Benchmark of error rate inference from UMI-tagged data and PCR error model

Mikhail Shugay

November 6, 2016

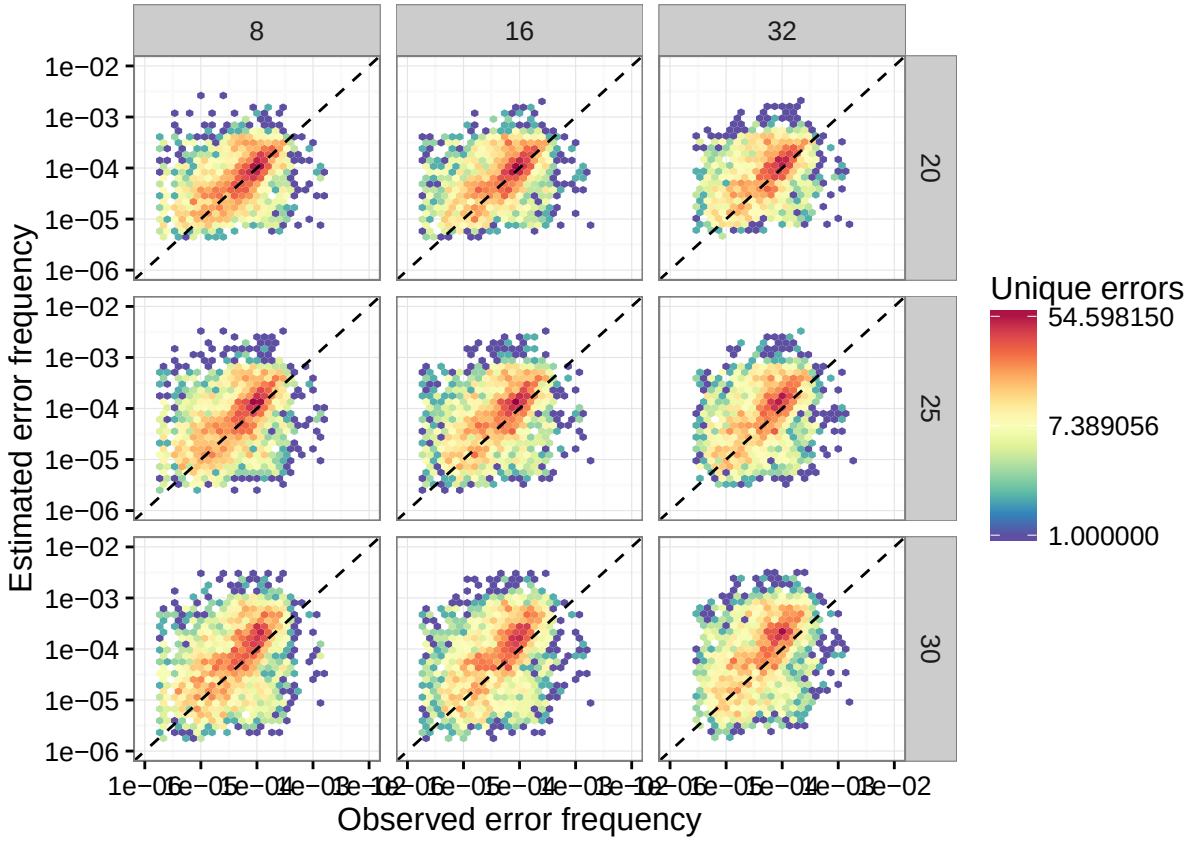
Load data from 2 independent experiments with 10 different PCR assays.

```
library(plyr)
library(ggplot2)
library(RColorBrewer)

df <- data.frame()

for (q in c(20, 25, 30)) {
  for (m in c(8, 16, 32)) {
    for (proj in c("73", "82")) {
      for (sample in c("encyclo", "kappa-hf-taq", "phusion", "sd-hs", "snp-detect",
                       "taq-hs", "tersus", "tersus-snp-buff", "truseq", "velox")) {
        .df <- read.table(
            paste("data",
              paste(paste(q, m, paste("polerr", proj, sep=""), sep="_"), sample, "variant.caller.txt", s
                    sep ="/"),
            header=T, sep="\t", stringsAsFactors = F)
        .df$q <- q
        .df$m <- m
        .df$proj <- proj
        .df$sample <- sample
        df <- rbind(df, .df)
      }
    }
  }
}

df <- subset(df, count > 0 &
               !grepl("D", mutation) & !grepl("I", mutation) & global.est == 0 &
               coverage > 0)

df$mut.split <- sapply(df$mutation, function(x) strsplit(as.character(x), "[S:>]"))
df$mutation.pos <- as.integer(sapply(df$mut.split, function(x) x[2]))
df$mutation.from <- sapply(df$mut.split, function(x) x[3])
df$mutation.to <- sapply(df$mut.split, function(x) x[4])
df$mut.split  <- NULL
```

Model estimates for error rate:

```
rf <- colorRampPalette(rev(brewer.pal(11, 'Spectral')))
r <- rf(32)

ggplot(df, aes(x=freq, y=error.rate)) +
  stat_binhex() +
  geom_abline(intercept = 0, slope=1, linetype="dashed") +
```

```
  scale_x_log10("Observed error frequency", limits=c(1e-6, 1e-2),
                breaks=c(1e-6, 1e-5, 1e-4, 1e-3, 1e-2)) +
  scale_y_log10("Estimated error frequency", limits=c(1e-6, 1e-2),
                breaks=c(1e-6, 1e-5, 1e-4, 1e-3, 1e-2)) +
  facet_grid(q~m) +
  scale_fill_gradientn("Unique errors", colors=r, trans="log") +
  theme_bw()
```




```
a <- aov(log10(freq) ~ I(log10(error.rate)) + q + m + proj, df)
summary(a)
```

```
##                          Df Sum Sq Mean Sq  F value   Pr(>F)
## I(log10(error.rate))      1   1435  1434.5 5769.797  < 2e-16 ***
## q                         1      9     9.4   37.988 7.21e-10 ***
## m                         1      6     6.3   25.189 5.23e-07 ***
## proj                      1      0     0.2    0.776    0.378
## Residuals             30074   7477     0.2
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
for (mm in unique(df$m)) {
  for (qq in unique(df$q)) {
    print(paste("m =", mm , "q =", qq, with(subset(df, q == qq & m == mm), cor(freq, error.rate, method
  }
}
```

```
## [1] "m = 8 q = 20 0.439187907278551"
## [1] "m = 8 q = 25 0.440171986104134"
```

```
## [1] "m = 8 q = 30 0.437514996064882"
## [1] "m = 16 q = 20 0.419952825360284"
## [1] "m = 16 q = 25 0.416393783658744"
## [1] "m = 16 q = 30 0.420338819999655"
## [1] "m = 32 q = 20 0.38017513895925"
## [1] "m = 32 q = 25 0.399270957195644"
## [1] "m = 32 q = 30 0.40434020987666"
```

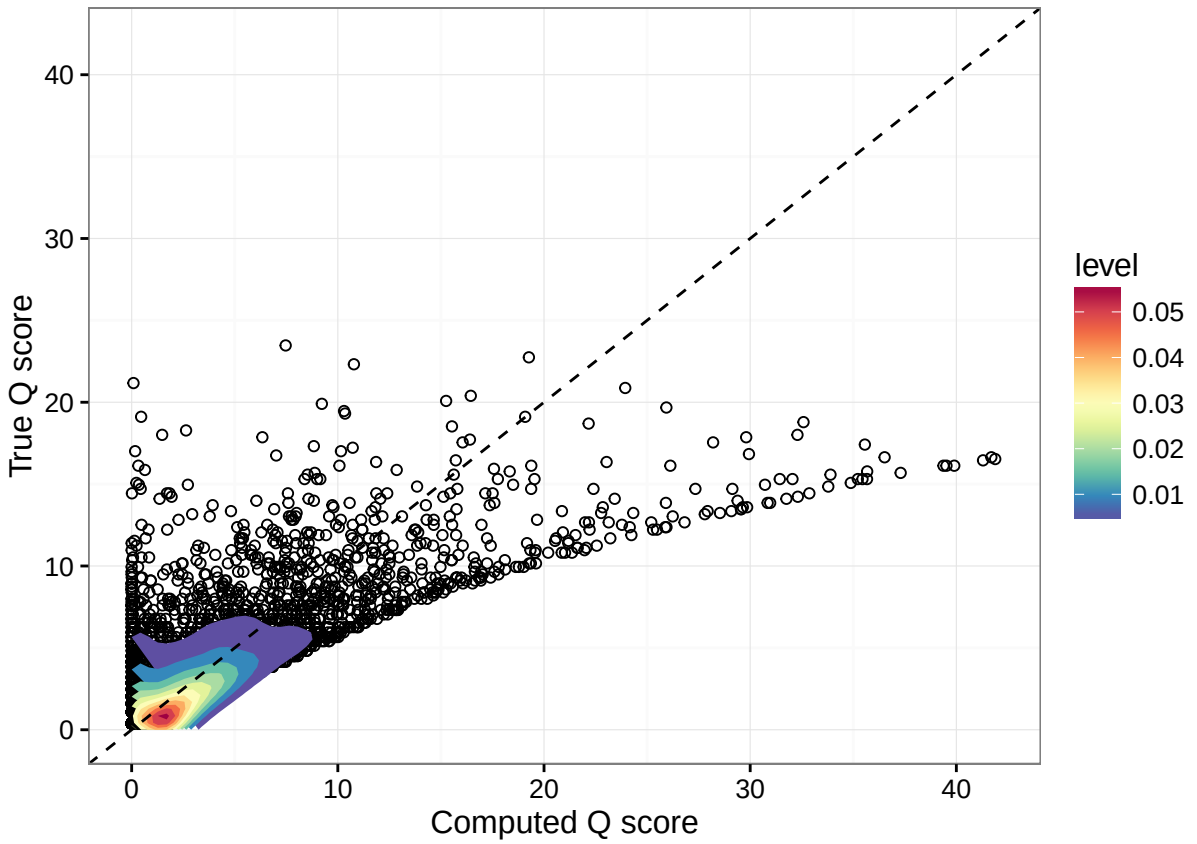
Computing error rate statistics:

```
#
df.1 <- subset(df, q == 25 & m == 8)

#lambda.m <- 5.2505409 + 0.3545431 * round(df.1$error.rate * df.1$coverage)
#lambda.var <- 35.089371 + 3.886613 * round(df.1$error.rate * df.1$coverage)

lambda.m <- 5.2505409 + 0.3545431 * with(df.1, minor.count.local * read.fraction.in.minors * (1 + 1 / m
eff <- 0.8
lambda.var <- 35.089371 + 3.886613 * lambda.m

df.1$alpha <- lambda.m^2 / lambda.var
df.1$beta <- lambda.m / lambda.var

#df.1$phi <- df.1$minor.count.local * df.1$read.fraction.in.minors
#n <- 20
#lambda <- 0.8
#df.1$alpha <- df.1$phi * (1 + lambda^2) / (1 + lambda)^2 / (df.1$phi / n / lambda + 1)
#df.1$beta <- (1 + lambda^2) / (1 + lambda)^2 / (df.1$phi / n / lambda + 1)
```

```
df.1$pval <- with(df.1,
                  1.0 - pnbinom(count, prob = beta / (1 + beta), size = alpha) +
                  0.5 * dnbinom(count, prob = beta / (1 + beta), size = alpha))
#df.1$pval <- with(df.1,
#                  1.0 - pbinom(count, prob = error.rate, size = coverage) +
#                    0.5 * dbinom(count, prob = error.rate, size = coverage))

df.count.summary <- ddply(df.1, .(count), summarize, weight = length(count))

df.1$pval.true <- with(df.count.summary,
                       sapply(df.1$count,
                         function(x)
                           sum(ifelse(x > count, 0, ifelse(x == count, 0.5 * weight, weight)))) / sum(weig

ggplot(df.1, aes(x=-10*log10(pval), y=-10*log10(pval.true))) +
  geom_point(shape=21) +
  stat_density2d(aes(fill=..level..), geom="polygon") +
  geom_abline(intercept = 0, slope = 1, color = "black", linetype="dashed") +
  scale_x_continuous("Computed Q score", limits=c(0,42)) +
  scale_y_continuous("True Q score", limits=c(0,42)) +
  scale_fill_gradientn(colors=r) +
  theme_bw()
```

```
## Warning: Removed 48 rows containing non-finite values (stat_density2d).
```

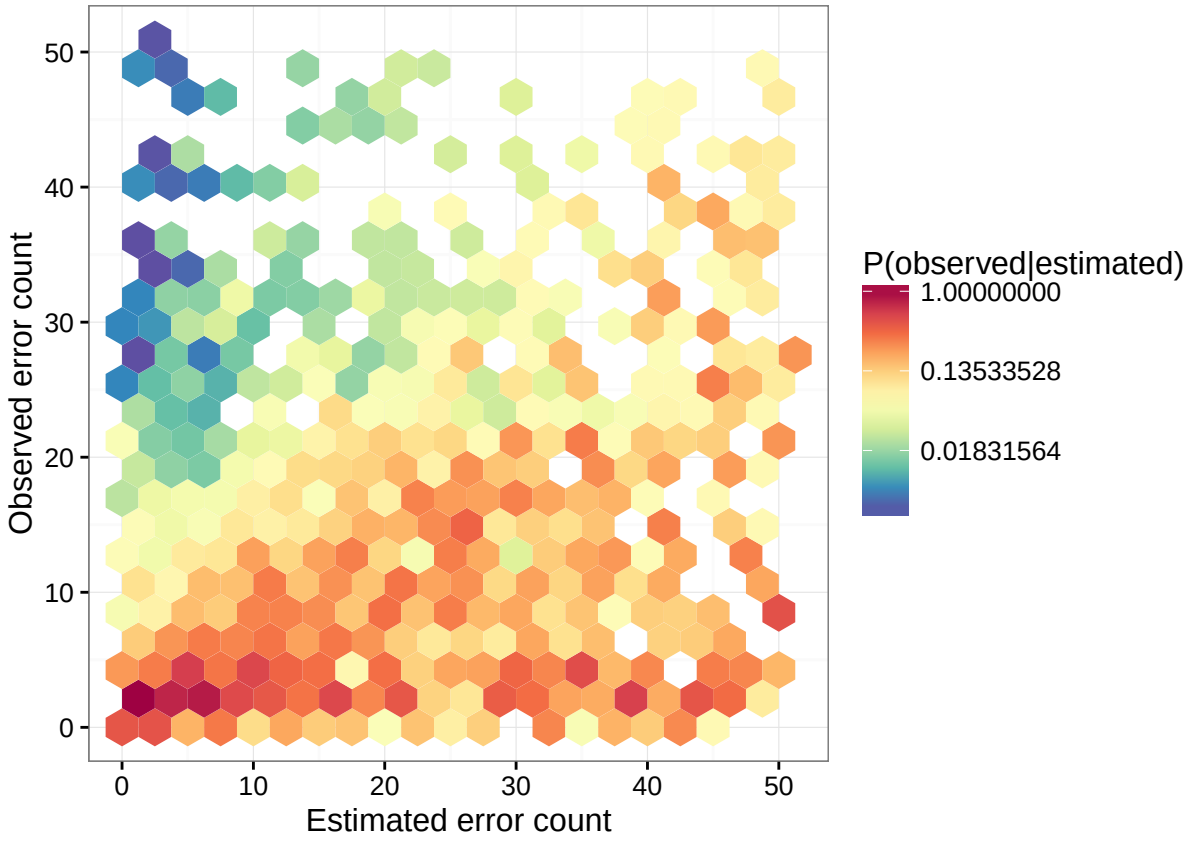
```
## Warning: Removed 48 rows containing missing values (geom_point).
```



We'll go Bayesian way to compute P-values (and Q-scores) of erroneous variants accounting for the indirect minor-based error model (MBEM) we apply to infer the error rate at a given position. We'll assume that observed erroneous variant counts are distributed according to Poisson distribution with `lambda` parameter that is estimated as `MBEM error rate * coverage`, modelling the uncertainity in `lambda` using Gamma distribution. Thus, we'll arrive to a Negative Binomial distribution for our final P-values estimates. First lets plot the conditional probability `P(lambda|MBEM error rate * coverage)`:

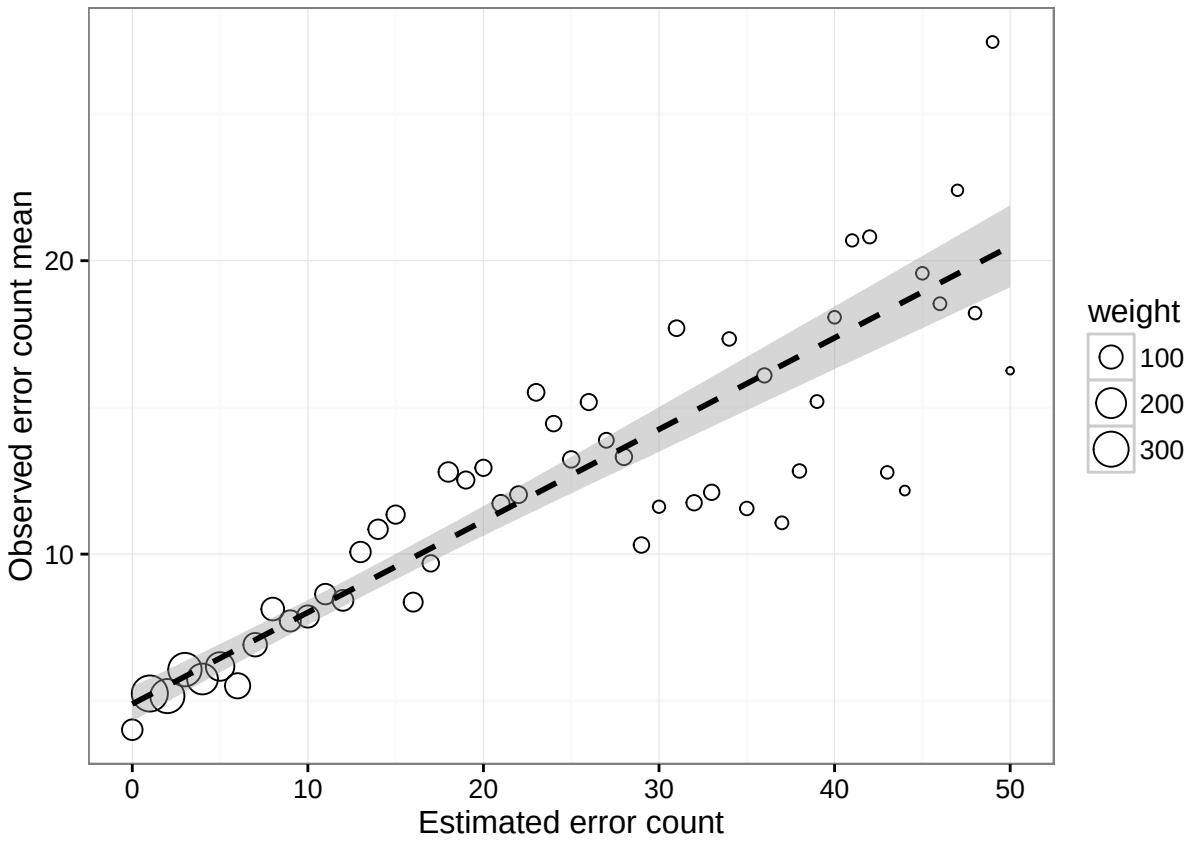
```
df.1 <- subset(df, q == 25 & m == 8)

df.1$count.est <- round(df.1$coverage * df.1$error.rate)
df.1$count.obs <- round(df.1$coverage * df.1$freq)
df.1 <- subset(df.1, count.est <= 50 & count.obs <= 50)

# Compute mean and variance for lambda, we'll need it later
df.1 <- ddply(df.1, .(count.est), transform, f.mean = mean(count))
df.1 <- ddply(df.1, .(count.est), transform, f.var = var(count))

df.1.s <- ddply(df.1, .(count.est, count.obs),
                summarize,
                weight = length(count.est), f.mean = f.mean[1], f.var = f.var[1])
df.1.m <- ddply(df.1.s, .(count.est), transform, weight.norm = weight / sum(weight))

# requires install.packages("hexbin")

ggplot(df.1.m, aes(x=count.est, y=count.obs, weight=weight.norm)) +
  geom_hex(bins = 20) +
  xlab("Estimated error count") +
```

```
  ylab("Observed error count") +
  scale_fill_gradientn("P(observed|estimated)", colors=r, trans="log") +
  theme_bw()
```



Now lets fit it with Gamma distribution estimating `alpha` and `beta` parameters using linear regression of mean and variance of `lambda` against `MBEM error rate * coverage`:

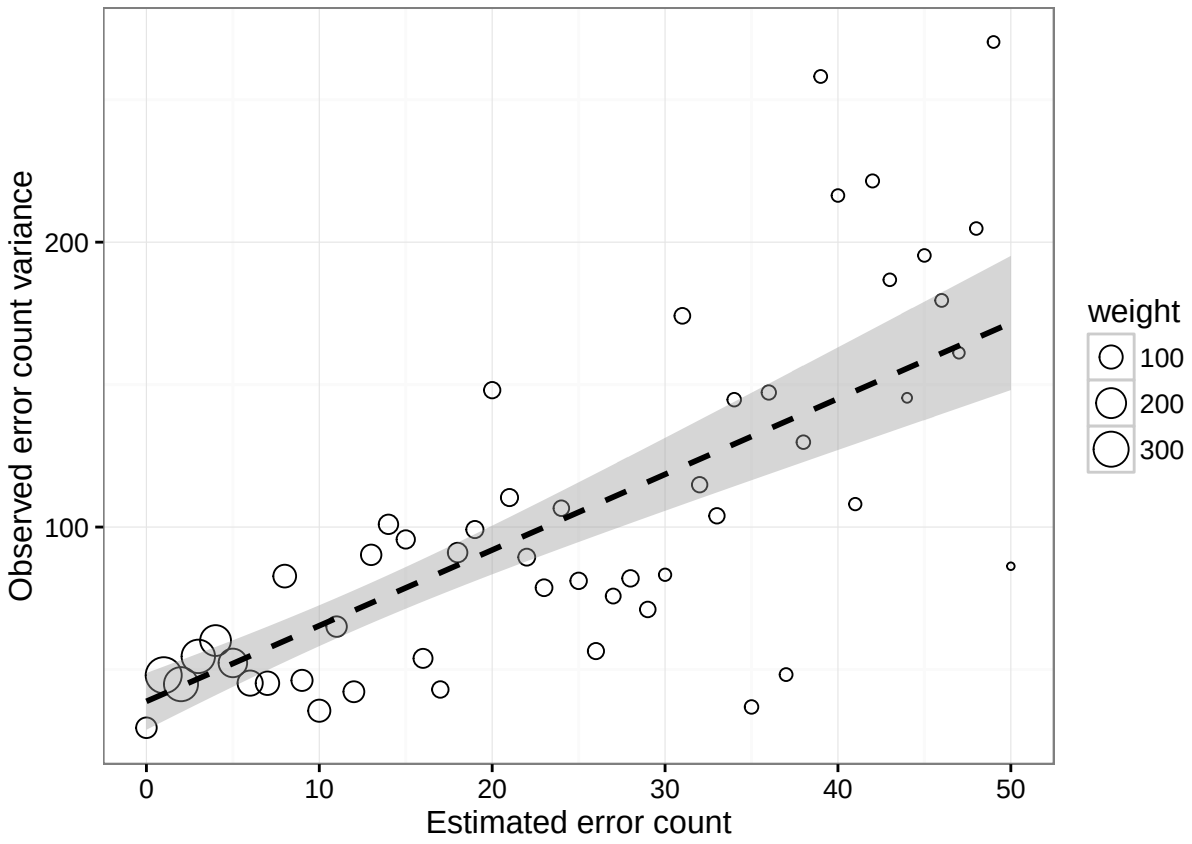
```
df.1.m.0 <- ddply(df.1.m, .(count.est, f.mean, f.var),
                  summarize,
                  weight = sum(weight))

ggplot(df.1.m.0, aes(x=count.est, y=f.mean)) + geom_point(aes(size=weight), shape=21) +
  geom_smooth(aes(weight=weight), method=lm, color="black", linetype="dashed") +
  xlab("Estimated error count") +
  ylab("Observed error count mean") +
  theme_bw()
```




```
l.mean.fit <- lm(f.mean ~ count.est,
                 df.1.m.0, weights=weight)

print(coef(l.mean.fit))
```

```
## (Intercept)   count.est
##   4.8934221   0.3119646
```

```
ggplot(df.1.m.0, aes(x=count.est, y=f.var)) + geom_point(aes(size=weight), shape=21) +
  geom_smooth(aes(weight=weight), method=lm, color="black", linetype="dashed") +
  xlab("Estimated error count") +
  ylab("Observed error count variance") +
  theme_bw()
```



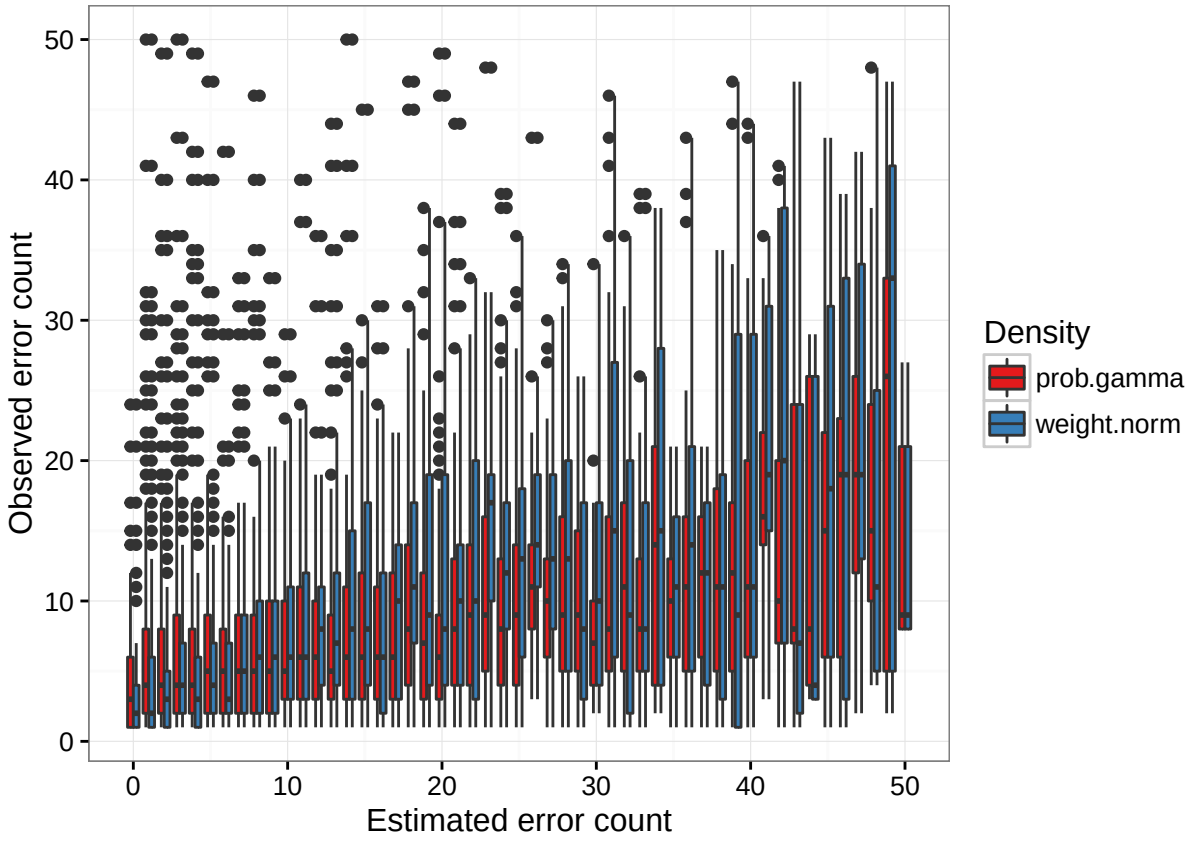

```
l.var.fit <- lm(f.var ~ count.est,
                df.1.m.0, weights=weight)

print(coef(l.var.fit))
```

```
## (Intercept)   count.est
##   38.815039    2.656366
```

Check fit by comparing probabilities computed using Gamma distribution and `P(lambda|MBEM error rate * coverage)`:

```
df.1.m <- ddply(df.1.m, .(count.est),
                transform,
              f.mean.est =
                coef(l.mean.fit)["(Intercept)"]
                + coef(l.mean.fit)["count.est"] * count.est,
              f.var.est =
                coef(l.var.fit)["(Intercept)"]
                + coef(l.var.fit)["count.est"] * count.est)

df.1.m$a <- df.1.m$f.mean.est * df.1.m$f.mean.est / df.1.m$f.var.est
df.1.m$b <- df.1.m$f.mean.est / df.1.m$f.var.est
df.1.m$prob.gamma <- dgamma(df.1.m$count.obs, rate = df.1.m$b, shape = df.1.m$a)

library(reshape2)

df.1.m.1 <- melt(df.1.m, id.vars = c("count.est", "count.obs"),
                 measure.vars = c("prob.gamma", "weight.norm"))
```

```
# requires install.packages("quantreg")
ggplot(df.1.m.1, aes(x=count.est, group = interaction(variable,count.est), fill=variable)) +
  geom_boxplot(aes(y=count.obs, weight=value)) +
  xlab("Estimated error count") +
  ylab("Observed error count") +
  scale_fill_brewer(palette = "Set1", "Density") +
  theme_bw()
```



Lets now compute P-values for errors using fitted Negative Binomial model and compare them to real P-values of error counts:

```
df.1.m$size <- df.1.m$a
df.1.m$prob <- df.1.m$b / (1 + df.1.m$b)

df.1.m$pval <- with(df.1.m,
                    1.0 - pnbinom(count.obs, prob = prob, size = size) +
                      0.5 * dnbinom(count.obs, prob = prob, size = size))

df.1.m$pval.true <- with(df.1.m, sapply(count.obs,
                           function(x) sum(ifelse(x > count.obs, 0, ifelse(x == count.obs, 0.5 * weight
                         / sum(weight))

ggplot(df.1.m, aes(x=-10*log10(pval), y=-10*log10(pval.true))) +
  geom_point(shape = 21) +
  stat_density2d(aes(fill=..level.., weight=weight), geom="polygon") +
  geom_abline(intercept = 0, slope = 1, color = "black", linetype="dashed") +
  scale_x_continuous("Computed Q score", limits=c(0,40)) +
```

```
  scale_y_continuous("True Q score", limits=c(0,40)) +
  scale_fill_gradientn(colors=r) +
  theme_bw()
```