A basic error model for UMI-tagged data

Mikhail Shugay

November 29, 2016

Control experiment

Some auxiliary functions

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(RColorBrewer)
library(stringr)
library(nloptr)
library(scales)

select <- dplyr::select

mtypes <- data.frame(mutation.fromto = c("A>C","A>G","A>T","C>A","C>G","C>T","G>A",
                                         "G>C","G>T","T>A","T>C","T>G"),
                     mutation.type = c("A>C,T>G","A>G,T>C","A>T,T>A","C>A,G>T",
                                       "C>G,G>C","C>T,G>A","C>T,G>A","C>G,G>C",
                                       "C>A,G>T","A>T,T>A","A>G,T>C","A>C,T>G"))

read_variant_table <- function(file_name, pos_filter = function(x) T,
                               count_filter = function(x) T) {
  .df <- read.table(file_name, header=T, sep="\t", stringsAsFactors = F)
  .df <- subset(.df, !grepl("^[DI]", mutation) & coverage > 100 & freq < 0.45)

  .df$mutation.fromto <- unlist(lapply(str_split(.df$mutation, ":"), function(x) x[2]))
  .df$mutation.pos <- as.integer(unlist(lapply(str_split(.df$mutation, ":"),
                                               function(x) str_sub(x[1], 2, nchar(x[1])))))
  .df <- merge(.df, mtypes)
  .df %>%
    filter(pos_filter(mutation.pos) & count_filter(count)) %>%
    select(mutation.pos, mutation.type, count, coverage)
}
```

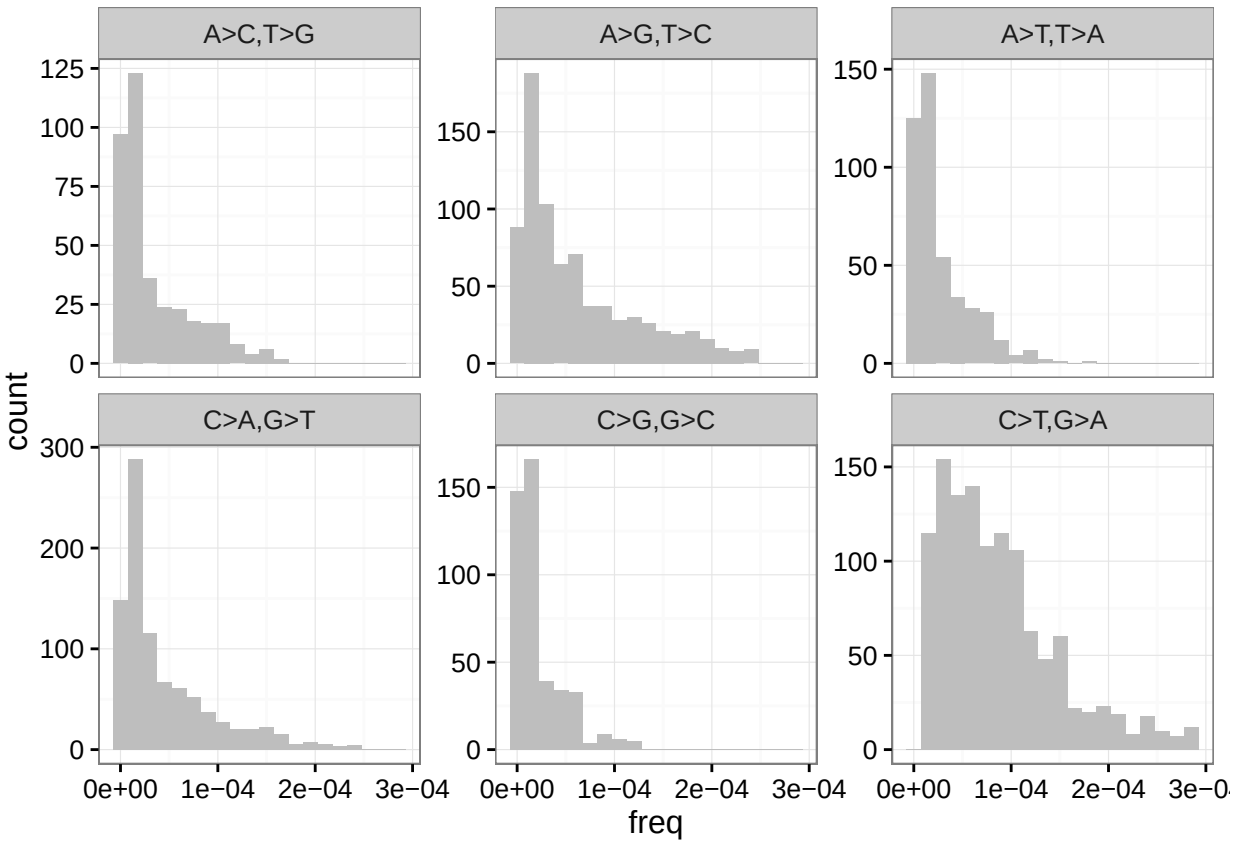
Load data from 2 independent experiments with library preparation performed using 10 different PCR assays: a known synthetic template sequence that was UMI-tagged and amplified using a set of different polymerases. At this stage a single event is a combination of substituted nucleotide, its substitution, position in template, sample (polymerase) and project (replica). Samples are obtained by grouping all events by substitution

type which is one of 6 from and to nucleotide combinations that can be observed when not knowing the exact strand at which a given error has happened. Note that the latter is done because in most practical applications this information is hard to obtain.

```
df <- data.frame()

samples <- c("encyclo", "kappa-hf-taq", "phusion", "sd-hs", "snp-detect",
                   "taq-hs", "tersus", "tersus-snp-buff", "truseq", "velox")

for (proj in c("73", "82")) {
  for (sample in samples) {
    file_name <- paste("data/25_8_polerr", proj, ".", sample,
                       ".variant.caller.txt", sep = "")

    # ignore positions with no errors
    # this includes primers/barcodes
    .df <- read_variant_table(file_name,
                              count_filter=function(x) x>0)
    .df$proj <- proj
    .df$sample <- sample
    df <- rbind(df, .df)
  }
}
```

Plot error frequencies grouped by substitution type

```
df.1 <- df %>%
  select(mutation.type, count, coverage) %>%
  mutate(freq = count / coverage) %>%
  # winsorize data
  group_by(mutation.type) %>%
  mutate(q5 = quantile(freq, 0.05), q95 = quantile(freq, 0.95)) %>%
  filter(freq >= q5 & freq <= q95) %>%
  select(mutation.type, freq)

ggplot(df.1, aes(x=freq)) + geom_histogram(fill="grey", bins=20) +
  facet_wrap(~mutation.type, scales="free_y") +
  theme_bw()
```



Estimating error rate distribution parameters for various substitution types

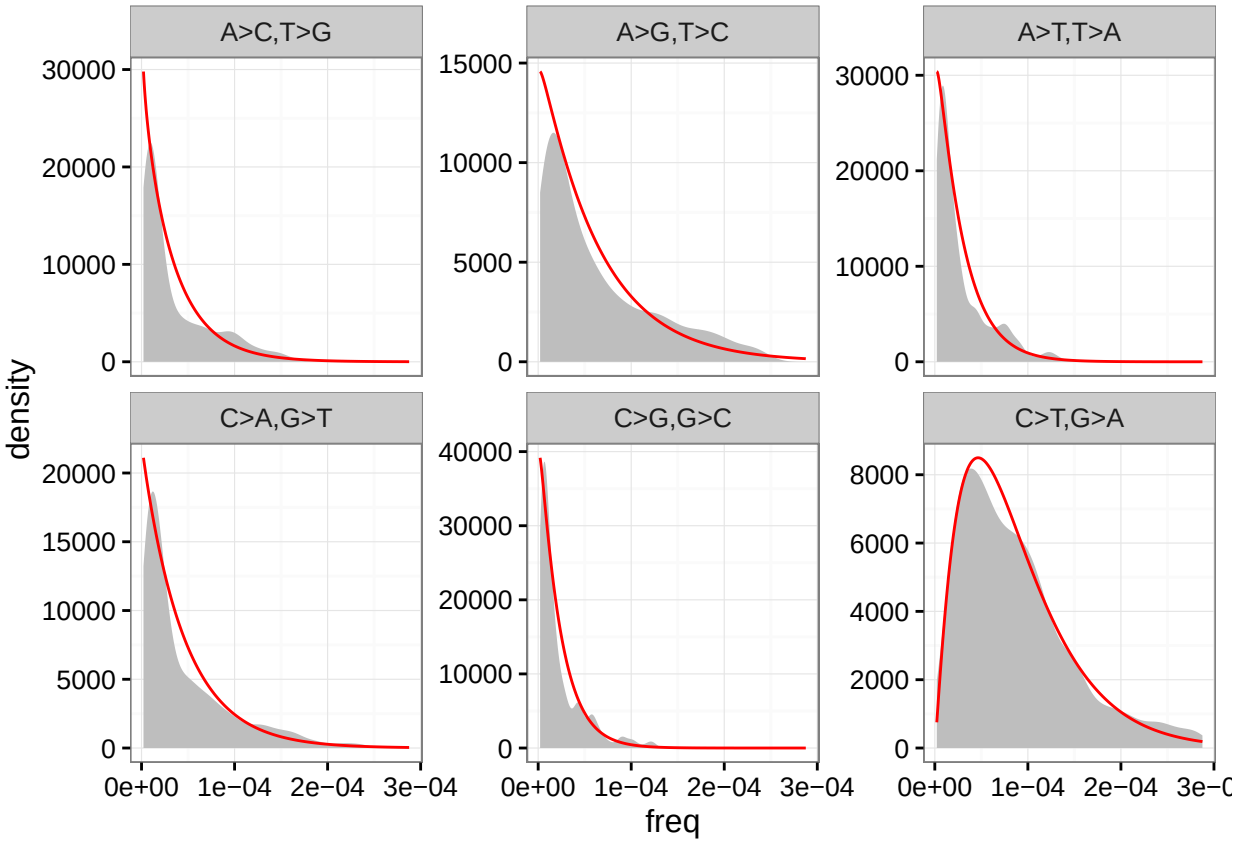
Fit frequency distributions with a Beta model.

```
betanll <- function(pars, data){
  alpha <- pars[[1]]
  beta <- pars[[2]]
  return (-sum(dbeta(data, shape1 = alpha, shape2 = beta, log = T)))
}

param_guess = function(data) {
  m = mean(data)
  v = sd(data) ^ 2
  c(m, (1 - m)) * (m * (1 - m) / v - 1)
}

fit_beta <- function(data) {
  list(nloptr(x0 = param_guess(data), eval_f = betanll,
              lb = c(0,0), data = data,
              opts = list(algorithm = "NLOPT_LN_SBPLX",
                          maxeval = 1e5))$solution)
}

fit.params <- df.1 %>%
  group_by(mutation.type) %>%
  summarize(fit = fit_beta(freq))
```

```
fit.params$alpha <- unlist(lapply(fit.params$fit, function(x) x[1]))
fit.params$beta <- unlist(lapply(fit.params$fit, function(x) x[2]))
fit.params$fit <- NULL
```

Checking goodness of fit. Note that poor fit in the vicinity of 0 can be attributed to sampling effect under finite coverage.

```
df.1.fit <- expand.grid(mutation.type = unique(df.1$mutation.type),
                        freq = seq(min(df.1$freq), max(df.1$freq),
                                   length.out = 1000))

df.1.fit <- merge(df.1.fit, fit.params)

df.1.fit$density <- with(df.1.fit, dbeta(freq, shape1 = alpha, shape2 = beta))


ggplot(data=df.1, aes(x=freq)) +
  geom_density(fill="grey", color=NA) +
  geom_line(data=df.1.fit, aes(y=density), color="red") +
  facet_wrap(~mutation.type, scales="free_y") +
  theme_bw()
```



The probability of error p_T for each subsitution type group T is fitted with a $Beta$ distribution

$$p_T \sim Beta(\alpha_T, \beta_T)$$

with parameters

```
print(fit.params)
```

```
## # A tibble: 6 × 3
##   mutation.type     alpha     beta
##          <fctr>     <dbl>    <dbl>
## 1       A>C,T>G 0.9278882 26823.24
## 2       A>G,T>C 1.0295689 16394.12
## 3       A>T,T>A 1.0838020 38974.43
## 4       C>A,G>T 0.9957058 21862.23
## 5       C>G,G>C 1.0584582 48029.97
## 6       C>T,G>A 2.1314829 24344.82
```

Thus the count of an error of a certain type n_T at the coverage N is distributed as

$$n_T \sim BetaBinom(N, \alpha_T, \beta_T)$$

Benchmark using an independent control sample

Load control data from UMI-tagged amplicon sequencing of control donor DNA

```
df.control <- data.frame()
for (r in 1:2) {
  for (m in 1:4) {
    file_name <- paste("data/p127.ballast", r, "_m", m, ".variant.caller.txt",
                       sep = "")
    df.control <- rbind(df.control, read_variant_table(file_name))
  }
}
```

Observed (black) and fitted (red) distribution of error ocurrences by error count

```
library(TailRank)
```

```
## Loading required package: oompaBase
```

```
alpha <- fit.params$alpha
names(alpha) <- fit.params$mutation.type
beta <- fit.params$beta
names(beta) <- names(alpha)

compute_p <- function(count, coverage, mutation.type) {
  mapply(function(k, n, a, b) dbb(k, n, a, b),
         count, coverage, alpha[mutation.type], beta[mutation.type])
}

df.control.1 <- df.control %>%
  group_by(mutation.type) %>%
  mutate(total = n()) %>%
  group_by(count, mutation.type) %>%
  mutate(coverage.med = as.numeric(median(coverage))) %>%
  group_by(count, mutation.type, coverage.med, total) %>%
  summarise(freq = n()) %>%
  mutate(freq.fit = total * dbb(count, coverage.med,
                                alpha[mutation.type], beta[mutation.type]))
```

```
ggplot(df.control.1, aes(x = count)) +
  geom_line(aes(y=freq), linetype="dashed")  +
  geom_point(aes(y=freq))  +
  geom_line(aes(y=round(freq.fit)), color="red") +
  scale_x_continuous("Erroneous variant count", limits=c(0,6), breaks = 0:6) +
  ylab("Occurrences") +
  facet_wrap(~mutation.type) +
  theme_bw()
```

```
## Warning: Removed 5 rows containing missing values (geom_point).
```