Stickbreaker Vignette

Craig Miller and James Van Leuven

Mon Jan 2 12:44:22 2017

- 2. Simulate training data matching this dataset
- 3. Fit training data to multinomial model
- 4. Calculate posteriors on Burns data

I. Introduction

Genetically modified organisms are used to test the phenotypic effects of mutation. The way mutations interact to affect fitness or other phenotypic traits (i.e. epistasis) is of fundamental importance in evolutionary biology. This package fits three basic models (additive, multiplicative, and stickbreaking) to fitness data and suggests the best fitting model by multinomial regression. Stickbreaker can also be used to simulate fitness data.

II. Basic Analysis

A. Data format

The data should be in a text file. If there are n mutations in the dataset, there should be $n+1$ columns in the file. The first row (header row) should be mutation names. The data in the first n columns are 0s and 1s that indicate the absence (0) or presence (1) of the mutation. All 0s indicates wild type; this needs to be the first row of data in the file. The last column is the estimated fitness of that genotype and should be titled `Fitness'. Columns to the right of these are permitted and are not used in the analysis.

B. Read in data

```
inpath <- system.file("data", package="Stickbreaker")
data <- read.csv(paste(inpath,"Chou.data.csv",sep="/"))
```

C. Fit models

The `fit.models()` function is a wrapper that fits the data to all three models. First, it estimates d and, using this estimate, fits the stickbreaking model. Then it fits the multiplicative and additive models. If you want to do this fitting to models individually and better understand what is going on, see the Detailed Features section.

To fit the models, you need to specify a few things. `d.range` specifies the range of possible values for d. If you fitness values are unusually large you may want to change this range. The upper bound is more important. The reason not to set this really large (say to 100 or 1000 when observed fitness values are say in the 1-20 range) is that the stickbreaking model approaches the additive model as d gets large and the coefficients get small. By putting a modest upper bound on stickbreaking, we demand that additivity remains a distinct model.

When the stickbreaking model doesn't fit the data very well or when the data is really noise, the proffered ways to estimate d can fail. Still, we need to fit the stickbreaking model (albeit poorly) for comparison reason. We do this simply by multiplying the fitness distance between the wildtype and the largest observed fitness by a factor, `d.adj.max` . The default is 1.1. By making it only slightly above one, we again ensure that stickbreaking remains distinct from the additive model.

`wts` specifies how to weight single mutations (genotypes with one mutation on the wildtype background) vs all other genotypes when estimating the coefficients. If the wildtype fitness is know which much greater accuracy than the other genotypes as will typically be the case, then the comparisons to it will have lower variance than the other comparisons. The default weighting is `c(2,1)` which indicates to weight the single mutations twice as heavily as all others which corresponds to our assumption that the wildtype fitness is know without error while all other genotypes have the same error.

```
fit.results <- fit.models(data, d.range=c(0.1, 10), d.adj.max=1.1, wts=c(2,1))
```

`fit.models()` returns a list of results (assigned here to `fit.results`). `fit.results$fit.smry` is the most important item. This is a vector of R^2 and P-scores–one each from the three models–that summarize model fits. These summary statistics will be used to determine the posterior probability of each model in the next code block. See paper for details about how these fit statistics are calculated.

The other items in the list returned from `fit.models()` are details of fit to each model: `$fit.stick`, `$fit.mult` and `$fit.add`. Each of these contains analogous information, itself organized as a list. [[1]] The coefficients (`$u.hats` for stickbreaking, `$s.hats` for multiplicative, and `$w.hats` for additive). [[2]] `$R2`. [[3]] `$sig.hat` is estimate of σ. [[4]] `$log.like` is the log-likelihood of the data. [[5]] `$regression.results` is itself another list with `$p.vals`, `lm.intercepts` and `lm.slopes` being the linear regression p-values, intercepts, and slopes for each mutation and `P` being the P-score. The last two items of `$regression.results` are matrices `$fitness.of.backs` and `$effects.matrix`. These are the data being regressed. `$fitness.of.backs` gives the fitness of the background genotype when a given mutation (column) is added to create a given genotype (row). `$effects.matrix` gives the fitness effect that this mutation has when added in the background genotype under the model being considered.

D. Generate multinomial and calculate posterior probabilities

After the data has been fit to the three models, we use the summary fit statistics to assign posterior probabilities to the three models. This is done in three steps: (1) simulate data from priors, (2) fit simulated data to multinomial regression and (3) calculate posterior probabilities.

Priors etc.: To simulate data you need to define priors and the number of replicate datasets to simulate. These are defined at the beginning of the next code block. `coes.prior` is a vector with the lower and upper bounds on the coefficients; `sig.prior` is the lower and upper bound on the σ. Priors are assumed to be uniformly distributed between these boundaries. `n.samps.per.mod` is the number of datasets to simulate per model. You may want to initially set this to a small value like 50 or 100 to check things are running and behaving properly. For a final analysis you should increase this number to 1000 or, better yet, 10000. Be warned that it will take some time to simulate 10000 datasets for each model. `d.range`, `d.adj.max`, and `wts` were discussed above. Finally, use `analysis.name` to assign a unique name so that the simulated fitted data and the multinomial model it leads to are saved with unique identifiers. These files may be of interest (and save you computing time) if you choose to tap into any of the detailed features detailed below. Simulated data and models fit to it are saved in the `inst' folder under` extdata/'and `models/' subfolders respectively.

One input that warrants discussion is `min.R2`. As discussed in the paper, when a model fits a dataset very poorly, it can generate R^2 values < 1. Occasionally, very negative values are produces (e.g. -100). These values create problems for the multinomial regression because the vast majority of the predictive R^2 values are in the 0 to 1 interval. Preliminary analysis revealed the method is improved by replace large negative R^2 values with a modest negative value. `min.R2` defines what this modest negative value is. -1 is suggested and used throughout this work.

Finally, notice the argument `print.interval`. When doing simulations, a replicate counter will be printed to the console at the interval defined here. If set to `NA`, nothing is printed.

```
  coes.prior <- c(0.05, 0.5)
  sig.prior <- c(0, 0.25)
  n.samps.per.mod <- 1000   # set to smaller value when setting code up; larger value when doing real analysis
  d.range <- c(0.1, 10)
  d.adj.max <- 1.1
  wts <- c(2,1)
  min.R2 <- -1
  analysis.name <- "Test"

  posterior.results <- sim.data.calculate.posteriors(fit.results$fit.smry, data, analysis.name, coes.prior, sig.prior, d.range, d.adj.max, wts, n.samps.per.mod, min.R2, print.interval=25)
```

```
## [1] "stick"
## [1] 1
## [1] 25
## [1] 50
## [1] 75
## [1] 100
## [1] 125
## [1] 150
## [1] 175
## [1] 200
## [1] 225
## [1] 250
## [1] 275
## [1] 300
## [1] 325
## [1] 350
## [1] 375
## [1] 400
## [1] 425
## [1] 450
## [1] 475
## [1] 500
## [1] 525
## [1] 550
## [1] 575
## [1] 600
## [1] 625
## [1] 650
## [1] 675
## [1] 700
## [1] 725
## [1] 750
## [1] 775
## [1] 800
## [1] 825
## [1] 850
## [1] 875
## [1] 900
## [1] 925
## [1] 950
## [1] 975
## [1] 1000
## [1] "mult"
## [1] 1
## [1] 25
## [1] 50
## [1] 75
## [1] 100
## [1] 125
## [1] 150
## [1] 175
## [1] 200
## [1] 225
## [1] 250
## [1] 275
## [1] 300
## [1] 325
## [1] 350
## [1] 375
## [1] 400
## [1] 425
## [1] 450
## [1] 475
## [1] 500
## [1] 525
## [1] 550
## [1] 575
## [1] 600
## [1] 625
## [1] 650
## [1] 675
## [1] 700
## [1] 725
## [1] 750
## [1] 775
## [1] 800
## [1] 825
## [1] 850
## [1] 875
## [1] 900
## [1] 925
## [1] 950
## [1] 975
## [1] 1000
## [1] "add"
## [1] 1
## [1] 25
## [1] 50
## [1] 75
## [1] 100
## [1] 125
## [1] 150
## [1] 175
## [1] 200
## [1] 225
## [1] 250
## [1] 275
## [1] 300
## [1] 325
## [1] 350
## [1] 375
## [1] 400
## [1] 425
## [1] 450
## [1] 475
## [1] 500
## [1] 525
## [1] 550
## [1] 575
## [1] 600
## [1] 625
## [1] 650
## [1] 675
## [1] 700
## [1] 725
## [1] 750
## [1] 775
## [1] 800
## [1] 825
## [1] 850
## [1] 875
## [1] 900
## [1] 925
## [1] 950
## [1] 975
## [1] 1000
## [1] "Fitting multinomial model to simulated data"
## # weights:  24 (14 variable)
## initial  value 3295.836866
## iter  10 value 1909.147525
## iter  20 value 1677.184060
## iter  30 value 1675.561302
## iter  30 value 1675.561299
## final  value 1675.561299
## converged
```

III. Detailed Features

A. Data and Setup

Whereas above we used wrapper functions to do the entire analysis is just a few commands, here we walk through the various functions in greater detail.

Read in data. See above (II. Basic Analysis) for format of data.

```
inpath <- system.file("data", package="Stickbreaker")
data <- read.csv(paste(inpath,"Khan.data.csv",sep="/"))
```

Define parameters to use in estimation (`wts` , `d.range` and `d.adj.max`). Again, these are discussed above. Then extract number of genotypes (`n.genos`), number of mutations (`n.muts`), the genotype matrix (`geno.matrix` ; the 0/1 columns in the data) and the fitness matrix (`fit.matrix` ; the last column in the data).

```
wts <- c(2,1)
d.range <- c(0.1, 10)
d.adj.max <- 1.1

n.genos <- length(data[,1])
n.muts <- length(data[1,])-1
geno.matrix <- data[,seq(1, n.muts)]
fit.matrix <- as.matrix(data[,(n.muts+1)])
```

B. Model fitting

1. Fit to stickbreaking model

We have two good estimators of the distance to the fitness boundary, d (see paper for details). We call them using the functions `d.hat.MLE()` and `d.hat.RDB()` . We take the MLE as the estimate unless it fails; in which case we use the RDB estimate. If both fail, we simply use the distance from wild type to the largest observed fitness value adjusted upwards by a factor `d.adj.max` . This decision rule is implement in the function `d.hat.seq()` . Why adjust the maximum estimate upwards at all? If we do not, the genotype with maximum fitness is at the boundary which is problematic since it results in a zero in the denominator when estimating d for this genotype. But by adjusting it up only slightly (e.g. 1.1), we ensure that stickbreaking remains distinct from the additive model (as discussed above).

```
d.hat.MLE <- estimate.d.MLE(geno.matrix, fit.matrix, d.range=d.range)
d.hat.RDB <- estimate.d.RDB(geno.matrix, fit.matrix)$d.hat.RDB
d.hat.seq <- estimate.d.sequential(geno.matrix, fit.matrix,
                                   d.hat.MLE, d.hat.RDB, d.range, d.adj.max)
d.hat.MLE
```

```
## [1] 0.4189998
```

```
d.hat.RDB
```

```
## [1] 0.6396245
```

```
d.hat.seq
```

```
##       MLE
## 0.4189998
```

After getting an estimate for d, we call `fit.stick.model.given.d()` . This returns a list the with the stickbreaking coefficients (`$u.hats`), R^2 (`$R2`), $\hat{\sigma}$ (`$sig.hat`) and the log-likelihood (`$log.like`).

The function also performs a linear regression of background fitness against effect. When data are generated and analyzed under the same model, the slope of this regression line is expected to be zero for each mutation with p-values distributed U(0,1). When the data are generated and analyzed under untrue models, the regression line is expected to have non-zero slope and lower p-values. One regression is performed per mutation and we calculate a p-value. We then summarize the data across mutations by taking the sum of the logs of the p-values and call this sum the P-score. The more negative the P-score, the stronger the evidence against a model being correct. The regression results are returned under the list item `$regression.results` with the sub-items giving p-values (`$p.vals`), intercepts (`$lm.intercepts`), and slopes (`$lm.slopes`) and the overall P-score (`$P`). The last two items of `$regression.results` are matrices `$fitness.of.backs` and `$effects.matrix` . These are the data being regressed. `$fitness.of.backs` gives the fitness of the background genotype when a given mutation (column) is added to create a given genotype (row). `$effects.matrix` gives the fitness effect that this mutation has when added in the background genotype under the model considered.

The last item in the list returned by `fit.stick.model.given.d()` is a matrix, called `$pred.matrix` , that contains model predicted fitness for each genotypes (appended to the data itself along with error and a binary string that can be useful in producing plots).

```
  fit.stick <- fit.stick.model.given.d(geno.matrix, fit.matrix, d.hat.seq, run.regressio
n=TRUE)
  fit.stick
```

```
## $u.hats
##      mut.r      mut.t      mut.s      mut.g 
## 0.06243185 0.42521631 0.42245493 0.14903233 
## 
## $R2
## [1] 0.9444234
## 
## $sig.hat
## [1] 0.0191714
## 
## $logLike
## [1] 41.56635
## 
## $regression.results
## $regression.results$p.vals
## [1] 0.9425579 0.8617554 0.8435559 0.8036109
## 
## $regression.results$lm.intercepts
## [1] 0.09890953 0.31185146 0.30620618 0.28674178
## 
## $regression.results$lm.slopes
## [1] -0.03486582  0.08965631  0.09413245 -0.12826224
## 
## $regression.results$P
## [1] -0.5967109
## 
## $regression.results$fitness.of.backs
##        [,1]  [,2]  [,3]  [,4]
##  [1,]    NA    NA    NA    NA
##  [2,] 1.003    NA    NA    NA
##  [3,]    NA 1.003    NA    NA
##  [4,]    NA    NA 1.003    NA
##  [5,]    NA    NA    NA 1.003
##  [6,] 1.196 1.025    NA    NA
##  [7,] 1.188    NA 1.025    NA
##  [8,] 1.078    NA    NA 1.025
##  [9,]    NA 1.188 1.196    NA
## [10,]    NA 1.078    NA 1.196
## [11,]    NA    NA 1.078 1.188
## [12,] 1.283 1.207 1.192    NA
## [13,] 1.219 1.129    NA 1.192
## [14,] 1.189    NA 1.129 1.207
## [15,]    NA 1.189 1.219 1.283
## [16,] 1.303 1.215 1.193 1.280
## 
## $regression.results$effects.matrix
##              [,1]      [,2]      [,3]        [,4]
##  [1,]          NA        NA        NA          NA
##  [2,]  0.05250600        NA        NA          NA
##  [3,]          NA 0.4606208        NA          NA
##  [4,]          NA        NA 0.4415277          NA
##  [5,]          NA        NA        NA 0.178997718
##  [6,] -0.01769913 0.4206552        NA          NA
##  [7,]  0.08119667        NA 0.4584386          NA
##  [8,]  0.14825592        NA        NA 0.261964897
##  [9,]          NA 0.4059833 0.3849562          NA
## [10,]          NA 0.4098840        NA 0.101770022
## [11,]          NA        NA 0.3226746 0.004273509
## [12,] -0.02158277 0.3395353 0.3826091          NA
## [13,] -0.12807897 0.2184302        NA 0.004347831
## [14,]  0.11158810        NA 0.2935156 0.037209345
## [15,]          NA 0.4892709 0.4137936 0.143885145
## [16,]  0.23529460 0.5603871 0.6026207 0.359155548
##
##
## $pred.matrix
##    mut.r mut.t mut.s mut.g string   fit     pred         error
## 1      0     0     0     0   0000 1.003 1.003000  0.000000e+00
## 2      1     0     0     0   1000 1.025 1.029159 -4.158929e-03
## 3      0     1     0     0   0100 1.196 1.181166  1.483447e-02
## 4      0     0     1     0   0010 1.188 1.180009  7.991489e-03
## 5      0     0     0     1   0001 1.078 1.065445  1.255549e-02
## 6      1     1     0     0   1100 1.192 1.196201 -4.201255e-03
## 7      1     0     1     0   1010 1.207 1.195116  1.188353e-02
## 8      1     0     0     1   1001 1.129 1.087705  4.129509e-02
## 9      0     1     1     0   0110 1.283 1.282907  9.286533e-05
## 10     0     1     0     1   0101 1.219 1.217058  1.942384e-03
## 11     0     0     1     1   0011 1.189 1.216073 -2.707303e-02
## 12     1     1     1     0   1110 1.280 1.291591 -1.159094e-02
## 13     1     1     0     1   1101 1.193 1.229853 -3.685253e-02
## 14     1     0     1     1   1011 1.215 1.228929 -1.392942e-02
## 15     0     1     1     1   0111 1.303 1.303636 -6.364326e-04
## 16     1     1     1     1   1111 1.331 1.311026  1.997393e-02
```

2. Fit dataset to multiplicative model

Here we fit to the multiplicative model using `fit.mult.model()`. The output is exactly analogous to that returned from `fit.stick.model.given.d()` discussed immediately above except the coefficients are named `$s.hats` (instead of `u.hats`).

```
fit.mult <- fit.mult.model(geno.matrix, fit.matrix, wts=wts)
#fit.mult
```

3. Fit dataset to additive model

And finally we fit the additive model using `fit.add.model()`. Again the outputs are the same except coefficients are `$w.hats`.

```
fit.add <- fit.add.model(geno.matrix, fit.matrix, wts=wts)
#fit.add
```

4. Visualize model fit

Let’s visualize the regression of background fitness vs effect size. This should help you see what is going on under the hood.

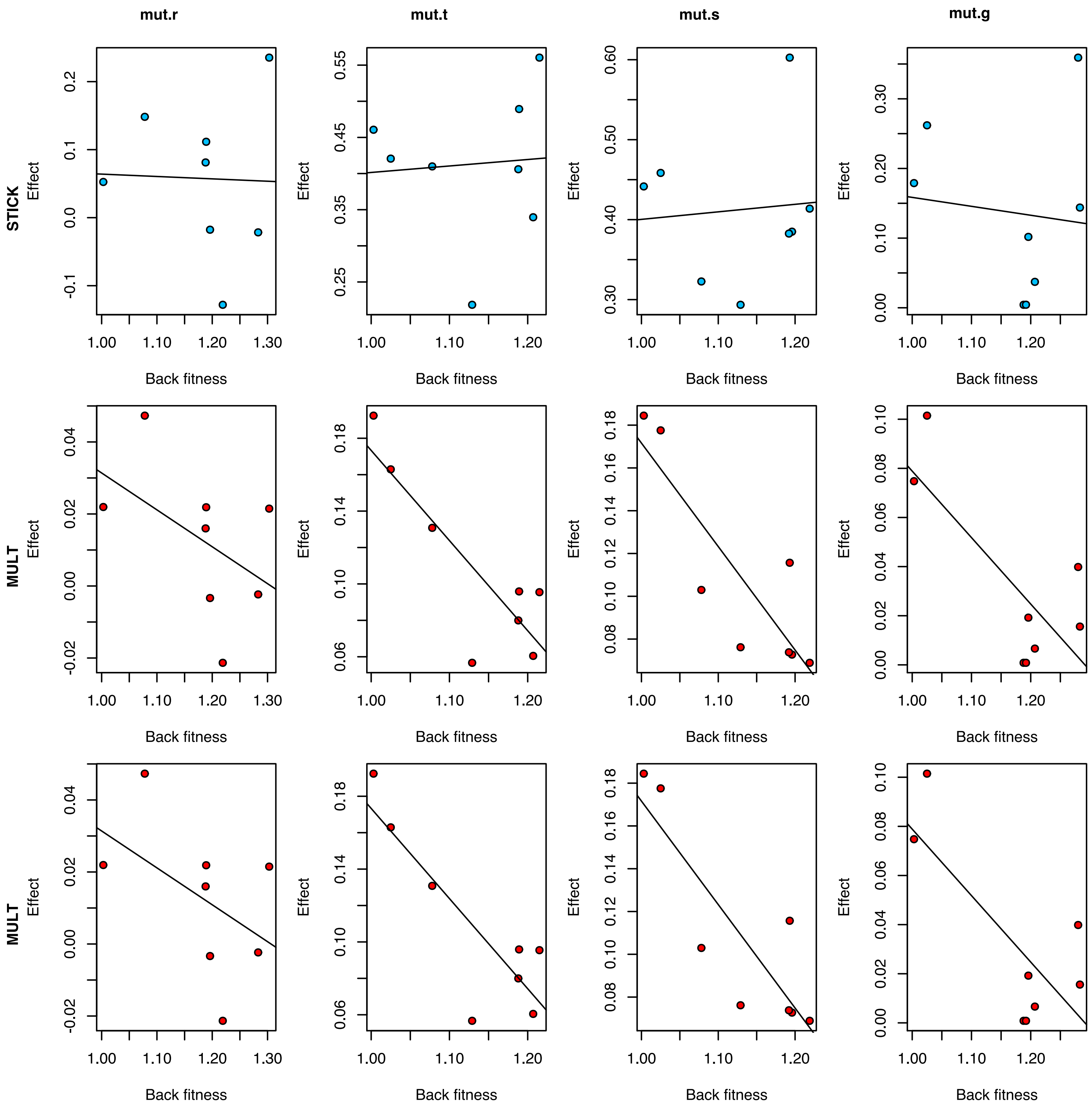
```
  outdir <- system.file("figures", package="Stickbreaker")
  plot.name <- "Ex_fit_vs_effect_plot.svg"
  file.path <- paste(outdir, plot.name, sep="/")
  svg(file=file.path, width=8, height=8)

  layout(mat=matrix(nrow=4, ncol=(1+n.muts), data=seq(1,4*(1+n.muts)), byrow=T),
widths=c(0.25, rep(3,n.muts)), heights=c(0.25, rep(3,3))) -> l
  #layout.show(l)
  par(mar=c(0,0,0,0))
  plot(0,type='n',axes=FALSE,ann=FALSE)
  mod.names <- c("STICK", "MULT", "ADD")
  mut.names <- colnames(data)[1:n.muts]
  mod.cols <- c("deepskyblue","red", "yellow")
  for (i in 1:n.muts){
    plot(0,type='n',axes=FALSE,ann=FALSE, ylim=c(0,1), xlim=c(0,1))
    text(0.5, 0.5, labels=mut.names[i], font=2)
  }
  #--- STICK ---
  par(mar=c(0,0,0,0))
  plot(0,type='n',axes=FALSE,ann=FALSE, ylim=c(0,1), xlim=c(0,1))
  text(0.5, 0.5, labels=mod.names[1], font=2, srt=90)
  par(mar=c(4,4,1,1))
  for (mut.i in 1:n.muts){
    plot(x=fit.stick$regression.results$fitness.of.backs[,mut.i], y=fit.stick$regressio
n.results$effects.matrix[,mut.i], ylab="Effect", xlab="Back fitness", pch=21, bg=mod.col
s[1])
    abline(fit.stick$regression.results$lm.intercepts[mut.i], fit.stick$regression.resul
ts$lm.slopes[mut.i])
  }
  #--- MULT ---
  par(mar=c(0,0,0,0))
  plot(0,type='n',axes=FALSE,ann=FALSE, ylim=c(0,1), xlim=c(0,1))
  text(0.5, 0.5, labels=mod.names[2], font=2, srt=90)
  par(mar=c(4,4,1,1))
  for (mut.i in 1:n.muts){
    plot(x=fit.mult$regression.results$fitness.of.backs[,mut.i], y=fit.mult$regression.r
esults$effects.matrix[,mut.i], ylab="Effect", xlab="Back fitness", pch=21,
bg=mod.cols[2])
    abline(fit.mult$regression.results$lm.intercepts[mut.i],
fit.mult$regression.results$lm.slopes[mut.i])
  }
  #--- ADD ---
  par(mar=c(0,0,0,0))
  plot(0,type='n',axes=FALSE,ann=FALSE, ylim=c(0,1), xlim=c(0,1))
  text(0.5, 0.5, labels=mod.names[2], font=2, srt=90)
  par(mar=c(4,4,1,1))
  for (mut.i in 1:n.muts){
    plot(x=fit.mult$regression.results$fitness.of.backs[,mut.i], y=fit.mult$regression.r
esults$effects.matrix[,mut.i], ylab="Effect", xlab="Back fitness", pch=21,
bg=mod.cols[2])
    abline(fit.mult$regression.results$lm.intercepts[mut.i],
fit.mult$regression.results$lm.slopes[mut.i])
```

```
  }
  dev.off()
```

```
## quartz_off_screen
##                 2
```

Plot the figure just created. You can see why the stickbreaking model is favored. The effect sizes show no clear trend by background fitness under stickbreaking while, for the multiplicative and additive models, there are strong trends.

```
    knitr::include_graphics(file.path)
```



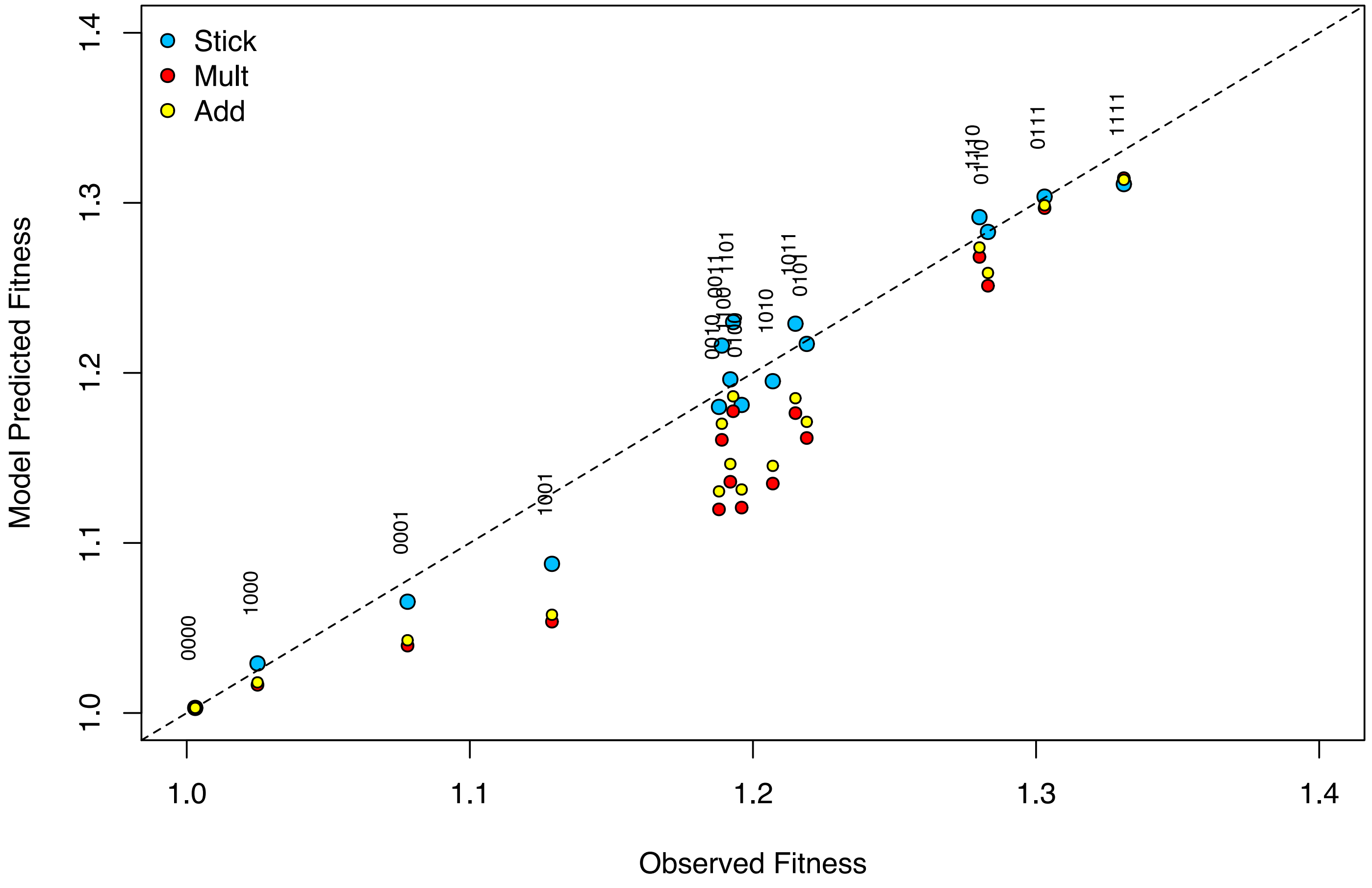
Now, let's look at the observed vs model predicted values. This is the other way we assess the fit of the model.

```
  outdir <- system.file("figures", package="Stickbreaker")
  plot.name <- "Ex_pred_obs_fit.svg"
  file.path <- paste(outdir, plot.name, sep="/")
  svg(file=file.path, width=8, height=6)

  mod.cols <- c("deepskyblue","red", "yellow")
  lims <- c(1,1.4)
  par(mar=c(5,4,4,1))
  plot(x=fit.matrix, y=fit.stick$pred.matrix$pred, ylim=lims, xlim=lims, ylab="Model Pre
dicted Fitness", xlab="Observed Fitness", pch=21, bg=mod.cols[1], cex=1.1)
  abline(0,1, lty="dashed")
  points(x=fit.matrix, y=fit.mult$pred.matrix$pred, pch=21, bg=mod.cols[2], cex=0.9)
  points(x=fit.matrix, y=fit.add$pred.matrix$pred, pch=21, bg=mod.cols[3], cex=0.8)
  text(x=fit.matrix, y=fit.stick$pred.matrix$pred, labels=fit.stick$pred.matrix$string,
srt=90, cex=0.7, pos=3, off=2)
  legend("topleft", legend=c("Stick", "Mult", "Add"), pch=21, pt.bg=mod.cols, bty="n")
  dev.off()
```

```
## quartz_off_screen
##                  2
```

Plot the figure just created.

```
    knitr::include_graphics(file.path)
```



C. Simulation Machinery

The method itself uses simulations to calculate posterior probabilities. We also did a lot of simulation work in the study itself. We show the interested user how to use the existing code to simulate datasets using the sim functions.

1. Single Datasets

a. Stickbreaking

Notice first we set the parameters: number of mutations, stickbreaking coefficients, sigma.

```
n.muts <- 3
coe <- 0.1
coes <- rep(coe, n.muts)
sigma <- 0.1
w.wt <- 1         #fitness of wild type
d.true <- 1       # distance to fitness boundary
n.genos <- 2^n.muts     # number of genotypes in full network
d.range <- c(0.1, 10)
d.adj.max <- 1.1

geno.matrix <- generate.geno.matrix(n=n.muts)
stick.sim.data <- sim.stick.data(n.muts, coes, sigma, d.true, w.wt, geno.matrix)
fit.matrix <- stick.sim.data$fit.matrix
#print(fit.matrix)
```

b. Multiplicative

```
n.muts <- 3
selcoe <- 0.3
selcoes <- rep(selcoe, n.muts)
sigma <- 0.1
w.wt <- 1         #fitness of wild type
n.genos <- 2^n.muts     # number of genotypes in full network

geno.matrix <- generate.geno.matrix(n=n.muts)
mult.sim.data <- sim.mult.data(n.muts, selcoes, sigma, w.wt, geno.matrix)
fit.matrix <- mult.sim.data$fit.matrix
#print(fit.matrix)
```

c. Additive

```
n.muts <- 3
addcoe <- 0.3
addcoes <- rep(addcoe, n.muts)
sigma <- 0.1
w.wt <- 1         #fitness of wild type
n.genos <- 2^n.muts     # number of genotypes in full network

geno.matrix <- generate.geno.matrix(n=n.muts)
add.sim.data <- sim.add.data(n.muts, addcoes, sigma, w.wt, geno.matrix)
fit.matrix <- add.sim.data$fit.matrix
#print(fit.matrix)
```

2. Batch of datasets

Here we show how the code can be used to generate many datasets under a given model and then fit each of them to that same model. This is good for assessing estimation error and bias. Notice in the code block below that parameters are now defined as vectors: `mut.vals` , `coe.vals` , `sig.vals` . The code will simulate under all combinations of the parameter values (i.e. the number of parameter combinations is `length(mut.vals) * length(coe.vals) * length(sig.vals)`). We also define the place to write the data to here. If you have already simulated data and want to skip this step, set `sim.batch.data <- FALSE` at the beginning of the next code block. Also notice we can suppress update printing by setting `print.status` to FALSE. Finally, notice that there is a Boolean argument called `run.regression` . This is used in model selection (next section). Here we are simply showing how to simulate data and, to save computational time, have toggled `run.regression` to FALSE.

a. Stickbreaking

The only added complexity for stickbreaking is the `fit.methods` parameter. This refers to what method of estimating d should be used. If set to `All` it will estimate d using each possible method and output the results individually (i.e. MLE, RDB, max, and seq). Or you can select an individual method (e.g. `seq` . Use `?sim.fit.stick.data.batch` to get details).

```
  sim.batch.data <- TRUE

  mut.vals <- c(3,4,5)
  coe.vals <- c(0.1, 0.3, 0.5)
  sig.vals <- c(0.02, 0.05, 0.08)
  w.wt <- 1
  d.true <- 1
  wts <- c(2,1)
  n.reps.ea <- 10  # set to a small value just to make code run fast
  d.range <- c(0.1, 10)
  fit.methods <- "seq"
  d.max.adj <- 1.0
  run.regression <- FALSE

  if (sim.batch.data == TRUE){
    outfile <- "Stick_batch_out_test"
    outdir <- system.file("extdata", package="Stickbreaker")
    outpath <- paste(outdir, outfile, sep="/")
    sim.fit.stick.data.batch(mut.vals=mut.vals, coe.vals=coe.vals, sig.vals=sig.vals, d.
true=d.true, d.range=d.range, w.wt=w.wt, n.reps.ea=n.reps.ea, print.status=FALSE, fit.me
thods=fit.methods, outpath=outpath, wts, d.max.adj=d.max.adj, run.regression,
RDB.method="pos")
  }
```

b. Multiplicative

```
  sim.batch.data <- TRUE

  mut.vals <- c(3,4,5)
  coe.vals <- c(0.1, 0.3, 0.5)
  sig.vals <- c(0.02, 0.05, 0.08)
  w.wt <- 1
  n.reps.ea <- 10

  #--- mulitiplicative ---
  if (sim.batch.data == TRUE){
    outfile <- "Mult_batch_out_test"
    outdir <- system.file("extdata", package="Stickbreaker")
    outpath <- paste(outdir, outfile, sep="/")
    sim.fit.mult.add.data.batch(epi.model="mult", mut.vals=mut.vals, coe.vals=coe.vals,
sig.vals=sig.vals, w.wt=w.wt, n.reps.ea=n.reps.ea, print.status=FALSE, outpath=outpath,
wts)
  }
```

c. Additive

```
  sim.batch.data <- TRUE

  mut.vals <- c(3,4,5)
  coe.vals <- c(0.1, 0.3, 0.5)
  sig.vals <- c(0.02, 0.05, 0.08)
  w.wt <- 1
  n.reps.ea <- 10

  if (sim.batch.data == TRUE){
    outfile <- "Add_batch_out_test"
    outdir <- system.file("extdata", package="Stickbreaker")
    outpath <- paste(outdir, outfile, sep="/")
    sim.fit.mult.add.data.batch(epi.model="add", mut.vals=mut.vals, coe.vals=coe.vals, s
ig.vals=sig.vals, w.wt=w.wt, n.reps.ea=n.reps.ea, print.status=FALSE, outpath=outpath, w
ts)
  }
```

D. Model Selection

Now we concern ourselves with the process of assigning posterior probability to each model.

1. Full network data

We begin by focusing on training to full network data (i.e. when all `2^n.muts` genotypes are present). Below we show how to do model selection with incomplete networks.

a. Simulate training data from priors

To perform model selection, our Bayesian approach is based on simulating data under each model and under a range of potential parameter values (i.e. priors). Each simulated dataset is fit to each of the three models. We simulate training data by drawing parameter values from uniform priors. Toggle to FALSE if you want to skip this or if you already have the data simulated. The parameter `n.samps.per.mod` defines how many datasets to simulate per model. Once you have the code running smoothly and you want to generate a training dataset to be used in further analysis, set this to large number (e.g. 10000).

Priors etc.: `coes.prior` and `sig.prior` specify the prior range of values to simulate coefficients and sigma values from. Be sure these are reasonable ranges for the data you are dealing with. Also see II. Basic Analysis section above about priors and other definitions in this code block such as `d.range`, `d.adj.max`, and `wts`. `muts.to.sim` is a vector that defines the different number of mutations you want to simulate training datasets for.

```
  sim.training.data.from.priors <- TRUE

  if (sim.training.data.from.priors == TRUE){
    coes.prior <- c(0.05, 0.5)
    sig.prior <- c(0, 0.25)
    n.samps.per.mod <- 1000
    d.true <- 1
    d.range <- c(0.1, 10)
    d.adj.max <- 1.1
    w.wt <- 1
    wts <- c(2,1)
    muts.to.sim <- c(4)   # vector; to simulate, for example, sets of 3, 4 and 5 mutatio
ns at once, use c(3,4,5)
    print.interval <- NA  # set to NA to block update printing

    for (mut.i in 1:length(muts.to.sim)){
      n.muts <- muts.to.sim[mut.i]
      print(paste("n.muts=", n.muts))
      outdir <- system.file("extdata", package="Stickbreaker")
      file.name <- paste("Training_simulated_priors_fit_data_", n.muts, "muts.txt",
sep="")
      outpath <- paste(outdir, file.name, sep="/")
      sim.data.from.priors.for.mod.selection(n.muts=n.muts, coes.prior=coes.prior, sigs.
prior=sig.prior, mods.to.sim=c("stick", "mult", "add"), d.true=d.true, d.range=d.range,
d.adj.max=d.adj.max, w.wt=w.wt, wts=wts, outpath=outpath, n.samps.per.mod=n.samps.per.mo
d, coe.sim.model="identical", coe.dist.par=NA, print.interval=print.interval)
    } #next mut.i
  }
```

```
## [1] "n.muts= 4"
```

b. Fit training data to multinomial model for model selection

Here we pass the training datasets to the `multinom()` function in the `nnet` package. It fits the data to a multinomial regression model and calculates the posterior probability of each model. We do this separately for each number of mutations. `fit.nnet.multinomial.regression()` is the function that handles this. The fitted model is then saved in the directory `inst/models` using the naming convention `nnet_mod_R2_P_nmuts.rda` where `nmuts` is the number of mutations in the data. If you want to skip the model fitting because it's already been done, set `fit.multinomial <- FALSE`.

Also note the defining of `min.R2`. See above for discussion of this input.

```
  model.file <- "nnet_mod_R2_P"
  moddir <- system.file("models", package="Stickbreaker")
  modpath <- paste(moddir, model.file, sep="/")
  data.file <- "Training_simulated_priors_fit_data"

  fit.multinomial <- TRUE
  min.R2 <- -1

  if (fit.multinomial == TRUE){

    mod.formula <- as.formula(model ~ R2.stick + R2.mult + R2.add + P.stick + P.mult +
P.add)
    muts.to.sim <- c(4)
    fit.nmet.models <- vector("list", length(muts.to.sim))
    for (mut.i in 1:length(muts.to.sim)){
      n.muts <- muts.to.sim[mut.i]
      indir <- system.file("extdata", package="Stickbreaker")
      file.name <- paste(data.file, "_", n.muts, "muts.txt", sep="")
      inpath <- paste(indir, file.name, sep="/")
      if (file.exists(inpath)){
        data <- read.table(file=inpath, header=TRUE)
        data$R2.stick[which(data$R2.stick < min.R2)] <- min.R2
        data$R2.mult[which(data$R2.mult < min.R2)] <- min.R2
        data$R2.add[which(data$R2.add < min.R2)] <- min.R2
        fit.nmet.models[[mut.i]] <- fit.nnet.multinomial.regression(data, mod.formula)
        m <- fit.nnet.multinomial.regression(data, mod.formula)
        modelout <- paste(modpath, "_", n.muts, "muts.rda", sep="")
        saveRDS(m, file=modelout)
      }
    }
  }
```

2. Incomplete networks

This is very similar to the full networks, except that we supply a particular dataset structure (in the form of `geno.matrix`) and the simulations are tailored to it.

a. Generate a partial dataset to analyze

To have a dataset to work with, let's generate one. Suppose we have five mutations and engineer only the first and last steps (i.e. wildtype plus each individual mutation and the 5-mutant genotypes with each individual mutation removed.) This is done in the code block below. We also define coefficients and sigma values. Then we simulate a set of fitness values under the stickbreaking model using `sim.stick.data()`.

```
  n.muts <- 5
  coes <- 0.2
  sigma <- 0.03
  geno.matrix <- generate.geno.matrix(5)
  geno.matrix <- geno.matrix[which(rowSums(geno.matrix) %in% c(0,1,4,5)),]
  fit.matrix <- sim.stick.data(n.muts, coes=coes, sigma=sigma, d.true=1, w.wt=1, geno.ma
trix)
```

b. Simulate training data with same structure from priors

Just as with full data, we now need to simulate data from the priors for training the multinomial model. However, we want every dataset to have the same structure as the data we just cooked up (i.e. first and last steps only). See the above discussion under *b. Fit training data to multinomial model for model selection* regarding other inputs.

```
  outdir <- system.file("extdata", package="Stickbreaker")
  file.name <- paste("Training_simulated_priors_fit_data_first_last", n.muts,
"muts.txt", sep="")
  simdata.outpath <- paste(outdir, file.name, sep="/")
  coes.prior <- c(0.05, 0.5)
  sig.prior <- c(0, 0.25)
  n.samps.per.mod <- 10   # Increase to a large number when analyzing real data
  d.true <- 1
  d.range <- c(0.1, 10)
  d.adj.max <- 1.1
  w.wt <- 1
  wts <- c(2,1)
  n.samps.per.mod <- 1000

  sim.partial.data.from.priors.for.mod.selection(geno.matrix,
                                                     coes.prior=coes.prior,
                                                     sigs.prior=sig.prior,
                                                     mods.to.sim=c("stick", "mult", "ad
d"),
                                                     d.true=1,
                                                     d.range=d.range,
                                                     d.adj.max=d.adj.max,
                                                     w.wt=1,
                                                     wts=wts,
                                                     outpath=simdata.outpath,
                                                     n.samps.per.mod=n.samps.per.mod,
                                                     coe.sim.model="identical",
                                                     coe.dist.par=NA,
                                                     print.interval=25)
```

```
## [1] "stick"
## [1] 1
## [1] 25
## [1] 50
## [1] 75
## [1] 100
## [1] 125
## [1] 150
## [1] 175
## [1] 200
## [1] 225
## [1] 250
## [1] 275
## [1] 300
## [1] 325
## [1] 350
## [1] 375
## [1] 400
## [1] 425
## [1] 450
## [1] 475
## [1] 500
## [1] 525
## [1] 550
## [1] 575
## [1] 600
## [1] 625
## [1] 650
## [1] 675
## [1] 700
## [1] 725
## [1] 750
## [1] 775
## [1] 800
## [1] 825
## [1] 850
## [1] 875
## [1] 900
## [1] 925
## [1] 950
## [1] 975
## [1] 1000
## [1] "mult"
## [1] 1
## [1] 25
## [1] 50
## [1] 75
## [1] 100
## [1] 125
## [1] 150
## [1] 175
## [1] 200
## [1] 225
## [1] 250
## [1] 275
## [1] 300
## [1] 325
## [1] 350
## [1] 375
## [1] 400
## [1] 425
## [1] 450
## [1] 475
## [1] 500
## [1] 525
## [1] 550
## [1] 575
## [1] 600
## [1] 625
## [1] 650
## [1] 675
## [1] 700
## [1] 725
## [1] 750
## [1] 775
## [1] 800
## [1] 825
## [1] 850
## [1] 875
## [1] 900
## [1] 925
## [1] 950
## [1] 975
## [1] 1000
## [1] "add"
## [1] 1
## [1] 25
## [1] 50
## [1] 75
## [1] 100
## [1] 125
## [1] 150
## [1] 175
## [1] 200
## [1] 225
## [1] 250
## [1] 275
## [1] 300
## [1] 325
## [1] 350
## [1] 375
## [1] 400
## [1] 425
## [1] 450
## [1] 475
## [1] 500
## [1] 525
## [1] 550
## [1] 575
## [1] 600
## [1] 625
## [1] 650
## [1] 675
## [1] 700
## [1] 725
## [1] 750
## [1] 775
## [1] 800
## [1] 825
## [1] 850
## [1] 875
## [1] 900
## [1] 925
## [1] 950
## [1] 975
## [1] 1000
```

c. Fit training data to multinomial model for model selection

Everything here is exactly the same as with full sets of data.

```
  model.file <- "nnet_mod_R2_P_5muts_first_last"
  moddir <- system.file("models", package="Stickbreaker")
  modpath <- paste(moddir, model.file, sep="/")

  fit.multinomial <- TRUE
  min.R2 <- -1

  if (fit.multinomial == TRUE){
    mod.formula <- as.formula(model ~ R2.stick + R2.mult + R2.add + P.stick + P.mult +
P.add)
    indir <- system.file("extdata", package="Stickbreaker")
    file.name <- paste("Training_simulated_priors_fit_data_first_last", n.muts, "muts.tx
t", sep="")
    inpath <- paste(indir, file.name, sep="/")
      if (file.exists(inpath)){
        data <- read.table(file=inpath, header=TRUE)
        data$R2.stick[which(data$R2.stick < min.R2)] <- min.R2
        data$R2.mult[which(data$R2.mult < min.R2)] <- min.R2
        data$R2.add[which(data$R2.add < min.R2)] <- min.R2
        m <- fit.nnet.multinomial.regression(data, mod.formula)
        modelout <- paste(modpath, "_", n.muts, "muts.rda", sep="")
        saveRDS(m, file=modelout)
      }
    }
```

E. Assessing Model Selection

Here we simulate test datasets and ask: how well does the model selection process covers in the last section perform? This was an important part of our paper; we include the relevant code blocks here in the event that the functions are useful to you.

1. Simulate test datasets

In the next code block we simulate datasets at specified parameter values. More specifically, it is necessary to specify the vector of models to simulate (`mods.to.sim`), a vector with the number of mutations to simulate (`muts.to.sim`), a vector with the coefficients to simulate (`coes.to.sim`) and a vector with the sigma values to simulate (`sigs.to.sim`). The code simulates `n.reps.ea` under all possible parameter value combinations. As elsewhere, `d.true`, `d.range`, and `wts` must be specified. Toggle `sim.testing.data.from.vectors` to FALSE if you want to skip this or if you already have the data simulated. Like all other simulated data, this is written to `inst/extdata`. Notice that here we are not drawing parameter values from priors, but rather simulating at specified values. The function that simulates at specified values is `sim.data.for.mod.selection()`.

```r
  sim.testing.data.from.vectors <- TRUE
  testdata.file <- "Test_simulated_fit_data"


  if (sim.testing.data.from.vectors == TRUE){
    mods.to.sim <- c("stick", "mult", "add")   # only the strings "stick", "mult" and "a
dd" are allowed
    muts.to.sim <- c(4)   # c(3,4,5)
    coes.to.sim <- c(0.1, 0.3)   # c(0.1, 0.2, 0.3, 0.4, 0.5)
    sigs.to.sim <- c(0.03, 0.06)
    d.true <- 1
    d.range <- c(0.1, 10)
    w.wt <- 1
    wts <- c(2,1)
    n.reps.ea <- 100
    for (mut.i in 1:length(muts.to.sim)){
      n.muts <- muts.to.sim[mut.i]
      print(paste("n.muts=", n.muts))
      outdir <- system.file("extdata", package="Stickbreaker")
      file.name <- paste(testdata.file, "_", n.muts, "muts.txt", sep="")
      outpath <- paste(outdir, file.name, sep="/")
      sim.data.for.mod.selection(n.muts=n.muts, coes.to.sim, sigs.to.sim, mods.to.sim,
d.true, d.range, w.wt,wts,  outpath, n.reps.ea=n.reps.ea)
    }
  }
```

```
## [1] "n.muts= 4"
```

2. Calculate posterior probabilities on each testing dataset

To do this, we need to tell it where the model was saved and where the testing data was saved. Note this code block can get turned off by setting `fit.test.data` to FALSE. The multinomial model we employ here to assign model selection was generated above in the section *b. Fit training data to multinomial model for model selection*.

Also notice that the resulting posteriors are output to the ‘inst/extdata’ folder and given a name that is the same as the source data file with `n.muts` and then `muts_wPosts.csv` appended.

```
fit.test.data <- TRUE

model.file <- "nnet_mod_R2_P"
moddir <- system.file("models", package="Stickbreaker")
modpath <- paste(moddir, model.file, sep="/")
testdata.file <- "Test_simulated_fit_data"

muts.to.lyze <- c(4)   # The mutations to analyze


if (fit.test.data == TRUE){

  for (mut.i in 1:length(muts.to.lyze)){
    n.muts <- muts.to.lyze[mut.i]
    print(paste("muts =", n.muts))
    modelin <- paste(modpath, "_", n.muts, "muts.rda", sep="")
    mod <- readRDS(modelin)

    datadir <- system.file("extdata", package="Stickbreaker")
    file.name <- paste(datadir, "/", testdata.file, "_", n.muts, "muts.txt", sep="")
    test.data <- read.table(file.name, header=TRUE)

    posteriors <- as.data.frame(matrix(nrow=length(test.data[,1]), ncol=3))
    colnames(posteriors)=c("add", "mult", "stick")

    for (j in 1:length(test.data[,1])){
      posteriors[j,] <- predict(mod, newdata=test.data[j,], type="probs")
    }

    test.data <- cbind(test.data, posteriors)
    outdata.file <- file.name <- paste(datadir, "/", testdata.file, "_", n.muts, "muts
_wPosts.csv", sep="")
    write.csv(x=test.data, file=file.name, row.names=FALSE)
  }  #next mut.i
}  # end if fit.test.data==TRUE
```

```
## [1] "muts = 4"
```

3. Read in posteriors and assess performance

Note that the names of files being read in here are specified in the previous code chunk. This chunk creates `posterior.list` , the top level corresponding to the number of mutations in the `muts.to.lyze` (i.e. mutations to analyze). For each number of mutations, the `posterior.list` contains a dataframe summarizing the frequency of false rejections (`false.rej` is the frequency that the true model is rejected), `true.unq` is the frequency that the true model is uniquely identified as correct, and `mean.post` is the mean posterior probability of the true model. Note that `rej.cut` must be specified to do these calculations. It is the posterior probability below which a model is considered rejected.

```
  muts.to.lyze <- c(4)
  rej.cut <- 0.05       # consider model rejected when it has posterior < this
  posterior.list <- vector("list", length(muts.to.lyze))
  names(posterior.list) <- paste("muts.",  muts.to.lyze, sep="")

  for (mut.i in 1:length(muts.to.lyze)){
     n.muts <- muts.to.lyze[mut.i]
     datadir <- system.file("extdata", package="Stickbreaker")
     indata.file <-  paste(datadir, "/", testdata.file, "_", n.muts, "muts_wPosts.csv",
sep="")
     data <- read.csv(file=indata.file, stringsAsFactors = FALSE)

     posterior.list[[mut.i]] <- summarize.posteriors.on.simulated.dataset(data, rej.cut)
  }
posterior.list[[1]]   # print to illustrate output
```

```
##    model n.muts coes sigma   n false.rej true.unq mean.post
## 1  stick      4  0.1  0.03 100      0.03     0.00 0.3704342
## 2   mult      4  0.1  0.03 100      0.01     0.01 0.5570763
## 3    add      4  0.1  0.03 100      0.00     0.00 0.4347485
## 4  stick      4  0.3  0.03 100      0.00     0.78 0.9689976
## 5   mult      4  0.3  0.03 100      0.00     1.00 0.9992401
## 6    add      4  0.3  0.03 100      0.00     0.92 0.9801675
## 7  stick      4  0.1  0.06 100      0.00     0.00 0.3828113
## 8   mult      4  0.1  0.06 100      0.01     0.00 0.4146707
## 9    add      4  0.1  0.06 100      0.00     0.00 0.3617309
## 10 stick      4  0.3  0.06 100      0.00     0.33 0.8402950
## 11  mult      4  0.3  0.06 100      0.00     0.97 0.9910413
## 12   add      4  0.3  0.06 100      0.00     0.12 0.8425023
```

IV. Other examples

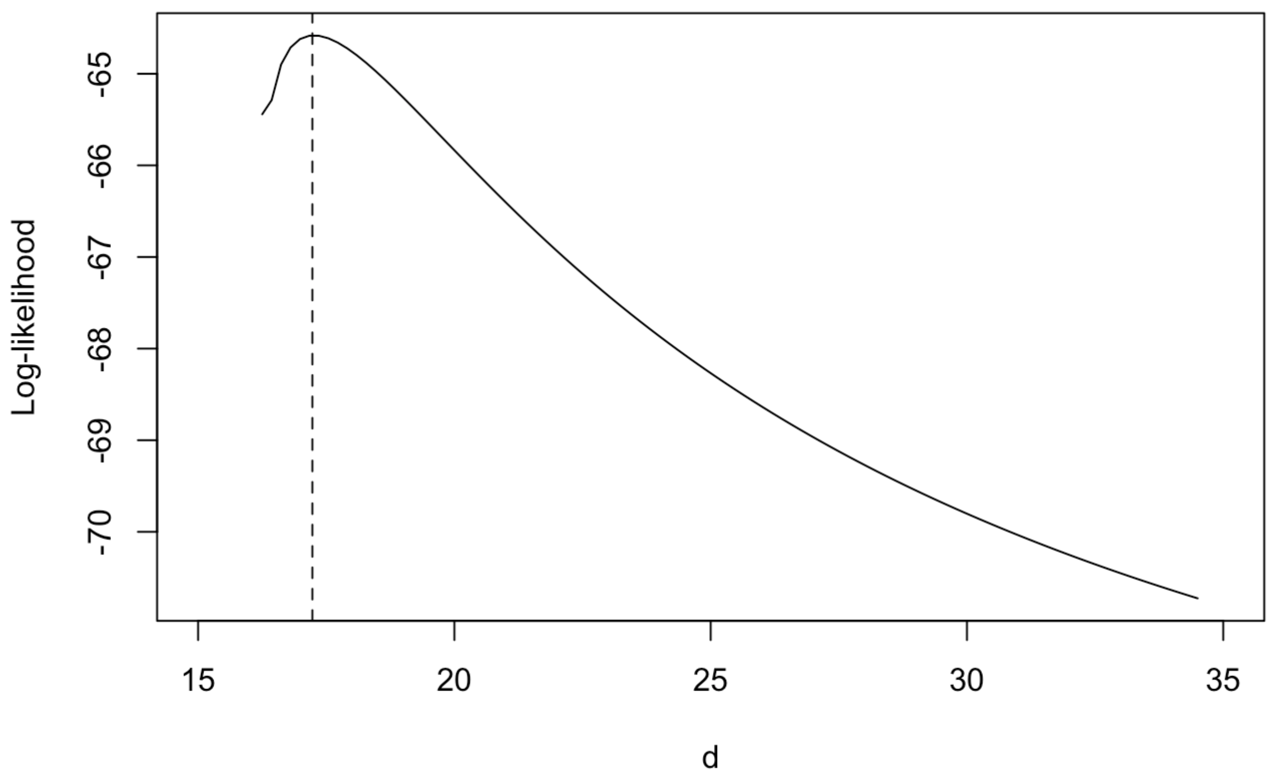
We analyzed two datasets in the code above (one in *II. Basic Anlaysis* and another in the first half of *III. Detailed Features*). Here we walk through analysis of another couple datasets to further illustrate the process.

A. Mutation pair data from bacteriophage ID11

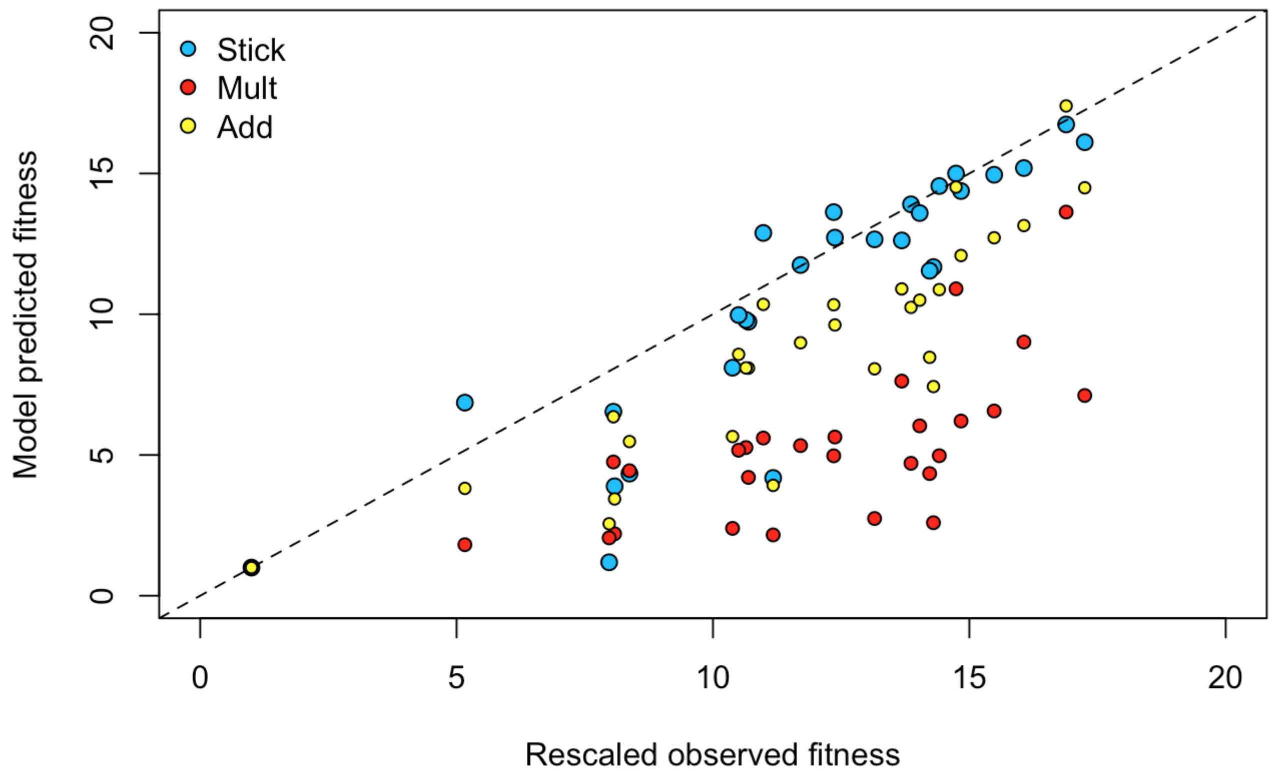
This data comes from Caudle, Miller and Rokyta, 2014, Environment determines epistatic patterns for a ssDNA virus, Genetics, 196(1):267-279. Note that in the raw data, the wild type has fitness of -4.1. While the additive and stickbreaking models deal with negative fitnesses just fine, it is problematic for the multiplicative model. We therefore shift the data so that wildtype is reassigned a value of 1.0 by simply adding 5.1 to all fitness values.

1. Fit models to data

```
    data(caudle.data)
    fit.col <- which(colnames(caudle.data)=="Fitness")
    n.singles <- fit.col-1
    n.muts <- n.singles

    fitness.shift <- -caudle.data$Fitness[1] + 1      # add this much to each fitness. N
eed wt to have positive fitness
    caudle.data$Fitness <- caudle.data$Fitness + fitness.shift
    geno.matrix <- geno.matrix.Caud <- caudle.data[,1:(fit.col-1)]
    fit.matrix <- fit.matrix.Caud <- t(t(caudle.data[,fit.col:fit.col]))
    wts <- c(2,1)
    d.range <- c(max(fit.matrix)-fit.matrix[1], 2*max(fit.matrix))
  d.adj.max <- 1.1
  d.hat.MLE <- estimate.d.MLE(geno.matrix, fit.matrix, d.range=d.range, wts)
  d.vect <- seq(d.range[1], d.range[2], length.out=100)

  # Visualize the likelihood of d
  like.profile <- calc.stick.logLn(geno.matrix, fit.matrix, d.vect, wts)
  plot(d.vect, like.profile, type="l", xlim=c(15,35), ylab="Log-likelihood", xlab="d")
  abline(v=d.hat.MLE, lty="dashed")
```



```
  d.hat.RDB <- estimate.d.RDB(geno.matrix, fit.matrix, no.est=NA)$d.hat.RDB  # doesn't w
ork for double data
  d.hat.seq <- estimate.d.sequential(geno.matrix, fit.matrix, d.hat.MLE, d.hat.RDB, d.ra
nge)

  fit.stick <- fit.stick.Caud <- fit.stick.model.given.d(geno.matrix, fit.matrix, d.hat.
seq, wts=wts, run.regression=TRUE)
  fit.mult <- fit.mult.Caud <- fit.mult.model(geno.matrix, fit.matrix, wts=wts)
  fit.add <- fit.add.Caud <- fit.add.model(geno.matrix, fit.matrix, wts=wts)
  fit.smry <- fit.smry.Caud <- summarize.fits.for.posterior.calc(fit.stick, fit.mult, fi
t.add)

  nmuts <- apply(geno.matrix, 1, sum)
  col.v <- c("deepskyblue", "red", "yellow")

  plot(x=fit.matrix, y=fit.stick$pred.matrix$pred, ylim=c(0,20), xlim=c(0,20), ylab="Mod
el predicted fitness", xlab="Rescaled observed fitness", pch=21, bg=col.v[1], cex=1.1)
  abline(0,1, lty="dashed")
  points(x=fit.matrix, y=fit.mult$pred.matrix$pred, pch=21, bg=col.v[2], cex=0.9)
  points(x=fit.matrix, y=fit.add$pred.matrix$pred, pch=21, bg=col.v[3], cex=0.8)
    legend("topleft", pch=21, pt.bg=col.v, bty="n", legend=c("Stick", "Mult", "Add"))
```



2. Simulate training data matching data structure

As usual, you must specify priors and other parameters that control the training simulation process. Toggle `sim.training.data.from.priors` to FALSE if you don't want to run this code.

```r
sim.training.data.from.priors <- TRUE

if (sim.training.data.from.priors == TRUE){
  mods.to.sim <- c("stick", "mult", "add")
  coes.prior <- c(0.05, 0.5)
  sigs.prior <- c(0, 0.25)
  n.samps.per.mod <- 1000

  d.true <- 1
  d.range <- c(0.1, 10)
  d.adj.max <- 1.1
  w.wt <- 1
  wts <- c(2,1)
  print.interval <- 25

  outdir <- system.file("extdata", package="Stickbreaker")
  file.name <- paste("Training_simulated_priors_fit_caudle_data.txt", sep="")
  outpath <- paste(outdir, file.name, sep="/")

  sim.partial.data.from.priors.for.mod.selection(geno.matrix, coes.prior=coes.prior, sigs.prior=sigs.prior, mods.to.sim=mods.to.sim, d.true=d.true, d.range=d.range, d.adj.max=d.adj.max, w.wt=w.wt, wts=wts, outpath=outpath, n.samps.per.mod=n.samps.per.mod, coe.sim.model="identical", coe.dist.par=NA, print.interval=25)
}
```

```
## [1] "stick"
## [1] 1
## [1] 25
## [1] 50
## [1] 75
## [1] 100
## [1] 125
## [1] 150
## [1] 175
## [1] 200
## [1] 225
## [1] 250
## [1] 275
## [1] 300
## [1] 325
## [1] 350
## [1] 375
## [1] 400
## [1] 425
## [1] 450
## [1] 475
## [1] 500
## [1] 525
## [1] 550
## [1] 575
## [1] 600
## [1] 625
## [1] 650
## [1] 675
## [1] 700
## [1] 725
## [1] 750
## [1] 775
## [1] 800
## [1] 825
## [1] 850
## [1] 875
## [1] 900
## [1] 925
## [1] 950
## [1] 975
## [1] 1000
## [1] "mult"
## [1] 1
## [1] 25
## [1] 50
## [1] 75
## [1] 100
## [1] 125
## [1] 150
## [1] 175
## [1] 200
## [1] 225
## [1] 250
## [1] 275
## [1] 300
## [1] 325
## [1] 350
## [1] 375
## [1] 400
## [1] 425
## [1] 450
## [1] 475
## [1] 500
## [1] 525
## [1] 550
## [1] 575
## [1] 600
## [1] 625
## [1] 650
## [1] 675
## [1] 700
## [1] 725
## [1] 750
## [1] 775
## [1] 800
## [1] 825
## [1] 850
## [1] 875
## [1] 900
## [1] 925
## [1] 950
## [1] 975
## [1] 1000
## [1] "add"
## [1] 1
## [1] 25
## [1] 50
## [1] 75
## [1] 100
## [1] 125
## [1] 150
## [1] 175
## [1] 200
## [1] 225
## [1] 250
## [1] 275
## [1] 300
## [1] 325
## [1] 350
## [1] 375
## [1] 400
## [1] 425
## [1] 450
## [1] 475
## [1] 500
## [1] 525
## [1] 550
## [1] 575
## [1] 600
## [1] 625
## [1] 650
## [1] 675
## [1] 700
## [1] 725
## [1] 750
## [1] 775
## [1] 800
## [1] 825
## [1] 850
## [1] 875
## [1] 900
## [1] 925
## [1] 950
## [1] 975
## [1] 1000
```

3. Fit training data to multinomial model

```
  model.file <- "nnet_mod_R2_P_caudle"
  moddir <- system.file("models", package="Stickbreaker")
  modpath <- paste(moddir, model.file, sep="/")
  data.file <- "Training_simulated_priors_fit_caudle_data.txt"
  min.R2 <- -1

  fit.multinomial <- TRUE

  if (fit.multinomial == TRUE){

    mod.formula <- as.formula(model ~ R2.stick + R2.mult + R2.add + P.stick + P.mult +
P.add)
    indir <- system.file("extdata", package="Stickbreaker")
    inpath <- paste(indir, data.file, sep="/")
    if (file.exists(inpath)){
      data <- read.table(file=inpath, header=TRUE)
      data$R2.stick[which(data$R2.stick<min.R2)] <- min.R2
      data$R2.mult[which(data$R2.mult<min.R2)] <- min.R2
      data$R2.add[which(data$R2.add<min.R2)] <- min.R2
      fit.nmet.model <- fit.nnet.multinomial.regression(data, mod.formula)
      m <- fit.nnet.multinomial.regression(data, mod.formula)
      modelout <- paste(modpath, ".rda", sep="")
      saveRDS(m, file=modelout)
    }
  }
```

4. Calculate posteriors on Cauldle data

```
  model.file <- "nnet_mod_R2_P_caudle"
  moddir <- system.file("models", package="Stickbreaker")
  modpath <- paste(moddir, model.file, sep="/")
  modelin <- paste(modpath, ".rda", sep="")
  model <- readRDS(modelin)
  post.data <- post.data.Caud <- calculate.posteriors.for.datasets(fit.smry.Caud,
model)
  smry.w.probs.Caud <- as.list(post.data)
  print(post.data.Caud)
```

```
##    R2.stick   P.stick   R2.mult   P.mult     R2.add     P.add         add
## 1 0.3104546 -11.53816 -5.151902 -29.7853 -0.5032295 -29.86559 0.001883599
##           mult     stick
## 1 3.077308e-06 0.9981133
```

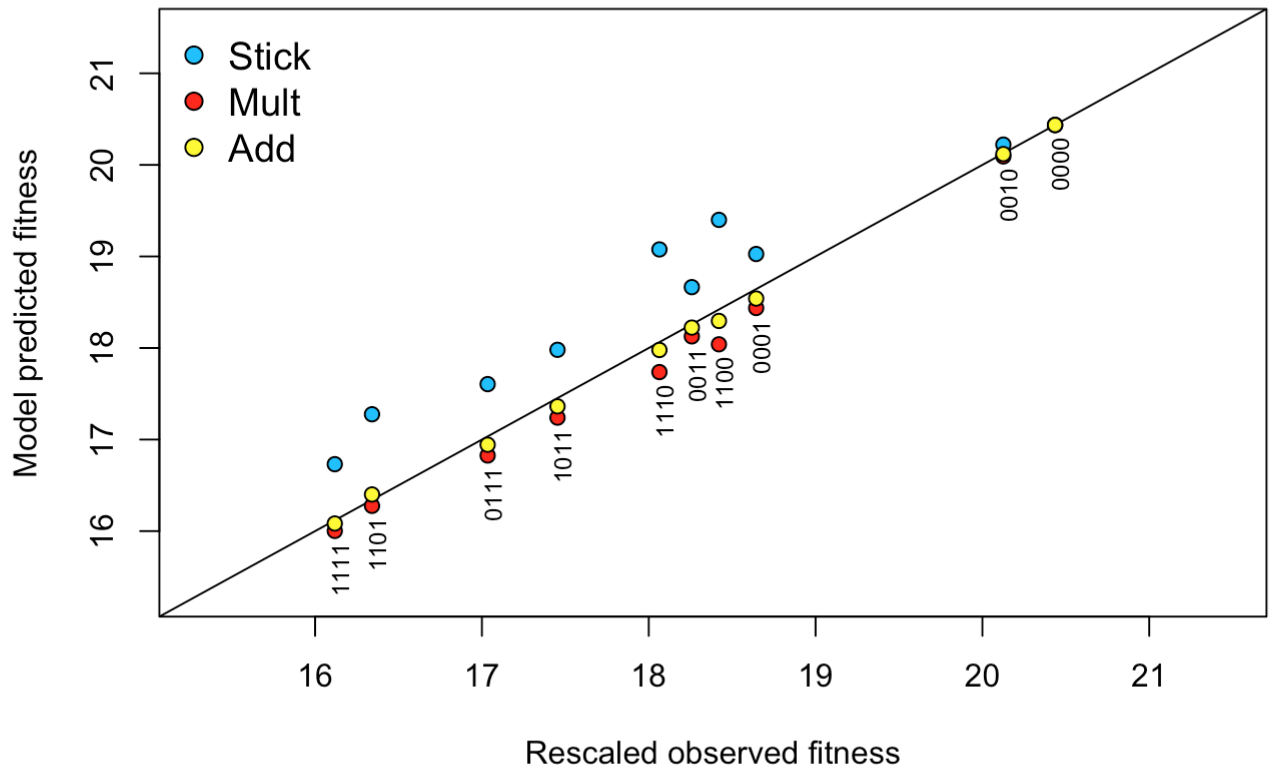
The results shows that, of the three models, stickbreaking is a far better approximation than additive or multiplicative. When only given these three options, the posterior probability of stickbreaking is > 99.8%. This result also shows how the method can be used with this type of single- and double-mutation data.

B. Synonymous recoded deleterious poliovirus data

This data is from Burns et al., 2006, Modulation of poliovirus replication fitness in HeLa cells by deoptimization of synonymous codon usage in the capsid region, J. Virology 80(7):3259-3272. Note that the raw data are in paper are plaque forming units per mL (PFU/mil). We take the log of this since when growth is exponential, growth rate will be proportional to the log of population size.

1. Fit data to models

```
  datadir <- system.file("data", package="Stickbreaker")
  load(paste(datadir, "burns.data.rda", sep="/"))
  data <- burns.data
  wts <- c(2,1)
  n.genos <- length(data[,1])
  n.muts <- length(data[1,])-1
  mut.i <- n.muts - 2
  n.mut.i <- mut.i
  geno.matrix <- geno.matrix.Burns <- data[,seq(1, n.muts)]
  fit.matrix <- fit.matrix.Burns <-  as.matrix(data[,(n.muts+1)])
  fit.matrix <- log(fit.matrix)
  muts.by.geno <- apply(geno.matrix, 1, sum)

  d.range <- c(min(fit.matrix), max(fit.matrix)*1.5)
  d.adj.max <- 1.1
  d.hat.MLE <- estimate.d.MLE(geno.matrix, fit.matrix, d.range=d.range)
  d.hat.RDB <- estimate.d.RDB(geno.matrix, fit.matrix)$d.hat.RDB
  d.hat.seq <- estimate.d.sequential(geno.matrix, fit.matrix, d.hat.MLE, d.hat.RDB, d.ra
nge)
  fit.stick <- fit.stick.Burns <- fit.stick.model.given.d(geno.matrix, fit.matrix, d.ha
t.seq, run.regression=TRUE)
  fit.mult <- fit.mult.Burns <- fit.mult.model(geno.matrix, fit.matrix, wts=wts)
  fit.add <- fit.add.Burns <- fit.add.model(geno.matrix, fit.matrix, wts=wts)
  fit.smry <- fit.smry.Burns <- summarize.fits.for.posterior.calc(fit.stick, fit.mult, f
it.add)

  lims=c(min(fit.matrix)*0.95, max(fit.matrix)*1.05)
  preds <-
as.data.frame(cbind(obs=fit.stick$pred.matrix$fit,stick=fit.stick$pred.matrix$pred,
mult=fit.mult$pred.matrix$pred, add=fit.add$pred.matrix$pred))
  rownames(preds) <- fit.stick$pred.matrix$string
  max.fits <- apply(preds, 1, function(x) max(x[2:4]))
  min.fits <- apply(preds, 1, function(x) min(x[2:4]))

  cex.v <- c(1,1,1)
  lab.cex <- 0.75
  leg.location <- "topleft"
  leg.cex <- 1.2
  col.v <- c("deepskyblue", "red", "yellow")
  plot(y=preds$stick, x=preds$obs, pch=21, bg=col.v[1], xlim=lims, ylim=lims,
cex=cex.v[1], xlab="Rescaled observed fitness", ylab="Model predicted fitness")
  abline(0,1)
  points(y=preds$mult, x=preds$obs, pch=21, bg=col.v[2], cex=cex.v[2])
  points(y=preds$add, x=preds$obs, pch=21, bg=col.v[3], cex=cex.v[3])
  text(y=min.fits, x=preds$obs, pos=1, off=1, srt=90, labels=rownames(preds), cex=lab.ce
x)
  legend("topleft", bty="n", pch=rep(21, 3), pt.bg=col.v, legend=c("Stick", "Mult", "Ad
d"), cex=leg.cex)
```



2. Simulate training data matching this dataset

```
  sim.training.data.from.priors <- TRUE

  if (sim.training.data.from.priors == TRUE){
    mods.to.sim <- c("stick", "mult", "add")
    coes.prior <- c(-0.05, -1)
    sigs.prior <- c(0, 0.25)
    n.samps.per.mod <- 1000

    d.true <- 1
    d.range <- c(1, 10)
    d.adj.max <- 1.1
    w.wt <- fit.matrix[1,1]
    wts <- c(2,1)
    print.interval <- 25

    outdir <- system.file("extdata", package="Stickbreaker")
    file.name <- paste("Training_simulated_priors_fit_burns2_data.txt", sep="")
    outpath <- paste(outdir, file.name, sep="/")

    sim.partial.data.from.priors.for.mod.selection(geno.matrix, coes.prior=coes.prior, s
igs.prior=sigs.prior, mods.to.sim=mods.to.sim, d.true=d.true, d.range=d.range,
d.adj.max, w.wt=w.wt, wts=wts, outpath=outpath, n.samps.per.mod=n.samps.per.mod, coe.si
m.model="identical", coe.dist.par=NA, print.interval=print.interval)

  }
```

```
## [1] "stick"
## [1] 1
## [1] 25
## [1] 50
## [1] 75
## [1] 100
## [1] 125
## [1] 150
## [1] 175
## [1] 200
## [1] 225
## [1] 250
## [1] 275
## [1] 300
## [1] 325
## [1] 350
## [1] 375
## [1] 400
## [1] 425
## [1] 450
## [1] 475
## [1] 500
## [1] 525
## [1] 550
## [1] 575
## [1] 600
## [1] 625
## [1] 650
## [1] 675
## [1] 700
## [1] 725
## [1] 750
## [1] 775
## [1] 800
## [1] 825
## [1] 850
## [1] 875
## [1] 900
## [1] 925
## [1] 950
## [1] 975
## [1] 1000
## [1] "mult"
## [1] 1
## [1] 25
## [1] 50
## [1] 75
## [1] 100
## [1] 125
## [1] 150
## [1] 175
## [1] 200
## [1] 225
## [1] 250
## [1] 275
## [1] 300
## [1] 325
## [1] 350
## [1] 375
## [1] 400
## [1] 425
## [1] 450
## [1] 475
## [1] 500
## [1] 525
## [1] 550
## [1] 575
## [1] 600
## [1] 625
## [1] 650
## [1] 675
## [1] 700
## [1] 725
## [1] 750
## [1] 775
## [1] 800
## [1] 825
## [1] 850
## [1] 875
## [1] 900
## [1] 925
## [1] 950
## [1] 975
## [1] 1000
## [1] "add"
## [1] 1
## [1] 25
## [1] 50
## [1] 75
## [1] 100
## [1] 125
## [1] 150
## [1] 175
## [1] 200
## [1] 225
## [1] 250
## [1] 275
## [1] 300
## [1] 325
## [1] 350
## [1] 375
## [1] 400
## [1] 425
## [1] 450
## [1] 475
## [1] 500
## [1] 525
## [1] 550
## [1] 575
## [1] 600
## [1] 625
## [1] 650
## [1] 675
## [1] 700
## [1] 725
## [1] 750
## [1] 775
## [1] 800
## [1] 825
## [1] 850
## [1] 875
## [1] 900
## [1] 925
## [1] 950
## [1] 975
## [1] 1000
```

3. Fit training data to multinomial model

```
  model.file <- "nnet_mod_R2_P_burns2"
  moddir <- system.file("models", package="Stickbreaker")
  modpath <- paste(moddir, model.file, sep="/")
  data.file <- "Training_simulated_priors_fit_burns2_data.txt"

  fit.multinomial <- TRUE
  min.R2 <- -1

  if (fit.multinomial == TRUE){

    mod.formula <- as.formula(model ~ R2.stick + R2.mult + R2.add + P.stick + P.mult +
P.add)
    indir <- system.file("extdata", package="Stickbreaker")
    inpath <- paste(indir, data.file, sep="/")
    if (file.exists(inpath)){
      data <- read.table(file=inpath, header=TRUE)
      data$R2.stick[which(data$R2.stick<min.R2)] <- min.R2
      data$R2.mult[which(data$R2.mult<min.R2)] <- min.R2
      data$R2.add[which(data$R2.add<min.R2)] <- min.R2
      fit.nmet.model <- fit.nnet.multinomial.regression(data, mod.formula)
      m <- fit.nnet.multinomial.regression(data, mod.formula)
      modelout <- paste(modpath, ".rda", sep="")
      saveRDS(m, file=modelout)
    }
  }
```

4. Calculate posteriors on Burns data

```
  model.file <- "nnet_mod_R2_P_burns2"
  moddir <- system.file("models", package="Stickbreaker")
  modpath <- paste(moddir, model.file, sep="/")
  modelin <- paste(modpath, ".rda", sep="")
  mod <- readRDS(modelin)
  post.probs <- post.probs.Burns <- calculate.posteriors.for.datasets(fit.smry.Burns, mo
d=mod)
  print(post.probs.Burns)
```

```
##    R2.stick   P.stick   R2.mult  P.mult    R2.add    P.add       add
## 1 0.6657245 -12.33277 0.9662473 -3.3358 0.9954908 -2.52204 0.9996889
##          mult        stick
## 1 2.37535e-05 0.0002873343
```

In this case, the multinomial regression strongly favors the additive model as the best explanation of the data.