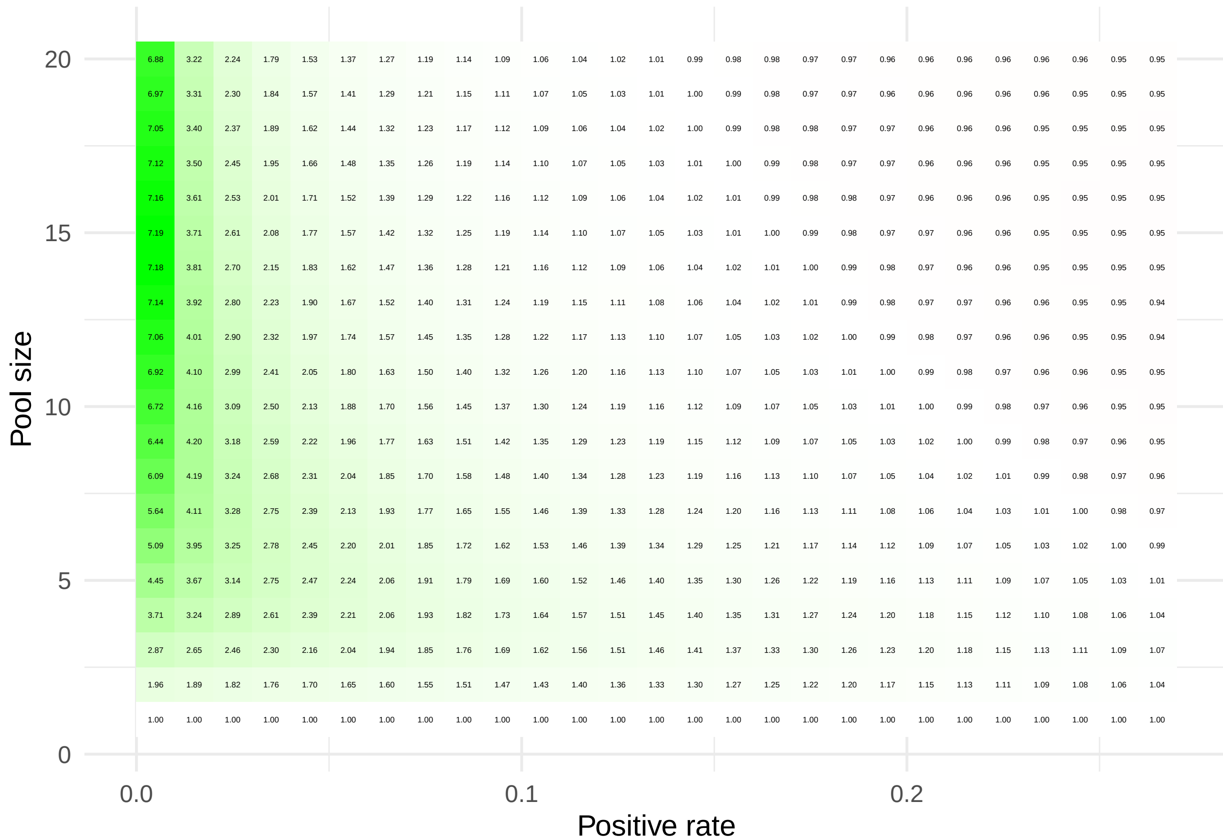
```
best_pool_matrix =
  with(res,
       tapply(capacity_gain,
              list(prob_positive, pool_size),
              max))

bps = data.frame(capacity_gain=apply(best_pool_matrix, 1, max),
      best_pool_size=as.numeric(colnames(best_pool_matrix)[apply(best_pool_matrix, 1, which.max)]))
bps$positive_rate = as.numeric(rownames(bps))
data.frame(pool_size = with(bps, tapply(best_pool_size, best_pool_size, min)),
           min_positive = with(bps, tapply(positive_rate, best_pool_size, min)),
           max_positive = with(bps, tapply(positive_rate, best_pool_size, max)),
           min_capacity_gain = with(bps, tapply(capacity_gain, best_pool_size, min)),
           max_capacity_gain = with(bps, tapply(capacity_gain, best_pool_size, max)))
```

```
##    pool_size min_positive max_positive min_capacity_gain max_capacity_gain
## 3          3        0.125        0.265          1.068070          1.507360
## 4          4        0.075        0.115          1.570947          1.930853
## 5          5        0.045        0.065          2.060122          2.465234
## 6          6        0.035        0.035          2.784530          2.784530
## 7          7        0.025        0.025          3.275836          3.275836
## 9          9        0.015        0.015          4.196597          4.196597
## 15        15        0.005        0.005          7.189192          7.189192
```

```
gg = ggplot(melt(best_pool_matrix),
       aes(fill=value, x=Var1, y=Var2)) +
  geom_tile() +
  scale_fill_gradient2(low="red", mid="white", high="green", midpoint=1) +
  geom_text(aes(label=format(value, digits = 2)), size=1.2) +
  theme_minimal() +
  ylab("Pool size")+xlab("Positive rate") + theme(legend.position = "none")
pdf("../results/posRateVSpoolSize.pdf", width=6, height=5)
print(gg)
dev.off()
```

```
## pdf
##   2
```

```
print(gg)
```



Streaming pooling algorithm

Suppose the current pool is of size i and the probability that the pool is poisitve is p_i. A new specimen with probability of being positive equal to p is proposed to be added to the pool. The probability that the pool containing these $i + 1$ is positive is then $P(p_i, p) = p_i + (1 - p_i) * p$. The expected number of tests for the $i + 1$ specimens is then $P(p_i, p)(1 + i + 1) + (1 - P(p_i, p)) = 1 + P(p_i, p)(1 + i) = 1 + (p_i + (1 - p_i)p)(i + 1)$

```
# Capacity gain for a pool test of n specimens with *pool* positive prob pp
cnp = function(n, pp){
  n/(1 + n*pp)
}

# Probability of pool being positive by adding a new specimen with individual probability pi
# to a pool with *pool* positive probability pp
add1pool_positive = function(pi, pp){
  pp + (1-pp)*pi
}

# Capacity gain by additing a new specimen with individual positvie probability pi
# to a pool of n-1 specimens with *pool* positive probability pp
cnnp = function(pi, pp, n){
  n / (1 + n * add1pool_positive(pi, pp))
}
```

```
pi = 0.05
p = 0.05
```

```r
for(i in 2:10){
  if(cnp(i-1, p) > cnnp(pi, p, i)){
    print(i-1)
    break
  }
  p = add1pool_positive(pi, p)
}
```

```
## [1] 5
```

```r
p
```

```
## [1] 0.2262191
```

```r
pool_positive(rep(pi, 5))
```

```
## [1] 0.2262191
```

```r
c(4, cnp(4, pool_positive(rep(pi, 4))))
```

```
## [1] 4.000000 2.296244
```

```r
c(5, cnp(5, pool_positive(rep(pi, 5))))
```

```
## [1] 5.000000 2.346211
```

```r
c(6, cnp(6, pool_positive(rep(pi, 6))))
```

```
## [1] 6.000000 2.317096
```

```r
c(7, cnp(7, pool_positive(rep(pi, 7))))
```

```
## [1] 7.000000 2.249618
```