A Computational Bayesian Approach to Ternary Network Estimation (ternarynet)

Matthew N. McCall and Anthony Almudevar

April 27, 2020

Contents

1 Introduction

This document describes ternarynet, which implements a computational Bayesian algorithm to estimate a ternary network from perturbation data. We strong recommend reading the paper, *Fitting Boolean Networks from Steady State Perturbation Data* (Almudevar *et. al* 2011) before proceeding with this vignette.

2 Getting Started

First begin by downloading and installing the ternarynet package.

```
> library(ternarynet)
```

2.1 Basic Input Data

The input data to the main ternarynet functions are a matrix of steady state observations and a matrix of perturbation experiments, where columns represent perturbation experiments and rows represent measured genes. The perturbation matrix consists of all zeros except for those genes experimentally perturbed, which are denoted by 1 if overexpressed or -1 if underexpressed. The steady state matrix consists of the response of each measured gene to each perturbation. Note that the perturbed gene(s) in each experiment are forced to response. For example if we perturb each of five genes by over-expressing each one, the perturbation matrix would be:

```
> perturbationObj
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    1    0    0    0
[3,]    0    0    1    0    0
[4,]    0    0    0    1    0
[5,]    0    0    0    0    1
```

A potential steady state matrix based on the perturbation experiments above is:

```
> steadyStateObj
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    1    1    1    0
[3,]    0    1    1    1    0
[4,]    0    1    1    1    0
[5,]    1    0    0    0    1
```

We can interpret the first perturbation experiment, persistent over-expression of gene 1 (column 1), as resulting in over-expression of gene 5.

2.2 Model Fitting

There are numerous modeling parameters that we could alter (these are outlined in the help files and described in Almudevar et al. (2011)), but in this example, we will call the ternary network fit using the default parameters (except for setting the random seed):

```
> tnfit <- tnetfit(steadyStateObj, perturbationObj, xSeed=11235)
```

This creates a ternaryFit object, tnfit, that contains the results of the model fitting. We can assess the model fit by examining the traces of four key parameters:

```
> plotTraces(tnfit)
```

2.3 Posterior Sampling

Once we have fit the ternary network model, we can sample from the posterior density on the model space:

```
> tnpost <- tnetpost(tnfit, xSeed=11235)
```

2.4 Summary Results

The ternaryPosterior object contains a wealth of information that can be used to answer a wide variety of statistical and biological questions; however, it is often convenient to summarize this information. The first summarization we will consider is reporting the posterior probabilities of the attractors resulting from either transient or persistent perturbations. These summaries can be produced as follows:

```
> attractorSummary(tnpost)
```

```
      index 1 2 3 4 5 PostProb
 [1,]     1 1 0 0 0 1        0
 [2,]     1 0 0 0 0 0        1
 [3,]     2 0 1 1 1 0        0
 [4,]     2 0 1 1 1 0        1
 [5,]     3 0 1 1 1 0        0
 [6,]     3 0 1 1 1 0        1
 [7,]     4 0 1 1 1 0        0
 [8,]     4 0 1 1 1 0        1
 [9,]     5 0 0 0 0 1        0
[10,]     5 0 0 0 0 0        1
```

```
> attractorSummary(tnpost, wildtype=FALSE)
```

```
      index 1 2 3 4 5 PostProb
 [1,]     1 1 0 0 0 1        0
 [2,]     1 1 0 0 0 1        1
 [3,]     2 0 1 1 1 0        0
 [4,]     2 0 1 1 1 0        1
 [5,]     3 0 1 1 1 0        0
 [6,]     3 0 1 1 1 0        1
 [7,]     4 0 1 1 1 0        0
 [8,]     4 0 1 1 1 0        1
 [9,]     5 0 0 0 0 1        0
[10,]     5 0 0 0 0 1        1
```

The first summary provides the attractors for a transient perturbation (the response of the wildtype network) and second summary for a persistent perturbation. The first column provides the number of the perturbation experiment (corresponding to the columns in the perturbation matrix), and the last column provide the posterior probability of each attractor. The middle columns are a summary of each attractor, as described in Almudevar et al. (2011).

In addition to investigating the attractor structure, one might also want to examine the network topology. A simple summary of the topology can be generated as follows:

```
> graphPosterior(tnpost)
```

```
       0 1      2      3     4      5
1 0.4805 0 0.1315 0.1285 0.143 0.1165
```

```
2 0.0000 0 0.0000 1.0000 0.000 0.0000
3 0.0000 0 0.0000 0.0000 1.000 0.0000
4 0.0000 0 1.0000 0.0000 0.000 0.0000
5 0.0000 1 0.0000 0.0000 0.000 0.0000
```

This produces a matrix where rows are children and columns are parents of regulatory relationships. The values in the matrix are the marginal posterior probabilities of each relationship. The first column represents the probability of a given gene having no parents.

3 Session Info

```
> sessionInfo()


R version 4.0.0 (2020-04-24)
Platform: x86_64-apple-darwin17.0 (64-bit)
Running under: macOS Mojave 10.14.6

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib

locale:
[1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] ternarynet_1.32.0

loaded via a namespace (and not attached):
[1] compiler_4.0.0  magrittr_1.5    tools_4.0.0     igraph_1.2.5
[5] pkgconfig_2.0.3
```