test

16 March, 2018

Contents

1 Main 1

1.1 Finding priors for the beta distribution . . . 1

1.1.1 Constraint 1: . . . 2

1.1.2 Constraint 2 . . . 2

1.1.3 Basics for Bayesian inference . . . 2

1.2 Inferring a three variable Bayesian network . . . 3

1.3 Number of Bayesian networks: . . . 3

1.3.1 Comment on the likelihood-equivalent prior . . . 4

1.3.2 Drawbacks of binary bayesian networks . . . 5

1.3.3 Effect of imaginary sample size . . . 5

1.3.4 Plot posteiror for P(C|A) in different models . . . 5

1.4 Comparing DREAM5 competitors . . . 7

1.4.1 Method . . . 7

1.4.1.1 Data . . . 7

1.4.1.2 Formalism . . . 8

1.5 One-shot algorithms . . . 8

1.6 A homemade regressional predictor with xgboost([2]) . . . 9

1.7 Comparison of algorithms . . . 9

1.8 Assessing the performance of algorithms with bootstrapped subnetworks . . . 9

2 References 11

3 Appendix 11

3.1 Code . . . 11

3.1.1 Fitting the beta distribution to constraints . . . 11

3.1.2 Systematic helpers for inference on the [0,1] interval . . . 13

3.1.3 Bayesian network inference . . . 15

3.1.4 Construct posterior accroding to the graphical model . . . 18

3.1.4.1 Graphs . . . 20

3.1.5 Comparison of algortihms . . . 21

3.1.5.1 Homemade xgboost-based regressional predictor (Gxgb) . . . 21

3.1.5.2 Calculation of QC-metrics . . . 24

3.1.5.3 Routines for commonly used algorithms . . . 32

3.1.5.4 Bootstrapping on sub-networks . . . 33

1 Main

1.1 Finding priors for the beta distribution

Beta distribution is the natural prior for a binomial/bernoulli distribution. Considering distribution of a binomial variable $X \mid \theta \sim Binom(\theta)$, in order to make its marginalisd distribution $P(X) = \int P(X \mid \theta) P(\theta).d\theta$ analytically tractable, one of the choice is to assume $\theta \sim Beta(M, \alpha)$, so that:

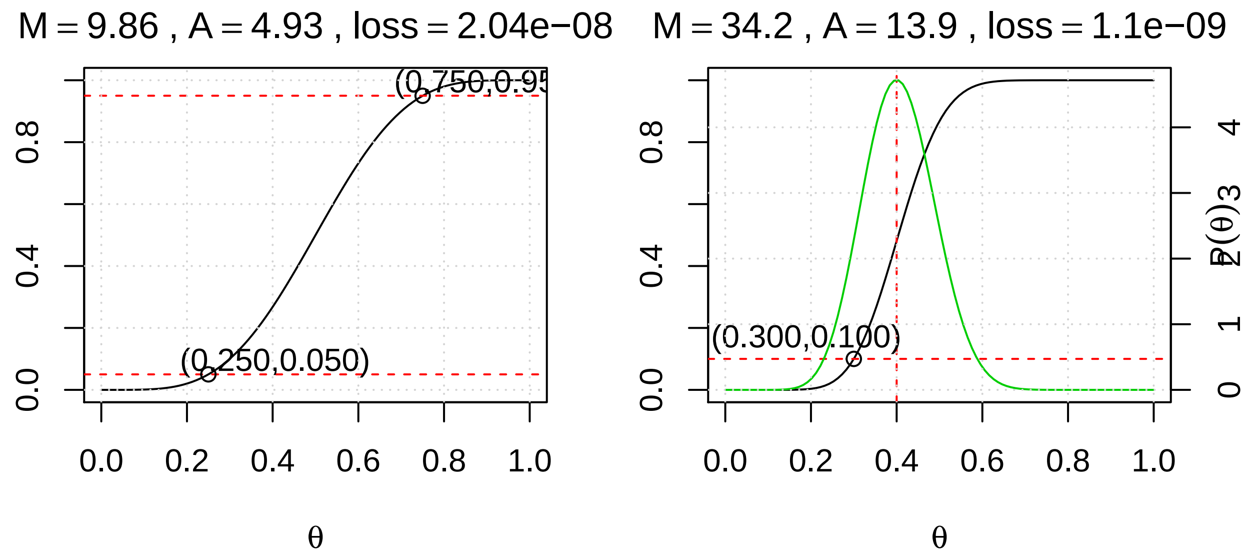


Figure 1: The shape of the beta distribution constrianed under the respective conditions. Left: Constraint 1. Right: Constraint 2. (green: PDF, black: CDF)

$$
\begin{aligned}
P(\theta) &= \frac{x^{\alpha-1}(1-x)^{M-\alpha-1}}{B(\alpha, M-\alpha)} \\
E(\theta) &= \frac{\alpha}{M}
\end{aligned}
$$

1.1.1 Constraint 1:

$$
\begin{aligned}
P(\theta \leq 0.25) = P(\theta \geq 0.75) &= 0.05 \\
P(\theta \leq 0.75) &= 0.95
\end{aligned}
$$

Fitted result: $\theta \sim$ Beta(4.933, 4.932)

1.1.2 Constraint 2

$$
\begin{aligned}
\text{argmax}_\theta(P(\theta)) &= 0.4 \\
P(\theta \leq 0.3) &= 0.1
\end{aligned}
$$

Fitted result: $\theta \sim$ Beta(13.863, 20.295)

1.1.3 Basics for Bayesian inference

$$
\begin{aligned}
\text{likelihood} : f(\theta) &= P(x \mid \theta) \\
\text{prior} : f(\theta) &= P(\theta) \\
\text{posterior} : f(\theta) &= P(\theta \mid x) = \frac{P(x \mid \theta)P(\theta)}{P(x)} \\
\text{marginal likelihood} : P(x) &= \int P(x \mid \theta)P(\theta).d\theta
\end{aligned}
$$

The observed toss sequence is: 011100101101

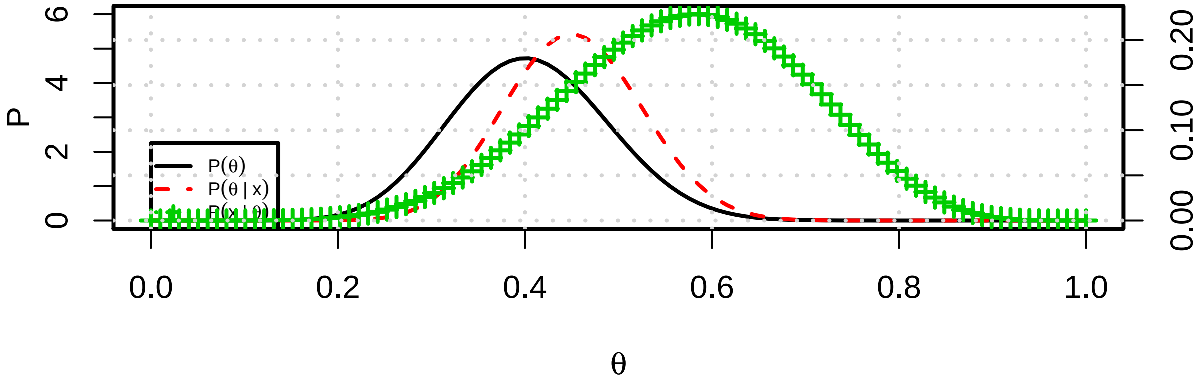


Figure 2: Inference of posterior distribution on parameter θ given mutatble sequence: 011100101101

Assume each coin toss is independent from each other, the likelihood of an observed sequence is only dependent on the total number of heads and not the sequence it occurred in. Denoting the coin toss as a sequence $\{x_i\}$ where $x_i \in \{0, 1\}$, we have

$$\begin{aligned} \#\text{head} &= \mathbb{1}\{x_i = 1\} \\ \#\text{head} &\sim Binom(|\{x_i\}|, \theta) \end{aligned}$$

Combining with the prior $\theta \sim$ Beta(13.863, 20.295), I calculated the marginal likelihood numerically to be $P(x) = 0.11$ and derived the posterior distribution accordingly (see figure 2). MLE is obtained at $\hat{\theta} = 0.45$

1.2 Inferring a three variable Bayesian network

I reparametrise the joint probability of the graph by replacing the conditional probability $P(child|parent)$ to be the quotient of two joint distirbution $\frac{P(child,parent)}{P(parent)}$. Because $P(child|parent)$ enters the marginalised likelihood as a Beta-Binomial probability, $P(parent)$ enters the term as a Drichlet-Multinomial probability:

$$P(x_k) = \frac{(n!)\,\Gamma\left(\sum \alpha_k\right)}{\Gamma\left(n + \sum \alpha_k\right)} \prod_{k=1}^{K} \frac{\Gamma(x_k + \alpha_k)}{(x_k!)\,\Gamma(\alpha_k)}$$

where $\sum x_k = N$ is the partition of sample into k categories, α_k is the imaginary sample size for each category (also known as prior concentration). This formulation has the advantage of easier coding.

To be consistent with bnlearn and deal, I did discared the multinomial terms in the calculation, leading to

$$P(x_k) = \frac{\Gamma\left(\sum \alpha_k\right)}{\Gamma\left(n + \sum \alpha_k\right)} \prod_{k=1}^{K} \frac{\Gamma(x_k + \alpha_k)}{\Gamma(\alpha_k)}$$

1.3 Number of Bayesian networks:

A V-variable network has $V(V-1)/2$ bivariate interaction (edges), each interaction can have 3 possible status (A->B, A<-B, A B). Hence altogether there are $n(V) = 3^{V(V-1)/2}$ possible networks. For $V = 3$,$n(3) = 27$

However, for this exerecise, the serach space is restircted to the graph set $G = \{$no-edge, A-C only, A-C and B-C$\}$.

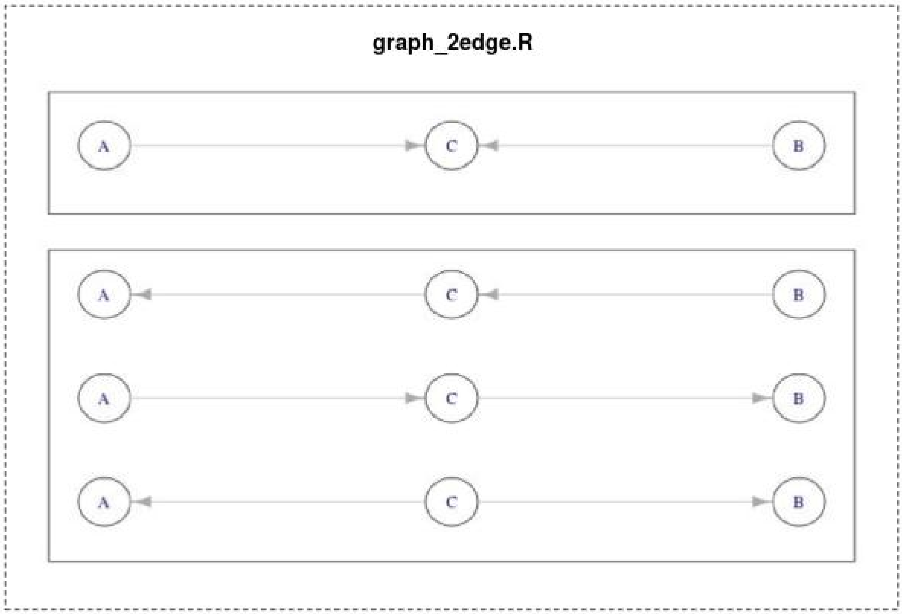
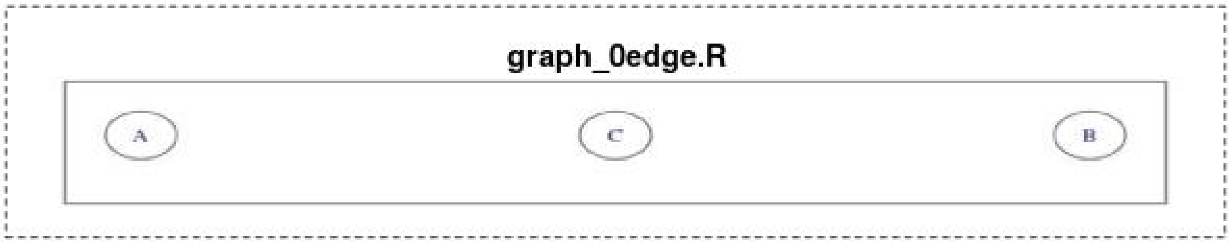
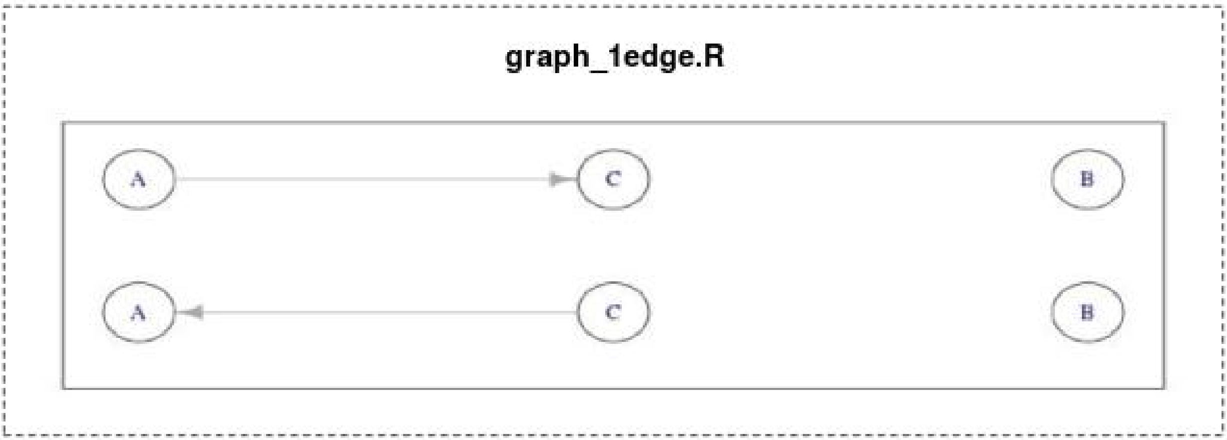


Figure 3: Graph with two edges but not A-B





1.3.1 Comment on the likelihood-equivalent prior

The likelihood-equivalent prior is set so that the imaginary sample size decreases as data is stratified by more variables. For example, if $P(A=1) \sim Beta(\eta(A_0), \eta(A_1))$, then the imaginary sample size for (A=1) is $\eta(A_1) = 2$. Hence if we then ask for $P(B=1 \mid A=1) \sim Beta(\eta(B_0 A_1), \eta(B_1 A_1))$, the imaginary η's must add up to the imageinary sample size of the condition $\mid A = 1$, (aka $\eta(B_0 A_1) + \eta(B_1 A_1) = \eta(A_1) = 2$).

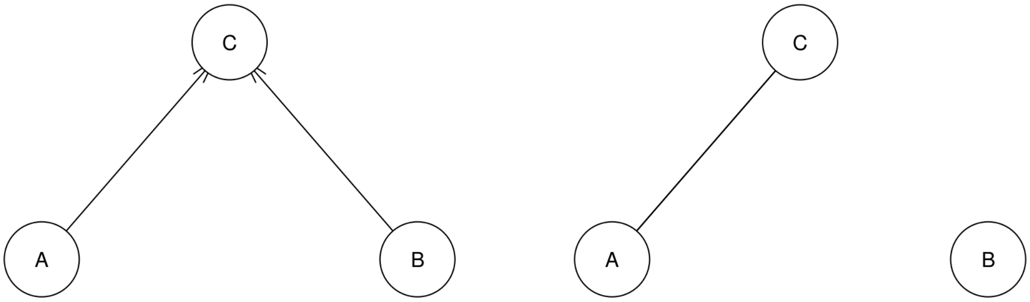


Figure 4: Best networks inferred using "bnlearn::pc.stable". Left: Dataset1. Right: Dataset2

Assuming two events are equally probable gives $\eta(B_0A_1) = \eta(B_1A_1) = 1$. For 3 variable, we can deduce $8\eta(ABC) = 4\eta(AB) = 2\eta(A) = \eta(0)$, setting $\eta(ABC) = 1$ gives $\eta(ABC) = 1, \eta(AB) = 2, \eta(A) = 4, \eta(0) = 8$, corresponding to different levels of stratification.

If a likelihood-preserving prior is used, then it is only the correlation structure that determines the relative feasibility of different graphs. Consider the 1-edge and 0-edge examples, the 0-edge example asserts $P(A \mid C = 0) = P(A \mid C = 1) = P(A)$, whereas the 1-edge example implies $P(A \mid C = 0) \neq P(A \mid C = 1)$, allowing an additional degree of freedom. The striking fact is that this additional DOF does not necessarily leads to a better model, in constrast to conventional mixture models where additional components always reduce likelihood. One of the reason is that the partiaion of $(A_0) = (A_0C_0) + (A_0C_1)$ is not arbitratry, but the general case is still confusing. A possible intution is that the additional DOF project the paramteric space to a higher dimension where the likelihood function overlaps less with the prior distribution.

1.3.2 Drawbacks of binary bayesian networks

If there are hidden latent variables in the bayes net, for example where the common parent of A and B (which is C) is conceived from the observers, then one will have to consider a graph with hidden variable in order to explain the data. In other words, a graphical prior needs to accommodate additional nodes to explain such data. Even though this is the case, it will be hard to express the case where n(A_0)=n(B_0)

1.3.3 Effect of imaginary sample size

Here we consider two imaginary sample sizes $\eta(0) = 8$ and $\eta(0) = 1$. A higher η indicates a sharper distribution of binomial probability θ (Setting $\eta(0) = 1$ implies $\eta(ABC) = 0.125, \eta(AB) = 0.25, \eta(A) = 0.5, \eta(0) = 1$)

The corresponding likelihood are calculated for both datasets (dat1 and dat2, see table 1).

- For dat1, the chain network (A-B-C) is the best at ISS=1, the A-B..C network is the best at ISS=8.
- For dat2, the chain network (A-B-C) is the best for ISS=1 and ISS=8

The prediction made by pc.stable is somewhat different (figure 4)

1.3.4 Plot posteiror for P(C|A) in different models

Here I visulise the posterior distribution using dataset 1 only. In order to show how different graphical models lead to different likelihood, I chose to contrast P(C|A) between [A][B][C](0-edge model) and [A][B][C|A] (1-edge model)

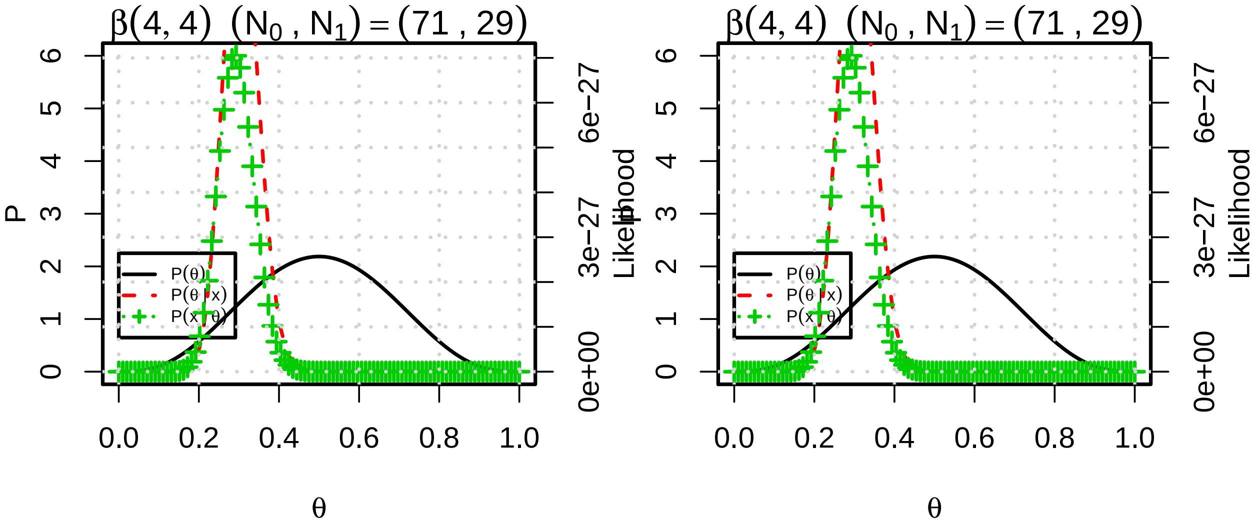


Figure 5: P(C|A)=P(C) according to the 0-edge graph, Left: P(C|A=0). Right: P(C|A=1)

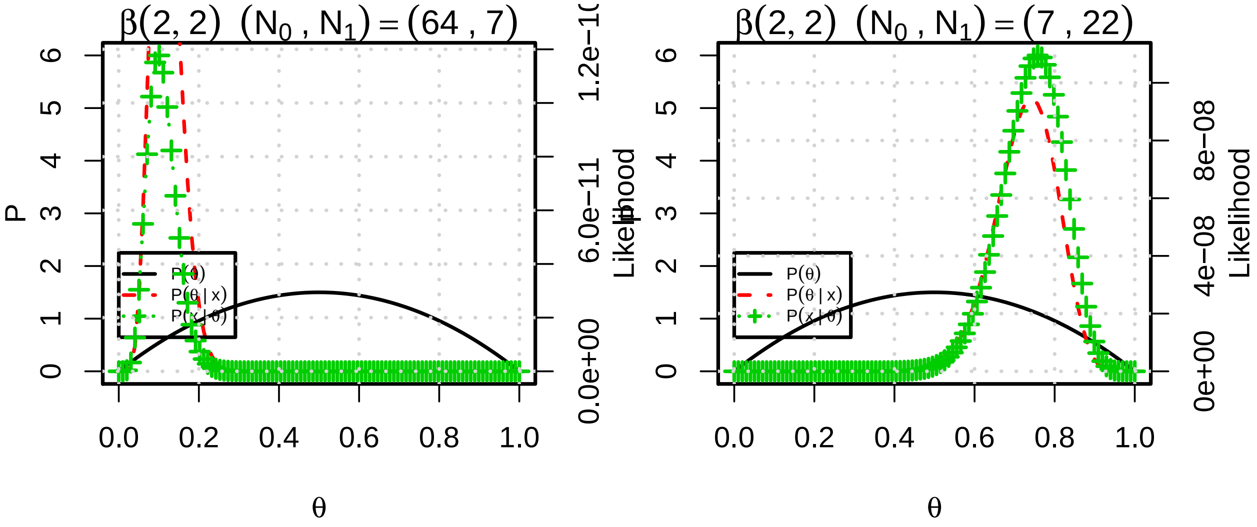


Figure 6: P(C|A) needs to be stratified according to the 1-edge graph, Left: P(C|A=0). Right: P(C|A=1)

Table 1: Marginalised likelihood of different network topology

model	myalgo.bde.iss	bnlearn.bde.iss	bnlearn.bic	iss	dat
[A][B][C]	-194.323	-194.323	-195.931	8	dat1
[A][B][C|A]	-175.536	-175.536	-176.906	8	dat1
[B][C][A|C]	-175.536	-175.536	-176.906	8	dat1
[B][C|B][A|C]	-168.817	-168.817	-170.591	8	dat1
[A][C|A][B|C]	-168.817	-168.817	-170.591	8	dat1
[C][A|C][B|C]	-168.817	-168.817	-170.591	8	dat1
[A][B][C|A:B]	-168.819	-168.819	-170.894	8	dat1
[A][B][C]	-196.617	-196.617	-195.931	1	dat1
[A][B][C|A]	-177.715	-177.715	-176.906	1	dat1
[B][C][A|C]	-177.715	-177.715	-176.906	1	dat1
[B][C|B][A|C]	-171.742	-171.742	-170.591	1	dat1
[A][C|A][B|C]	-171.742	-171.742	-170.591	1	dat1
[C][A|C][B|C]	-171.742	-171.742	-170.591	1	dat1
[A][B][C|A:B]	-170.894	-170.894	-170.894	1	dat1
[A][B][C]	-195.699	-195.699	-197.408	8	dat2
[A][B][C|A]	-180.318	-180.318	-181.025	8	dat2
[B][C][A|C]	-180.318	-180.318	-181.025	8	dat2
[B][C|B][A|C]	-178.579	-178.579	-179.994	8	dat2
[A][C|A][B|C]	-178.579	-178.579	-179.994	8	dat2
[C][A|C][B|C]	-178.579	-178.579	-179.994	8	dat2
[A][B][C|A:B]	-180.568	-180.568	-183.104	8	dat2
[A][B][C]	-198.093	-198.093	-197.408	1	dat2
[A][B][C|A]	-181.803	-181.803	-181.025	1	dat2
[B][C][A|C]	-181.803	-181.803	-181.025	1	dat2
[B][C|B][A|C]	-181.165	-181.165	-179.994	1	dat2
[A][C|A][B|C]	-181.165	-181.165	-179.994	1	dat2
[C][A|C][B|C]	-181.165	-181.165	-179.994	1	dat2
[A][B][C|A:B]	-183.684	-183.684	-183.104	1	dat2

In 0-edge model, $P(C|A) = P(C)$ and the distribution is indifferent for $A = 0$ and $A = 1$ (figure 5). The term enters likelihood function as a beta-binomial.

In contrast, the 1-edge model prescribes that $P(A, C) \neq P(A)P(C)$, and two separate distribution must be considered for $P(C|A)$ (figure 6). The $\prod_{C,A} P(C|A)$ term factors out to be $\prod_{C|A=0} P(C|A = 0) \prod_{C|A=0} P(C|A = 1)$, as the product of two beta-binomial with independent probability but the same prior. It would be interesting to explore the precise condition under which the factored likelihood exceeds the original single beta-binomial. Clearly, the 0-edge model fails to capture the difference between $P(C|A = 0)$ and $P(C|A = 1)$ (loglik=-194.3, compared to 1-edge loglik=-175.5), but the underlying mathematics remains to be dissected.

1.4 Comparing DREAM5 competitors

1.4.1 Method

1.4.1.1 Data

GNW network: I used the pre-simulated data "medium_network.rda" that included the expression levels of 40 genes along with the underlying true network to test the performance of various algorithms. This dataset is generated by GeneNetWeaver ([1]) hence called GNW dataset.

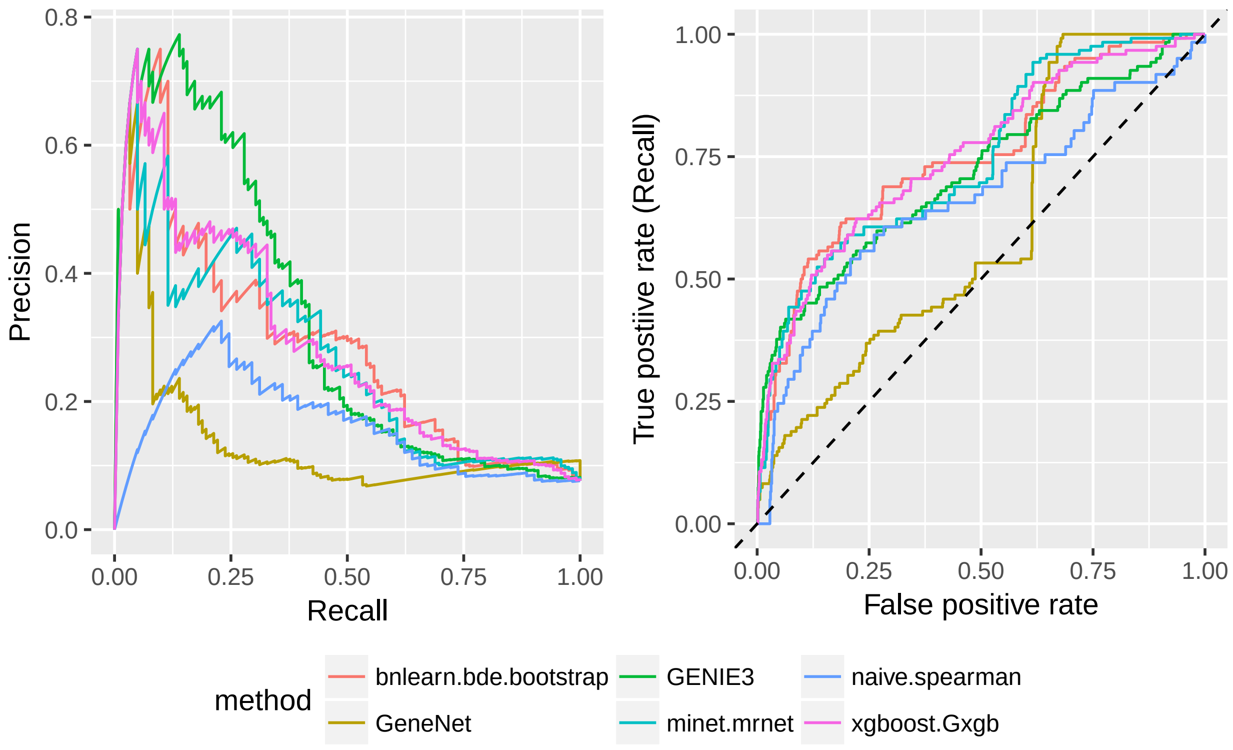


Figure 7: Comparing performances of different methods on the provided GNW sample

Table 2: Performance statistics on the GNW dataset

	AROC	AUPR	F1
GENIE3	0.719	0.322	0.397
xgboost.Gxgb	0.754	0.283	0.377
bnlearn.bde.bootstrap	0.751	0.282	0.380
minet.mrnet	0.746	0.268	0.386
naive.spearman	0.662	0.159	0.278
GeneNet	0.604	0.139	0.195

DREAM5 dataset: The E. coli network in the Dream5 competition has proved to be fittable and hence used here to evaluate the consistence of algorithms.

1.4.1.2 Formalism

I assumed the interaction matrix A_{ij} is symmetric and binary, and $A_{ij} = 1$ if there is interaction between gene i and gene j, otherwise $A_{ij} = 0$. The problem then reduce to a binary classification (existence of an undirected edge) and standard confusion matrix may be constructed to evaluate the performance. Specifically, area-under-precision-recall-curve (AUPR) and area-under-the-ROC (AROC) will be plotted for comparision, along with a list of F1 score.

1.5 One-shot algorithms

pc.stable and hc(score='bde') was run on the GNW dataset but the precision was not so exciting (table 3). However, notice pc.stable is giving a higher precision, potentially warranting a bootstrap checking in the future.

Table 3: Comparision of one-shot algorithms that output binary weights

TP	FP	TN	FN	PR	RC	SP	NPN	F1	method
8	14	705	53	0.364	0.131	0.981	0.930	0.193	bnlearn.pc
11	25	694	50	0.306	0.180	0.965	0.933	0.227	bnlearn.bde

1.6 A homemade regressional predictor with xgboost([2])

GENIE3 ([3]) uses random forest to fit the decision tree and then extract the importance. But here I will be fitting the trees with "xgboost" instead because of its simple interface and famous efficacy ([2]). Specifically, I fitted a decision tree to the function $x_j = f(\vec{x}_{-j})$, where x_j is expression of the chosen gene and x_{-j} is that of the rest genes. I used the Least sqaure loss ("reg:linear") for the fitting.

By the magic of xgboost, an importance vector is obatined (p^i_j indicates relative importance of gene i in predicting gene j, with $\sum_i p^i_j = 1$). I intuitively intepret it as the ratio of variance explained by splitting on gene i (though without firm matheamtical checking). Hence the importance vector is then scaled by the variance of the predictee ($P_{ij} = p^i_j \cdot Var(p_j)$) and then pooled together as the final confidence score for the connection between gene i and gene j, with a higher score indicating a higher likelihood of interaction.

1.7 Comparison of algorithms

A preliminary comparision is done between GeneNet, naive-spearman-predictor, bootstraped discrete bayesian net (bnlearn::boot.strength), GENIE3, homemade-xgboost (Gxgb) and MRnet. (figure 7, table 2). It is conceivable that GENIE3 gave the best AUPR (~0.377), but discrete bayesian net also gave a very robust prediction.

1.8 Assessing the performance of algorithms with bootstrapped subnetworks

Note all predictions are very far from perfect (AUPR=1.00), which leads to the evaluation part. In short, each algorithm has pros and cons. Regressional predictors usually has very poor intepretability, and there is no mathematical framework that would guarantee its convergence to the true network. With the discrete bayesian networks, the discretisation is a somewhat arbitrary process that might introduce artefact to the result, as well as choosing a suitable imaginary sample size (ISS) for the network (here I will stick to ISS=1 for simplicity).

Furthermore, a big headache for evaluating any algorithm is its performance will depend on the dataset. A most trivial example consider comparing the prediction result on the 40-gene dataset and then on a random sub-network generated by performing a random walk on the original network in a post-hoc fashion (so that the sub-network has non-zero adjacency matrix due to connection between neighbors). The AUPR is chosen as the main statistics and recorded for each sub-network to produce a bootstrapped distribution (figure 8). This procedure is performed for GENIE3 and bnlearn because they gave the best performance in preliminary benchmarking (table 2). Gxgb was not included due to its slowness (further optimisation is required).

The same bootstrapping was performed on the original DREAM5 training data for network 3 (the E. coli network, 4511 genes, figure 9). It can be seen that bootstrapped bnlearn performs similarly to the GENIE3, while taking a much shorter time to run (both measured using 16 cores). Although this may be due to the small size of the sub-network, (10 nodes), it is possible that GENIE3 is performing redundant computation than actual required by the inference. It is also observed that both algorithms performs consistently poorer on the DREAM5 datasets.

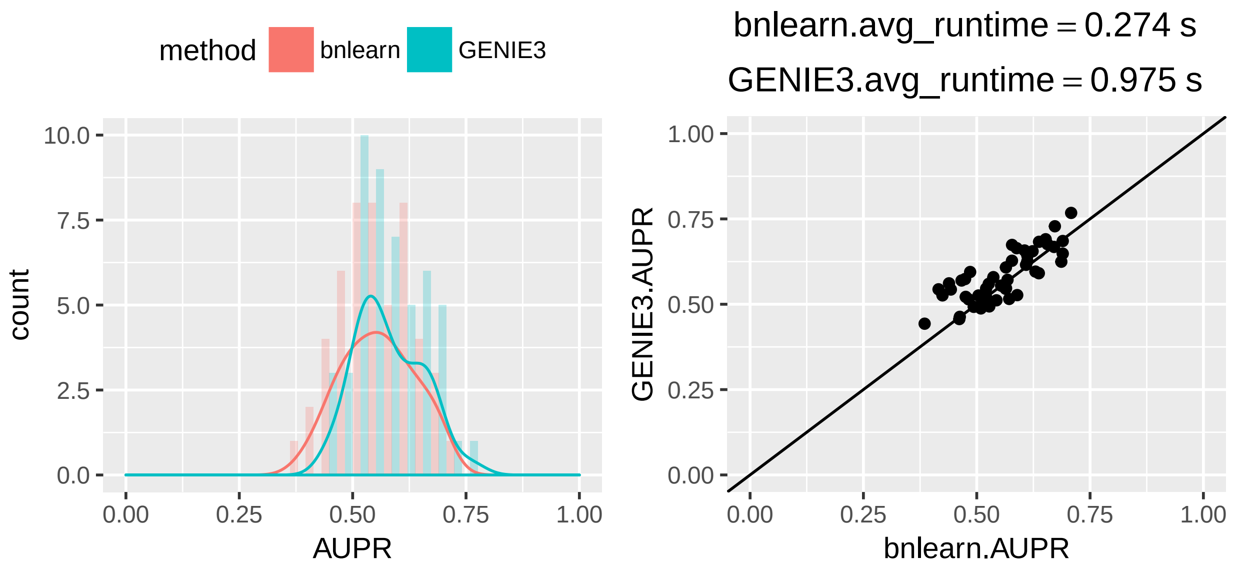


Figure 8: Comparing “GENIE3” and “bnlearn” on the bootstrapped subnets of the provided GeneNetWeaver simulation

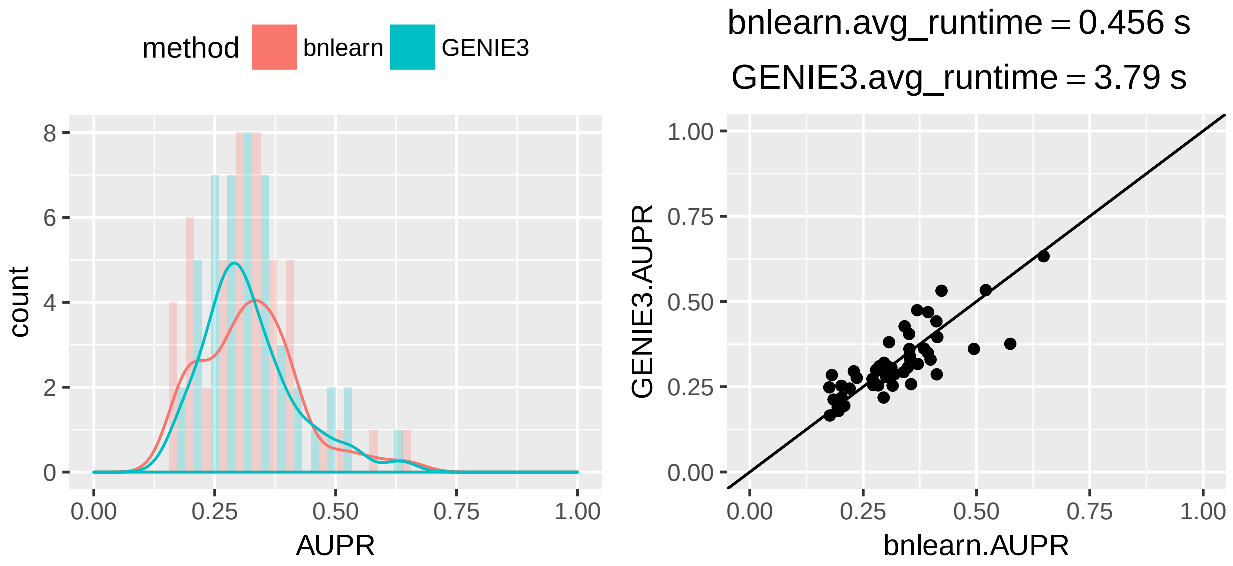


Figure 9: Comparing “GENIE3” and “bnlearn” on the bootstrapped subnets of DREAM5 network 3 (E. coli)

2 References

[1] Schaffter T, Marbach D, Floreano D. GeneNetWeaver: in silico benchmark generation and performance profiling of network inference methods. Bioinformatics (Oxford, England) 2011;27:2263–70. doi:10.1093/bioinformatics/btr373.

[2] Chen T, Guestrin C. XGBoost: A scalable tree boosting system. CoRR 2016;abs/1603.02754.

[3] Schaffter T, Marbach D, Floreano D. GeneNetWeaver: in silico benchmark generation and performance profiling of network inference methods. Bioinformatics (Oxford, England) 2011;27:2263–70. doi:10.1093/bioinformatics/btr373.

3 Appendix

3.1 Code

“Rutil” is my collection of utility and can be viewed online at https://github.com/shouldsee/Rutil/

This code appendix is also submitted as a html separately.

3.1.1 Fitting the beta distribution to constraints

```
library(Rutil)

preview<- function(p,xs =seq(0,1,length.out = 100),silent=F
                   ,xlab = 'x',ylab='y'
                   ,...){
  dots = list(...)
  # if (is.null(dots$xlab)){ dots$xlab='x'}
  # if (is.null(dots$ylab)){ dots$ylab='y'}
  ys= p(xs)
  xlab = (substitute(xlab))
  ylab = (substitute(ylab))
  # ys = deparse(substitute(y))
  # xs = deparse(substitute(xs))
  dat <- list(x=xs,y=ys
              ,xlab=xlab
              ,ylab=ylab
  )
  # dat <- list(y=ys)
  if (silent) {return(dat)}
  # with(data,plot())
  do.call(plot,args = c(dat,dots) )
}
pbeta.ma <- function(x,m,a){pbeta(x,a,m-a)}
dbeta.ma <- function(x,m,a){dbeta(x,a,m-a)}

loss <- function(param){
  m = param[1]
  a = param[2]
  # p<-function(x) {pbeta(x,a,m-a)}
  x = c(0.25,0.75)
```

```r
  y_true = c(0.05,0.95)
  ps <- pbeta(x,a,m-a)
  Metrics::mse(y_true,ps)
  # mean((ps - y_true)^2)
}


out = optim(c(2,1),loss,method = "BFGS",control = list(maxit=300))
out$par <- as.list(setNames(out$par,c('m','a')))
# outf = partial(dbeta.ma,m = out$par$m,a = out$par$a)
outf = partial(pbeta.ma,m = out$par$m,a = out$par$a)
out$p <- outf
out$d <- partial(dbeta.ma,m = out$par$m,a = out$par$a)
res1 <- out


loss <- function(param,debug = F){
  m = param[1]
  a = param[2]
  # a = max(a,1)
  # m = max(m,2)
  # p<-function(x) {pbeta(x,a,m-a)}
  # x = c(0.25,0.75)
  # y_true = c(0.05,0.95)

  p = partial(pbeta.ma,m=m,a=a)
  d = partial(dbeta.ma,m=m,a=a)
  # MODE = min(max((a-1)/(m-2),0),1)
  opout = optim( c(0.4),
                 # d,
                 {function(x){-d( min(max(x,1E-4),1-1E-4) )}},
                 method = 'BFGS', control = list(maxit=500))
  MODE= opout$par


  y_true = c( 0.4, 0.1)
  y_pred = c( MODE, p(0.3))
  if (debug){
    # print(MODE)
    print(p(0.3))
    cat('y_pred:',y_pred,'\n')
    cat('y_true:',y_true,'\n')
  }
  Metrics::mae(y_true, y_pred)
  # mean((ps - y_true)^2)
}

system.time({
  out = optim(c(10,5),loss,
              control = list(maxit=500)
              # ,method = "BFGS"
  )
  out$par <- as.list(setNames(out$par,c('m','a')))
  # outf = partial(dbeta.ma,m = out$par$m,a = out$par$a)
  outf = partial(pbeta.ma,m = out$par$m,a = out$par$a)
  out$p <- outf
  out$d <- partial(dbeta.ma,m = out$par$m,a = out$par$a)
  res2 <- out
  print(res2$value)
  loss(unlist(res2$par),debug=T)

})
```

3.1.2 Systematic helpers for inference on the [0,1] interval

```
#### Likelihood:
logL_maker <- function(obsB){
  N = length(obsB)
  X = sum(obsB)
  function(p1,...) {
    dbinom(x= X,size = N,p1,...)}
  # return( )
  # d(data)
  # pa * obj$prior
}

.infer<- setRefClass('infer',fields = list(
  # prior='function',
  Fparam = 'list',
  # dpos = 'function', ### Un-normalised posterior
  marg = 'list'
))

make_dpos <- function(ob){
  ob$Fparam$dpos <- function(x){ob$Fparam$L(x)*ob$Fparam$prior(x)}
}

marg_obs <- function(ob){
  f = ob$Fparam$dpos
  val = integrate( f, 0,1)$value
  ob$marg = list(obs=val)
  val
}

bayes_infer <- function(prior,L,low=0,high=1){
  # cnorm =
  post_raw <- function(PAR){prior(PAR)*L(PAR)}
  res = integrate(post_raw,low,high)
  cnorm = res$value
  post <- function(PAR){post_raw(PAR)/cnorm}

}
make_post <- function(ob){
  ob$Fparam$post <- bayes_infer(ob$Fparam$prior,ob$Fparam$L)
  # ob$Fparam$post <- function(PAR){ob$Fparam$L(PAR)*ob$Fparam$prior(PAR)/ob$marg$obs}
}


plot_bayes<-function(ob,ylab1=quote(P),ylab2='Likelihood') {
  # par(mar=c(4.2,4.1,1.1,4.1))
  par(lwd=2)
  i = 1
  preview(ob$Fparam$prior,type= 'l'
          ,xlab = quote(theta)
          ,ylab=''
          ,lty=i
          ,col=i
          ,ylim=c(0,6)
          ,pch = i
          # ,lwd = 2
  )
  mtext(ylab1,side=2,line=2)

  i = i+1
  # par(new = T)
  dat = preview(ob$Fparam$post,type= 'l',silent = T)
  do.call(lines,c(dat
                  ,lty=i
                  ,col=i
                  ,pch = i
                  # ,lwd = 2
  ))
  # text(labels = c("red line"))

  i = i+1
  legend(0, 2.25,
         legend=c(
           # substitute(bquote(P(theta))),
           # deparse(substitute(P(theta))),
           expression(P(theta)),
           expression(P(theta~'|'~x)),
           expression(P(x~'|'~theta))),
         # "Prior",
         # 'Posterior',"Likelihood"),
         pch =c(NA,NA,3),
         col=1:i,
         lty=1:i, cex=0.6)

  par(new = T)
  preview(ob$Fparam$L,type= 'b',silent = F
          ,lty=i
          ,col=i
          ,pch=i
          # ,lwd = 2

          ,axes = F,xlab='',ylab='')

  mtext(side=4,line=2,ylab2);axis(side = 4)
  grid()

}

obsS = '011100101101'
obsB <- strsplit(obsS,'')[[1]] =='1'

out <-res2

ob <-list()
##### Prior:
prior <- out$d
ob$Fparam$prior <- prior


L = logL_maker(obsB)
logL = partial(L,log = T)

ob$Fparam$L <- L
ob$Fparam$logL <- logL

ob = do.call(.infer$new,ob)


{
  make_dpos(ob)
  marg_obs(ob)
  make_post(ob)
  # ob

}
q2infer <- ob
```

3.1.3 Bayesian network inference

```
Rutil::install.packages.lazy(c('dagitty','pcalg','jpeg'))
library(igraph)

require(igraph)
require(Rutil)
source('dirichlet.R')

datadir = './assignment_1_files/'
load(file.path(datadir,"small_network_1.rda"),verbose = 1)
load(file.path(datadir,"small_network_2.rda"),verbose = 1)


#' @export
lp.diri <- function(n,eta,equiv = T){
  Sn = sum(n)
  Se = sum(eta)
  logP = lgamma(Se) - lgamma(Se + Sn) + sum(lgamma(n+eta) - lgamma(eta))
  if (equiv){
    ### apply equivalency correction (optionial)
    logP = logP + ndchoose(n,log = T)
  }
  logP
}


lp.nodes <- function(nodes,tb,eta=array(1,dim(tb)),...){
  tb.marg = margin.table(tb,margin = nodes)
  eta.marg = margin.table(eta,margin = nodes)
  lp = lp.diri(tb.marg,eta.marg,...)
}
source('net_post.R')
.PGM_binary <- setRefClass('PGM_binary',
                      fields = list(
                        # dat=c('array','data.frame'),
                        dat='data.frame',
                        tb ='table',
                        mdlgraph='ANY',
                        F = 'list'
                      ),methods = list(
                        make_table = function(){tb <<- table(dat)},
                        preprocess = function(){
                          g <- mdlgraph
                          pars = adjacent_vertices(g, V(g), mode = c("in"))
                          F$get_parent <<-  function(x)pars[[x]]
                        },
                        lp.node= function(.self,selfNode,eta0=1,
                                           eta= array(eta0,dim(.self$tb)),...){
                          sess = .self
                          parNode = sess$F$get_parent(selfNode)
                          nodes = c(selfNode,parNode)
                          lp.nodes(nodes,sess$tb,eta,...) - lp.nodes(parNode,sess$tb,eta,...)
                        },
                        logL = function(.self,...){
                          sess = .self
                          lps = sapply( V(sess$mdlgraph), partial(sess$lp.node,...))

                          sum(lps)
                        },
                        net_posterior = net_posterior
                      )
                    )


##### Testing the algorithm
source('graph_2edge.R')
sess = .PGM_binary$new(
  # dat=dat1,
  dat=dat2,
  mdlgraph=graph_from_adjacency_matrix(aM))
```

```
sess$make_table()
sess$preprocess()
sess$lp.node(1)%>%print
sess$logL()%>%print



##### My custom algorithm that computes the
##### likelihood of a dag on a dataset
myalgo <- function(g,dat,...){
  if (is(g,'matrix')){
    g <- graph_from_adjacency_matrix(g)
  }
  sess = .PGM_binary$new(
    dat=dat,
    mdlgraph=g)
  sess$make_table()
  sess$preprocess()
  sess$logL(...)
}

source('graph_2edge.R')

library(bnlearn)
#' Convert an adjacency matrix to a "bnlearn" object
#' @export
amat2bn <- function(aM,nodes = rownames(aM)){
  # require(bn)
  if (is.null(nodes)){
    nodes = LETTERS[1:nrow(aM)]
  }
  g0 <- bnlearn::empty.graph(nodes)
  bnlearn::amat(g0) <- aM
  g0
}


###### Load all graphs (Stored as adjacency matrix)
lst = list()
data.scripts = c('graph_0edge.R',
                 'graph_1edge.R',
                 'graph_2edge.R'
)
for (f in data.scripts){
  source(f)
  lst = c(lst,aMs)
}
all.graphs <- lst



#### Helper functions to compute data.frames
```

```
aM2bn <- function(aM,dat){
  eta0 = 1
  bn <- amat2bn(aM)
  bn$iss = 2^nrow(aM) * eta0
  bn$dat <- dat
  bn
}


{bn2df <- function(bn, iss=bn$iss){
  bnscore <- score(bn,bn$dat,'bde',iss=iss)
  bnscore <- score(bn,bn$dat,'bde',iss=iss)
  # bn
  myalgo.bde<- myalgo( amat(bn),bn$dat,eta0 = iss/8,equiv=F)
  list(model=modelstring(bn),myalgo.bde.iss=myalgo.bde,bnlearn.bde.iss=bnscore
       ,bnlearn.bic=score(bn,bn$dat,'bic')
       ,iss=iss
  )
}}


main <- function(dat = dat1,...){
  # bn.list <- lapply(lst,aM2bn)
  f = partial(aM2bn,dat=dat)
  bn.list <- lapply(lst,f)
  # bn.list <- mapply(aM2bn,lst,dat)
  g <- partial(bn2df,...)
  # df <- sapply(bn.list,g)
  df <- Rutil::combine_args(rbind)(lapply(bn.list,g))
  df <- Rutil::unlist.df(data.frame(df))
  df <- cbind(df,dat= deparse(substitute(dat)))
}
```

3.1.4 Construct posterior accroding to the graphical model

```
source('fitting_beta.R')
source('bayes_infer.R')
library(Rutil)

stratifier <- function(.self,selfNode,eta0=1,
                       eta = array(eta0,dim(.self$tb)), #### potentially caching eta
                       parent_state=NULL,
                       ...){
  sess <- .self
  parNode = as.numeric(sess$F$get_parent(selfNode))
  #### ignore parent_state if parNode is not matched
  parent_state <- parent_state[seq_along(parNode)]
  #### ignore parNode if parent is not specified
  parNode <- parNode[seq_along(parent_state)]
  nodes <- c(parNode,selfNode)
  idx = cbind(rbind(parent_state,parent_state),child_state=1:2)
  list(
```

```r
    tb.marg = margin.table(sess$tb,margin = nodes)[idx],
    eta.marg = margin.table(eta,margin = nodes)[idx])
}


logL_maker <- function(obsB,binom=T){
  #### Make a likelihood function based on a observation vector
  if(length(obsB)==2){
    N=sum(obsB)
    X=obsB[2]
  }else{
    N = length(obsB)
    X = sum(obsB)
  }
  if(binom){
    function(p1,...) {
      dbinom(x= X,size = N,p1,...)}
  }else{
    function(p1,...){
      p1^X*(1-p1)^(N-X)
    }
  }
}

net_posterior <- function(.self,
                          childNode,
                          parent_state=rep(1,length(.self$F$get_parent(childNode))) ,
                          eta0=1
                          ,debug=0
                          ,binom= F
                          ){
  sess <- .self
  out <- stratifier(sess,
                    childNode,
                    parent_state = parent_state)

  if(debug){print(out)}
  eta<- out$eta.marg

  ob <- .infer$new()
  ob$Fparam$prior <- partial(dbeta,shape1=eta[1],shape2=eta[2])
  L <-logL_maker(out$tb.marg,binom=binom)
  ob$Fparam$L <- L
  ob$Fparam$logL <- partial(L,log=T)

  {
    make_dpos(ob)
    marg_obs(ob)
    make_post(ob)
  }
  return(ob)
}
```

```
#### Examples
#
# aM <- all.graphs[[3]]
# sess <- .PGM_binary$new(mdlgraph=igraph::graph_from_adjacency_matrix(aM))
# sess$preprocess()
#
# {
#   sess$dat <-dat1
#   sess$make_table()
#   plot(sess$mdlgraph)
#   print(sess$tb)
#   ob <- sess$net_posterior(2,parent_state = 1,debug=1)
#   plot_bayes(ob)
#   title(bquote(beta(4,4)))
# }
#
# {
#   sess$dat <-dat2
#   sess$make_table()
#   print(sess$tb)
#   ob <- sess$net_posterior(2,parent_state = c(1),debug=1)
#   plot_bayes(ob)
#   title(bquote(beta(4,4)))
# }
```

3.1.4.1 Graphs

```
aMs = list()
{
  aM = cbind(c(0,0,0),
             c(0,0,0),
             c(0,0,0))
  ess = pcalg::dag2essgraph(aM)
  # check_aM(aM)
  aMs = c(aMs,list(aM))
}
```

```
aMs = list()
{
  aM = cbind(c(0,0,0),
             c(0,0,0),
             c(1,0,0))
  ess = pcalg::dag2essgraph(aM)
  # check_aM(aM)
  aMs = c(aMs,list(aM))
}

{
  aM = cbind(c(0,0,1),
             c(0,0,0),
             c(0,0,0))
  # check_aM(aM)
  aMs = c(aMs,list(aM))
}
```

```
aMs = list()
{
  aM = cbind(
    c(0,0,1),  #### A
    c(0,0,0),  #### B
    c(0,1,0))  #### C
  # check_aM(aM)
  aMs = c(aMs,list(aM))
}

{
  aM = cbind(c(0,0,0),
             c(0,0,1),
             c(1,0,0))
  # check_aM(aM)
  aMs = c(aMs,list(aM))
}

# layout_o
{
  aM = cbind(c(0,0,1),
             c(0,0,1),
             c(0,0,0))
  ess = pcalg::dag2essgraph(aM)
  # check_aM(aM)
  aMs = c(aMs,list(aM))
}

{
  aM = cbind(c(0,0,0),
             c(0,0,0),
             c(1,1,0))
  # check_aM(aM)
  aMs = c(aMs,list(aM))
}
```

3.1.5 Comparison of algortihms

3.1.5.1 Homemade xgboost-based regressional predictor (Gxgb)

```
library(xgboost)
library(Rutil)
library(dplyr)

##### Predict gene expression (predictee)
#### from all other genes (predictors)
##### 5-fold CV is ran to probe best parameter
####  to prevent overfitting
regGene <- function(expr.dat
  ,topred = 5
  ,nthread =1
  ,silent = 1
  ,objective ='reg:linear'
  # ,model.dir = 'model/'
  ,model.dir = NULL
  ,eta = 0.05
  ,max_depth = ncol(expr.dat)-1
  ,...
  ){
  genes<-colnames(expr.dat)
  gene<-genes[topred]
  dlabel = expr.dat[,topred]
  dtrain <- xgb.DMatrix(as.matrix(expr.dat[,-topred]),label=dlabel)

  param <- list(max_depth=max_depth,
                nthread = nthread
                ,objective=objective
                ,silent = silent
                ,eta = eta
            )
  param <- modifyList(param,list(...))
  res <- xgb.cv( param,dtrain,nrounds = 300
                 ,verbose = 1-silent
                 ,callbacks = list(cb.cv.predict(save_models = T),
                                   cb.early.stop(10,metric_name = 'test-rmse',
                                                 verbose = 1-silent)
                                   )
                 ,nfold= 5
                 )
  print(sprintf('Gene %s: \t best iteration:%d \t test_rmse_mean:%E'
                ,gene,res$best_iteration
        ,tail(res$evaluation_log$test_rmse_mean,1)))
  param$ntreelimit <- res$best_ntreelimit

  mdl = res$models[[1]]
  mdl = xgb.train(param,dtrain,nrounds=res$best_iteration)
  pd = predict(mdl,dtrain)
  xgb.attr(mdl,'gene')<-genes[topred]
  if (!is.null(model.dir)){
    fname = paste0(model.dir,genes[topred],'.xgb')
    xgb.save(fname = fname,model = mdl)
  }
  mdl
}


##### Convert a xgboost model to data.frame that
##### outlines importance of each predictor
model2df <- function(mdl){
  gene <- xgb.attr(mdl,'gene')
  print(gene)
  Feature= setdiff(genes,gene)
  # xgb.plot.tree(mdl)
  tryCatch({
    imat <- xgb.importance(model=mdl,feature_names = Feature)
  },
  error=function(e){
    # print(e)
    if((grepl('Non-tree model detected!',e))) {
      print('[WARN] Constant tree encountered')
      imat <- data.frame(Feature,Gain=0,Cover=0,Frequency=0)
    }else{
      stop(e)
    }
  }) -> imat
  imat <- cbind(output=gene,imat)

}

### load all models in the cache folder
load.folder <- function(model.dir,callback=xgb.load,...){
  fs <- list.files(model.dir,full.names = T,...)
  print(bquote('[]Loading'~.(length(fs))~objects))
  lapply(fs, callback)
}


### The wrapper function that predict adjacency from
### observed expression
Gxgb.fit <-function(dat,model.dir = NULL,post=T,silent=1,...){
  {
    # dat <- as.matrix(expr.dat)
    if(!is.null(model.dir)){
      dir.create(model.dir)
    }
    # browser()
    dat <- as.matrix(dat)
    genes<-colnames(dat)
    # expr.mat <- as.matrix(expr.dat)
    param <- list(
                 expr.dat=dat,
                 objective='reg:linear',
                 # objective='reg:logistic',
                 nthread=4,
                 eta=0.1,
                 max_depth= length(genes)-1,
                 alpha=0.25,lambda = 0.25,
                 model.dir = model.dir,
                 silent= silent
    )
    param <- modifyList(param,list(...))
    param$f = regGene
    f =  combine_args(partial)(param)
    mdls = lapply( seq_along(genes), f)
    if(post){
      df<-Gxgb.post(dat,mdls)
    }else{
      mdls
    }
  }
}


#### Post processing of the raw importance scores
Gxgb.post <- function(dat,mdls=NULL,model.dir=NULL){
  if (is.null(mdls)){
    mdls<-load.folder(model.dir,callback = xgb.load)
  }
  df = rbind_list(lapply(mdls,model2df))
  VAR = apply(dat,2,var)
  df$score <-  df$Gain * VAR[df$output] * VAR[df$Feature]
  df
}





##### Example
# if (interactive()){
#   {
#     load('assignment_1_files/medium_network.rda',verbose = T)
#     pc <- prcomp(t(expr.dat))
#     par(mfrow=c(1,2))
#     biplot(pc)
#     plot(apply(expr.dat,2,var)%>%sort)
#
#     # expr.dat <- as.matrix(expr.dat)
#     expr.dat.std <- apply(expr.dat,2,function(x)(x-mean(x))/sd(x) )
#     colnames(expr.dat.std) <- colnames(expr.dat)
#     # df <- Gxgb.fit(expr.dat.std)
#     df <- Gxgb.fit(expr.dat)
#     # df <- Gxgb.post(expr.mat,model.dir = 'model/')
#   }
#
#   load('assignment_1_files/medium_network_true.rda')
#   genes <- colnames(expr.dat)
#   adj.true <- pair2adj(true.pairs,genes = genes)
#   {
#     pairs = as.matrix(df[,c('Feature','output')])
#     res <- pair2adj(pairs,genes=genes,is.indexed = F,symmetric = F
#                     ,fill = df$score
#                     # ,fill = df$Gain
#     )
#     qc <- pipeline(res)
#   }
# }
```

3.1.5.2 Calculation of QC-metrics

```r
library(Rutil)
library(dplyr)
if (interactive()){
  try(setwd(dirname(sys.frame(1)$ofile)))
}
source('routine.R')
source('subnet.R')
# source('minfo.R')


##### initialise an adjacency matrix
init.amat <- function(index,fill=F){
  m<- matrix(fill,ncol=length(index),nrow =length(index) )
  NAME <- names(index)
  colnames(m) <- NAME
  rownames(m) <- NAME
  m
}

#### Helper function to symmetrise matrixs
as.symmetric <- function(mat){
  tmat =  t(mat)
  if (is.logical(mat)){
    mat = mat | tmat
  }else if(is.numeric(mat)){
    mat = (mat + tmat)/2
  }
  mat
}
upper.tri.get <- function(mat,...){
  mat[upper.tri(mat,...)]
}

normalise <- function(x){
  (x-mean(x))/sd(x)
}
renorm <- function(x){
  x/sum(x)
}

dictify<-function(entity){
  index <- setNames(seq_along(entity),entity)
}
dict.get <- function(x,index){index[x]}

#### Backend for extracting ROC-statistics
confmat2list <- function(tb){
  if( 'TRUE' %in% rownames(tb)){
    TP = tb['TRUE','TRUE']
    FP = tb['TRUE','FALSE']
  }else{
    TP=0
    FP=0
  }
  if( 'FALSE' %in% rownames(tb)){
    TN = tb['FALSE','FALSE']
    FN = tb['FALSE','TRUE']
  }else{
    TN = 0
    FN = 0
  }
  list(
    TP = TP,
    FP = FP,
    TN = TN,
    FN = FN
  )
}

#### Take two binary vector and return ROC-statistics
confusion.matrix <- function(pred,true,as.list = T){
  res <- cbind(
    pred,
    true
  )
  colnames(res) <- c('pred','gdtruth')
  tb <- table(as.data.frame(res))
  if(as.list){
    # confmat2list(tb)
    res <- confmat2list(tb)
    res <- within(res,{PR <-TP/(TP+FP);res} )
    res <- within(res,{RC <-TP/(TP+FN);res} )
    res <- within(res,{SP <-TN/(TN+FP);res} )
    res <- within(res,{NPN<-TN/(TN+FN);res} )
    res <- within(res,{F1 <-2/(1/PR+1/RC) })
    # res$tab <- tb
    res
  }else{
    tb
  }
}

##### Take an ordered T/F prediction to calculate
##### ROC statistics at different thresholds
confusion.matrix.from_pred <- function(tseq,as.list = T){
  TP <- cumsum(tseq)
  FP <- cumsum(1-tseq)
  P <- seq_along(tseq)
  P <- c(0,P)
  N <- rev(P)
  TP <- c(0,TP)
  FP <- c(0,FP)
  nT <- tail(TP,1)
  nF <- tail(FP,1)
  TN <- nF - FP
  data.frame(
  PR = TP/P,
  RC = TP/nT,
  SP = TN/nF,
  NPN= TN/N
  )
}




##### Convert a data.frame into an adjacency matrix
pair2adj <- function(pairs,as.list = F,genes=NULL,is.indexed=F,fill= T,symmetric = T){
  if(is.null(names(genes))) {
    index <- dictify(genes)
  }else{
    index <- genes
  }
  iadjmat <- init.amat(index)
  #### Maybe numeric avoids repeatedly hashing?
  indexer <- partial(dict.get,index=index)
  indexF <- function(x){
    x$index <- apply(x$pairs,c(2),indexer)
    x
  }
  v <- list(pairs=pairs)
  if (!is.indexed){
    v <- indexF(v)
  }else{
    v$index = v$pairs
  }
  v$adjmat <- iadjmat

  v$adjmat[v$index] <- fill
  if (symmetric){
    v$adjmat = as.symmetric(v$adjmat)
  }
  if(as.list){
    v
  }else{
    v$adjmat
  }
}

#### Take two data.frame and return the ROC-statistics
performance.pairs <- function(genes,
                              pred.pairs,
                              true.pairs){
  if(is.null(names(genes))) {
    index <- dictify(genes)
  }else{
    index <- genes
  }
  iadjmat <- init.amat(index)
  #### Maybe numeric avoids repeatedly hashing?
  indexer <- partial(dict.get,index=index)
  indexF <- function(x){
    x$index <- apply(x$mat,c(2),indexer)
    x
  }

  adjmats <- Map(partial(pair2adj,genes=genes),list(pred.pairs,true.pairs))


  #### play with the adjacency matrix
  res <- confusion.matrix(
    upper.tri.get(adjmats[[1]],diag= F)
    ,upper.tri.get(adjmats[[2]],diag= F)
    ,as.list= T
  )
  res
}




##### Benchmarking using an adjacency matrix
##### or data.frame that specifies (from,to,score)
pipeline <- function(res,npt=100,nan.fill=0,
                     dmat.prep=identity, adj.true=NULL, genes=NULL,
                     ...){
  if(is.null(adj.true)){
    adj.true <- parent.frame()$adj.true
  }
  if(is.null(genes)){
    genes <- parent.frame()$genes
  }
  if(is.matrix(res)){

    res[is.nan(res)]<-nan.fill
    res<-dmat.prep(res)
    ps <- reshape2::melt(res,value.name='score')
  }else if(is.data.frame(res)){
    ps<-res
    if(!'score'%in%names(ps)){
      names(ps)[3] <- 'score'
    }
    res <- pair2adj(as.matrix(ps[,c(1,2)]),is.indexed = is.numeric(ps[,'score']),
                    fill=ps[,3],genes=genes)
  }

  ps <- arrange(ps,desc(score))

  pd <- adj.true[as.matrix(ps[,1:2])]
  qc <- confusion.matrix.from_pred(pd)
  qc.meta <- list(qc.dat=qc,adjmat=res)
  qc <- post.qc(qc.meta,dmat.prep=identity,...)
  qc
}


#### Compute AUPR and AROC after extracting ROC-statistics
post.qc<-function(qc.meta,silent=F,method='linear',...){
  qc <- qc.meta$qc.dat
  tryCatch(
    {
      funcs <- list(
        AUPR=approxfun(qc$RC,qc$PR,method=method,rule = 2,yright = 1E-4,yleft = 1E-4),
        AROC=approxfun(1-unlist(qc$SP),qc$RC,method=method,rule = 2)
      )
      #### Code looks complicated because old
      #### pipeline causes RC to not fully cover [0,1]
      funcs$F1 <- function(RC) { PR = funcs$AUPR(RC);2/(1/RC+1/PR)}
      globals = list()
      RG=rev(1-range(qc$SP))
      globals = lapply(funcs[c('AROC')],
                       function(x) do.call(
                         integrate, c(f=x,as.list(RG),subdivisions=1000))$value)
      RG=range(qc$RC)
      globals = c(globals,
                  lapply(funcs[c('AUPR')],
                         function(x) do.call(
                           integrate, c(f=x,as.list(RG),subdivisions=1000))$value)
      )

      F1s <- 2/(1/qc$RC+1/qc$PR )
      Fidx <-which.max(F1s)
      # globals$F
      globals$F1 <- F1s[Fidx]
      coords = list(F1=c(qc$RC[Fidx], qc$PR[Fidx]) )

    },error=function(e){
      print("[ERR]fallback to basic plots")
      par(mfrow=c(1,2))
      within(qc,
             plot(RC,PR,type='b',ylim=c(0,1)))
      within(qc,
             plot(1-unlist(SP),RC,'b',cex=.8)

      )
      stop(e)
    }
  )

  qc.meta <- modifyList(qc.meta,
                        list(qc.dat=qc,globals=globals,coords=coords)
  )
  if (silent){
    # qc.meta
    ;
  }else{
```

```r
    diagnose(qc.meta,...)
  }
  qc.meta
}


#### Components of Old Benchmarking pipeline
#### Now replaced with confusion.matrix.from_pred()
thresholder <- function(mat,thres){
  as.symmetric(mat >= thres)
}

select.cutoff<-function(res,N=100){
  RG = range(res)
  lst = combine_args(seq)(c(as.list(RG),length.out =N%/%2))#[-1]
  lst <- c(lst, quantile(unlist(res),seq(-0,1,length.out = N%/%2)))
  lst <- c(lst,RG[1]-.1,RG[2]+.1)
  lst <- sort(lst)
}


#### Produce diagnostic plots
diagnose <- function(qc.meta,add.dmat=T,dmat.prep=log){
  # qc <- qc.meta$qcdat
  within(qc.meta,
         {
      if(add.dmat){
        # par(mfrow = c(1,3))
        image(dmat.prep(adjmat))

      }else{
        # par(mfrow = c(1,2))
        ;
      }

      within(qc.dat,
             plot(RC,PR,type='b',ylim=c(0,1),
                  xlab='Recall',
                  ylab='Precision'))
      # lines(qc$RC,funcs$AUPR(qc$RC))
      combine_args(points)(c(as.list(coords$F1),pch=2,col=2))
      print(coords$F1)
      # text(bquote'')
      text(coords$F1[1],coords$F1[2]
           ,labels=bquote(F1==.(round(globals$F1,2) ))
           ,adj = c(-.25,-.25),pch=2,col=2)
      title(bquote(atop(AUPR==.(globals$AUPR),
                 F1==.(globals$F1))))
      plot(1-unlist(qc.dat$SP),qc.dat$RC,'b',cex=.8
           ,ylab='True positive rate'
           ,xlab='False positive rate')
      abline(0,1,lty=2)
      title(bquote(AROC==.(globals$AROC)))
```

```
     }
  )

}



binarise.aggF <- function(df,aggF,MARGIN=2){
  #### Binarise a given network using some aggregation
  apply(df,MARGIN,function(x){as.factor(x>aggF(x)) })
}


load.assignment.data <- function()
{
  with(parent.frame(),
       {
         ##### Define shared data
         datadir <-'assignment_1_files/'
         load(file.path(datadir,"medium_network_true.rda"),verbose = F)
         load(file.path(datadir,"medium_network.rda"),verbose = F)

         head(true.pairs)
         head(expr.dat[,1:5])

         genes = colnames(expr.dat)
         ngene = length(genes)
         expr.dat.bin <- as.data.frame( binarise.aggF(expr.dat,median))
         adj.true <- pair2adj(true.pairs,genes = genes)
         g.true <- igraph::graph_from_adjacency_matrix(adj.true)
       })
}

if (interactive()){
  # datadir <-'assignment_1_files/'
  # load(file.path(datadir,"medium_network_true.rda"),verbose = F)
  # load(file.path(datadir,"medium_network.rda"),verbose = F)
  load.assignment.data()

  {
    genes <- colnames(expr.dat)
    index <- dictify(genes)
    set.seed(0)
    N = 20
    test.pairs <- matrix(sample(genes,size = N*2,replace = F),ncol=2)
    res <-perfromance.pairs(genes,test.pairs,true.pairs)
    # print(res)
    res
  }

  routine.bnlearn.bootstrap(expr.dat,cluster=clu)
  pipeline(minet::mrnet(minet::build.mim(expr.dat)))
  # par(mfrow=c(2,3))
  # res <-routine.xgb(expr.dat)
  # pipeline(res)
  gres <- routine.GENIE3(expr.dat,nCores = 12)
  pipeline(gres)



  {
    A={function(){outer(index,index,function(x,y) x*0)}}
    B={function(){outer(index,index,function(x,y) rep(T,length(x)))}}
    C=function(){
      m<- matrix(F,ncol=length(index),nrow =length(index) )
      NAME <- names(index)
      colnames(m) <- NAME
      rownames(m) <- NAME
      m
    }
    microbenchmark::microbenchmark(A(),B(),C(),times=1000)
  }

}
```

3.1.5.3 Routines for commonly used algorithms

```
routine.GENIE3<-function(dat,nCores=6,silent=1,...){
  require(GENIE3)
  require(doParallel)
  GENIE3(t(dat),nCores=nCores)
}

routine.bnlearn.bootstrap <- function(expr.dat,iss=1,R=200,score='bde',silent=0,...){
  require(bnlearn)

  expr.dat.bin <- bnlearn::discretize(as.data.frame(expr.dat),method = 'quantile',breaks=3)
  res.bnlearn <- boot.strength((expr.dat.bin),algorithm = 'hc',algorithm.args=list(
    score=score
    ,iss=iss),
    cpdag = T,R=R,...)
  res.bnlearn <- arrange(res.bnlearn,desc(strength))
}



routine.xgb <- function(expr.dat,...){

  source('xgb.R')
  df <- Gxgb.fit(expr.dat,model.dir = 'qc/',silent=1)
  pairs = as.matrix(df[,c('Feature','output')])
  VAR <- apply(expr.dat,2,var)
  res <- pair2adj(pairs,genes=genes,is.indexed = F,symmetric = F,
                  fill = df$Gain*VAR[df$output]*VAR[df$Feature])
  qc <- pipeline(res,...)
  qc$method <-'xgb'
}

routine.mrnet <- function(expr.dat){
  minet::mrnet(minet::build.mim(expr.dat))
}

if(interactive()){
  PKGMethod='MI.inverted'
  fig.cap=PKGMethod
  PKG = strsplit(PKGMethod,'\\.')[[1]][1]
  # library(PKG,character.only = T)
  res <- mat.invert(make.mi(expr.dat),post=abs)
  qc <- pipeline(res,silent=1)
  qc$method =  PKGMethod
  qc.list[[PKGMethod]] <- qc

  PKGMethod='MI.raw'
  fig.cap=PKGMethod
  PKG = strsplit(PKGMethod,'\\.')[[1]][1]
  # library(PKG,character.only = T)
  res <- (make.mi(expr.dat))
  qc <- pipeline(res,silent=1)
  qc$method =  PKGMethod
  qc.list[[PKGMethod]] <- qc


  PKGMethod='COV.inverted'
  fig.cap=PKGMethod
  PKG = strsplit(PKGMethod,'\\.')[[1]][1]
  # library(PKG,character.only = T)
  res <- mat.invert(cov(expr.dat),post = abs)
  qc <- pipeline(res,silent=1)
  qc$method =  PKGMethod
  qc.list[[PKGMethod]] <- qc

}
```

3.1.5.4 Bootstrapping on sub-networks

```
#### Generate a subnet
#### from a list, return a list
subnet <- function(env.data,ntrial=10,ngene=10,...){
  if(is.null(env.data$gmat)){
    env.data$gmat = gmat_from_adjacency(env.data$adj.true)
  }
  with(env.data,
       {
         for (i in 1:ntrial){
           gis <- unique(markov(gmat,...))
           if(length(gis)>ngene){gis=head(gis,ngene);break}
         }

         list(
           genes = genes[gis],
           adj.true=adj.true[gis,gis],
           expr.dat = expr.dat[,gis],
           g.true = graph_from_adjacency_matrix(adj.true[gis,gis])
         )
       }
  )
}


#### Markov chain sampler
#### Perform random walk on a graph
gmat_from_adjacency <- function(amat){
  if (is(amat,'igraph')){
    amat <- as.symmetric(as.matrix(igraph::as_adjacency_matrix(amat)))
  }
  pmat <- apply(amat,2,renorm)
  generator.mat <- apply(pmat,2,cumsum)
}
lookup<-function(r,cur,gen.mat){
  which(r<gen.mat[,cur])[1]
}
markov <- function(generator.mat,init.prob=rep(1,nrow(generator.mat)),nT=100){
  init =sample(seq_along(init.prob),prob = init.prob,1)
  rand <- runif(nT)
  out <- vector(length=nT)
  cur <- init
  for (i in seq_along(rand) ){
    out[i]=cur
    r = rand[i]
    cur <- lookup(r,cur,generator.mat)
  }
  out
}



#### Runing bnlearn and GENIE3 side by side
contraster <- function(){
  env.sub <- subnet(env.data,nT=50)
  if(length(env.sub$genes)<5){return(list())}
  with(env.sub,
       {
         print(length(genes))
         t1 <- system.time({res <- routine.bnlearn.bootstrap(expr.dat,clu=clu)})
         r1 <- pipeline(res,adj.true = adj.true,silent=1)
         t2 <- system.time({res <- routine.GENIE3(expr.dat,nCores=16)})
         r2 <- pipeline(res,adj.true = adj.true,silent=1)
         r1$globals$runtime <- t1[3]
         r1$globals$method  <- 'bnlearn'
         r2$globals$runtime <- t2[3]
         r2$globals$method  <- 'GENIE3'
         d <-rbind(r1$globals,r2$globals)
         d <-cbind(d,ngene =length(genes))
         d
       }) ->out
}



plot.lst<-function(lst){
  require(ggplot2)
  df <- as.data.frame(lst)
  df <- unlist.df(df)
  df$i <- ceiling(1:nrow(df)/length(unique(df$method)))

  p1 <- ggplot(df) + geom_histogram(aes(x=AUPR,fill=method,y=..count..),position = "dodge",alpha=0.25) +
    geom_density(aes(x=AUPR,color=method,y=..density..)) + xlim(0,1) + theme(legend.position = 'top')

  # ggplot(df) + geom_density(aes(x=AUPR,color=method,y=..count..))
  df.melt <- reshape2::melt(df,id.vars=c('i','method'),measure.vars=c('AUPR'),
                            value.name='AUPR')
  val1 <- filter(df,method=='bnlearn')
  val2 <- filter(df,method=='GENIE3')
  p2 <- ggplot()+geom_point(data=data.frame(bnlearn.AUPR=val1$AUPR,GENIE3.AUPR=val2$AUPR),
                            aes(x=bnlearn.AUPR,y=GENIE3.AUPR)) +
    geom_abline(intercept = 0,slope=1) +xlim(0,1)+ylim(0,1) +
    ggtitle(bquote(atop(
      bnlearn.avg_runtime==.(mean(val1$runtime))~s,
      GENIE3.avg_runtime==.(mean(val2$runtime))~s
    )
    ))
  p <- gridExtra::grid.arrange(p1,p2,ncol=2)
}



### (Deprecated) Old heuristic random walk on a graph
### non-markovian
randomWalk <- function(links,size= 50,directed=F)
{
  if (is.matrix(links)){
    nodes <- colnames(links)
    amat <- links
  }else{
    nodes <- unique(unlist(links))
  }
  init <- sample(which(rowSums(amat)!=0,1),1)
  # cur<-nodes[init
  idxseq <- sample(seq_along(nodes),size=size-1,replace = T)
  idxseq <- c(idxseq)
  outseq <- c(init,idxseq)
  idx <- init
  for (i in seq_along(idxseq)){
    outseq[i+1] <- idx
    if(directed){
      pointer = amat[idx,]
    }else{
      pointer = amat[idx,] | amat[,idx]
    }
    idx <- which(pointer)[ idxseq[i]%%sum(pointer)+1 ]
  }
  nodes[unique(outseq)]
}
```

3.1.5.4.1 Working script for GNW dataset

```
load.assignment.data()
env.data <- list(adj.true = adj.true,expr.dat=expr.dat,genes=genes,
                 gmat = gmat_from_adjacency(adj.true))
out <- replicate(50,contraster(),simplify = F)
lst <- combine_args(rbind)(out)
save(lst, env.data, file='assignment-subnet.rdf5')
```

3.1.5.4.2 Working script for DREAM5 dataset, network 3

```
{
  fname = 'dream5/training data/Network 3 - E. coli/net3_expression_data.tsv'
  f <- readLines(fname)
  dat <- read.csv(fname,sep='\t')
  expr.dat.big <- as.matrix(dat)


  fname <- 'dream5/test data/DREAM5_NetworkInference_GoldStandard_Network3 - E. coli.tsv'
  dat <- read.csv(fname,sep='\t')
  true.pairs.big<-dat[dat[,3]==1,1:2]
  genes.big <- colnames(expr.dat.big)
  adj.true.big <- pair2adj(true.pairs.big ,genes = genes.big)
}

env.data <- list(adj.true = adj.true.big,expr.dat=expr.dat.big,genes=genes.big,
                 gmat = gmat_from_adjacency(adj.true.big))

out <- replicate(50,contraster(),simplify = F)
lst <- combine_args(rbind)(out)
save(lst, env.data, file='e-coli-subnet.rdf5')
```