6_17_2020 all genes

Ethan Ashby

6/15/2020

Overview

According to Saptarshi, bootstrapping the frequencies (as performed in '06_15_2020_4_genes.Rmd') works for generating SE estimates for seen mutations, but does not extend to SE for unseen mutations. Thus, if we chose to pursue a bootstrap method, we would have to conduct a parametric bootstrap, which assumes that each of the N_r's are modeled. by a Poisson distribution.

We know that the Good-Turing Probabilities for at least one variant is directly a function of N_1. What about the SE?

Goal: generate Good-Turing estimates for all genes in the TCGA dataset (non-hypermutated samples) that appear at some frequency threshold (relative freq>0.1 or >10 occurrences). Keep track of SE's and plot against N_1 or rare variants ($\sum N_r$ for $r = 1, 2, 3$). Also keep track of the number of times we achieve a problematic slope estimate (b>-1).

Installing new variantprobs package

Took some finagling. I updated R, the commandline tools. But ultimately it took deleting a 'Makevars' file from '~/usr/.R' to get the package to compile.

goodTuring_SE function

I wrote a function which takes in a gene name, and then outputs a list with four elements: (i) a vector of good-turing probabilities for all seen and unseen variants, (ii) the Chao estimate of N0, (iii) the SE of the Good-Turing estimates, (iv) whether the smoothing algorithm provided a logical beta estimate. I used the function 'mclapply' with 8 cores to apply this function over a list of genes with a mutation rate >0.03 (as in the original Somatic Variant richness paper). Runtime was fast and the output was a list containing all the above information.

```
#####
#Filter genes w/ frequency>0.03
#####

num_samps<-length(unique(tcga_nh$patient_id))
#filter variants where mutation rate (num_variants/num_samples)>0.03
filtered_variants<-tcga_nh %>% dplyr::group_by(Hugo_Symbol) %>% dplyr::filter(dplyr::n()/num_samps>0.03)

#this produces 858 genes
```

```
filtered_variants$Hugo_Symbol %>% unique() %>% length()

#now define your gene list as these filtered genes
genelist<-filtered_variants$Hugo_Symbol %>% unique()
#takes ~10 seconds to run
variant_stats<-c(mclapply(genelist, goodTuring_SE, mc.cores=8))
names(variant_stats)<-genelist
```

Exploring results

Of our 45 genes with adequate mutation rate, 45 achieved acceptable Good-Turing estimates ($\beta < -1$).

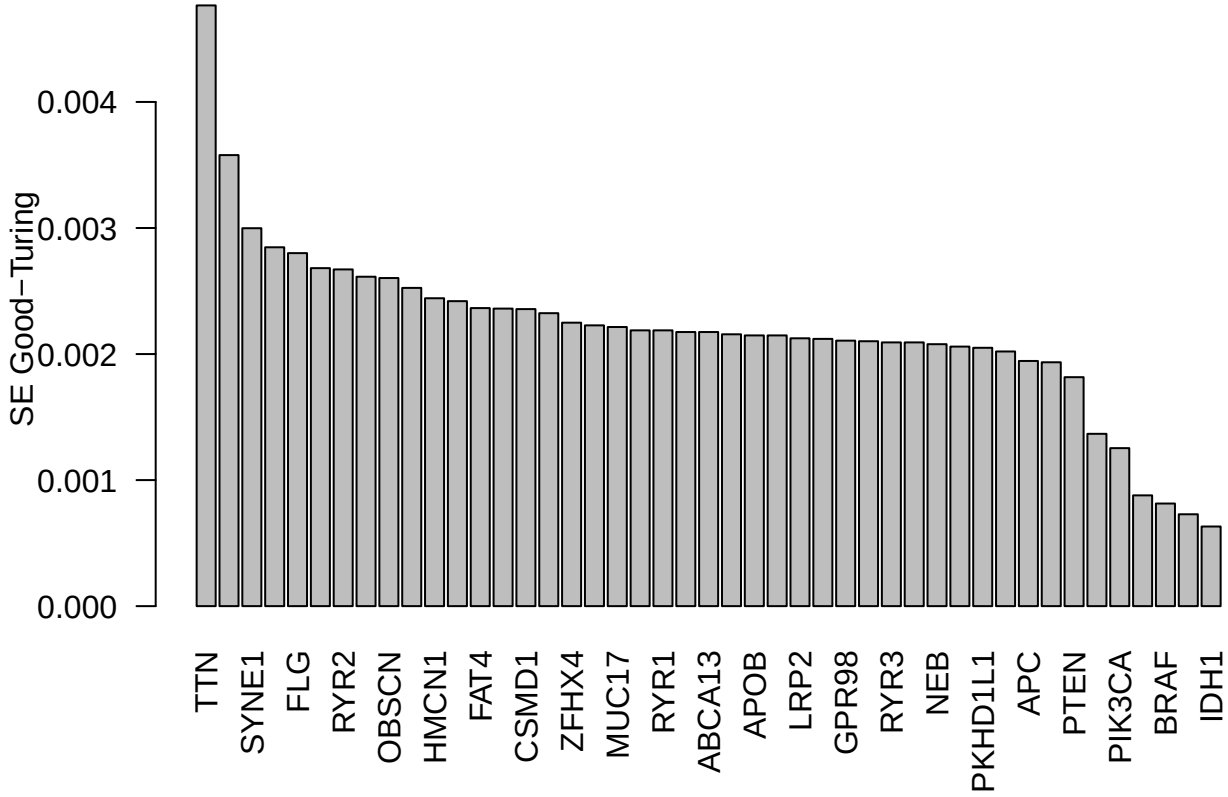

Figure 1: Barplot of ordered Good-Turing standard errors for 45 genes that met mutation frequency criteria

The majority of Good-Turing standard errors fell within [0.002, 0.003] (*Figure 1*). A density plot of the Good-Turing standard errors shows positive kurtosis 3.3529303(relatively less weight in tails) a slight right-skew 0.5401474 (*Figure 2.*)

A plot of the Good-Turing standard errors against the gene-wise N_1 values indicated that these standard errors are a logarithmic function of N_1 (*Figure 3*).

To further investigate this, I fit a variety of logarithmic functions to these data using the package in R. The best fit I obtained was a Log-logistic function with 3 parameters, that practically interpolated the data (sum of squared residuals=1.8560061×10^{-8}).

$$f(x) = 0 + \frac{d - 0}{1 + \exp(b * (\log(x) - e))}$$

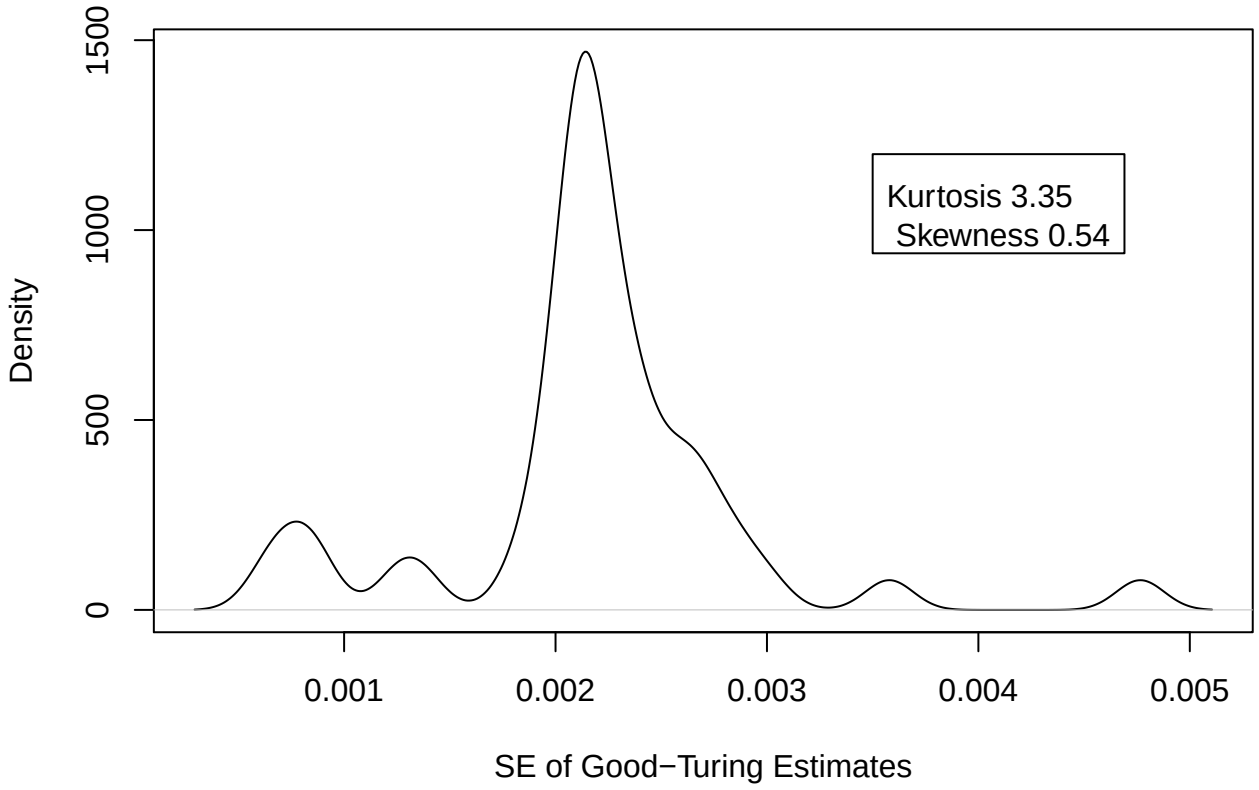

Figure 2: Right-skewed density of Good-Turing Standard Errors

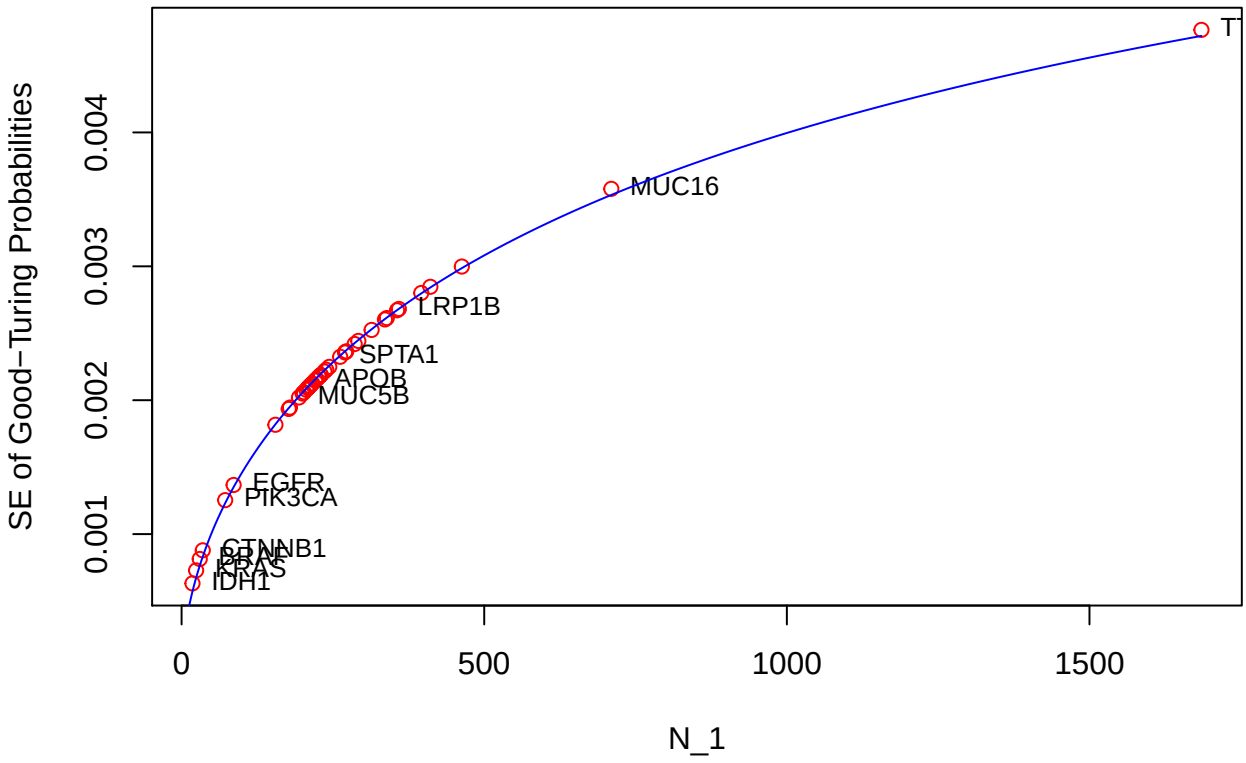

Figure 3: Relationship between N_1 and SE of Good-Turing Unseen Variants Probabilities

The parameter values for log-logistic function are shown in the table below.

I validated that my bootstrap method which samples variant frequencies does not perform well at capturing the SE of P(one or more unseen variant).