R-EndSem

May 29, 2020

I have referred the official bnlearn documentation for implementing the Bayesian network.

https://www.bnlearn.com/
https://www.bnlearn.com/examples/useR19-tutorial/
https://www.bnlearn.com/examples/custom/

[59]:
```
# install.packages("BiocManager")
# BiocManager::install("Rgraphviz")
# install.packages("bnlearn")
# install.packages("ggplots")
# install.packages("GGally")
# install.packages("polycor")
# install.packages("dplyr")
```

```
Installing package into '/home/maddy/R/x86_64-pc-linux-gnu-library/3.6'
(as 'lib' is unspecified)

also installing the dependencies 'purrr', 'tidyselect', 'BH', 'plogr'
```

[61]:
```
library(bnlearn)
library(gplots)
library(GGally)
library(polycor)
library(dplyr)
```

[62]:
```
col_names =
 ↪"age,sex,cp,trestbps,chol,fbs,restecg,thalach,exang,oldpeak,slope,ca,thal,num"
col_names = strsplit(col_names, ",")
col_names = as.vector(col_names)
col_names = unlist(col_names)
```

[64]:
```
test_data = sample_n(data, 50)
```

[3]:
```
data = read.csv("processed.cleveland.data",header = FALSE,sep = ",", col.names
 ↪= col_names)
```

[4]:
```
my_col_names = "age,cp,trestbps,chol,thalach,exang,oldpeak,ca,thal,num"
my_col_names = strsplit(my_col_names, ",")
my_col_names = as.vector(my_col_names)
my_col_names = unlist(my_col_names)
```

Randomly selected attributes

[5]:
```
my_col_names
```

1. 'age' 2. 'cp' 3. 'trestbps' 4. 'chol' 5. 'thalach' 6. 'exang' 7. 'oldpeak' 8. 'ca' 9. 'thal' 10. 'num'

[6]:
```
data = data[,my_col_names]
```

[7]:
```
dim(data)

head(data)
```

1. 303 2. 10

A data.frame: 6 × 10

	age <dbl>	cp <dbl>	trestbps <dbl>	chol <dbl>	thalach <dbl>	exang <dbl>	oldpeak <dbl>	ca <dbl>	thal <dbl>	n <
1	63	1	145	233	150	0	2.3	0	6	0
2	67	4	160	286	108	1	1.5	3	3	2
3	67	4	120	229	129	1	2.6	2	7	1
4	37	3	130	250	187	0	3.5	0	3	0
5	41	2	130	204	172	0	1.4	0	3	0
6	56	2	120	236	178	0	0.8	0	3	0

[8]:
```
my_col_names = "age,cp,trestbps,chol,thalach,exang,oldpeak,ca,thal,num"
my_col_names = strsplit(my_col_names, ",")
my_col_names = as.vector(my_col_names)
my_col_names = unlist(my_col_names)


continous_col_names = "age,trestbps,chol,thalach,oldpeak"
continous_col_names = strsplit(continous_col_names, ",")
continous_col_names = as.vector(continous_col_names)
continous_col_names = unlist(continous_col_names)


discrete_col_names = "cp,exang,ca,thal,num"
discrete_col_names = strsplit(discrete_col_names, ",")
discrete_col_names = as.vector(discrete_col_names)
discrete_col_names = unlist(discrete_col_names)

x_col_names = "age,cp,trestbps,chol,thalach,exang,oldpeak"
x_col_names = strsplit(x_col_names, ",")
x_col_names = as.vector(x_col_names)
```

```
x_col_names = unlist(x_col_names)
```

[9]:
```
data[,"num"][data[,"num"] > 0]  = 1
```

[10]:
```
data[,discrete_col_names] <- lapply(data[,discrete_col_names], as.factor)
data[,continous_col_names] <- lapply(data[,continous_col_names], as.numeric)
```

[11]:
```
head(data)
```

A data.frame: 6 × 10

	age <dbl>	cp <fct>	trestbps <dbl>	chol <dbl>	thalach <dbl>	exang <fct>	oldpeak <dbl>	ca <fct>	thal <fct>	nur <fc
1	63	1	145	233	150	0	2.3	0	6	0
2	67	4	160	286	108	1	1.5	3	3	1
3	67	4	120	229	129	1	2.6	2	7	1
4	37	3	130	250	187	0	3.5	0	3	0
5	41	2	130	204	172	0	1.4	0	3	0
6	56	2	120	236	178	0	0.8	0	3	0

0.0.1 Dataset understanding

There is no missing value in the dataframe which is evident from -

[12]:
```
sum(is.na(data))
```

6

For all the continous variables, we do the histogram plot to see if they are Gaussian

[13]:
```
head(data[,continous_col_names])
dim(data[,continous_col_names])
```

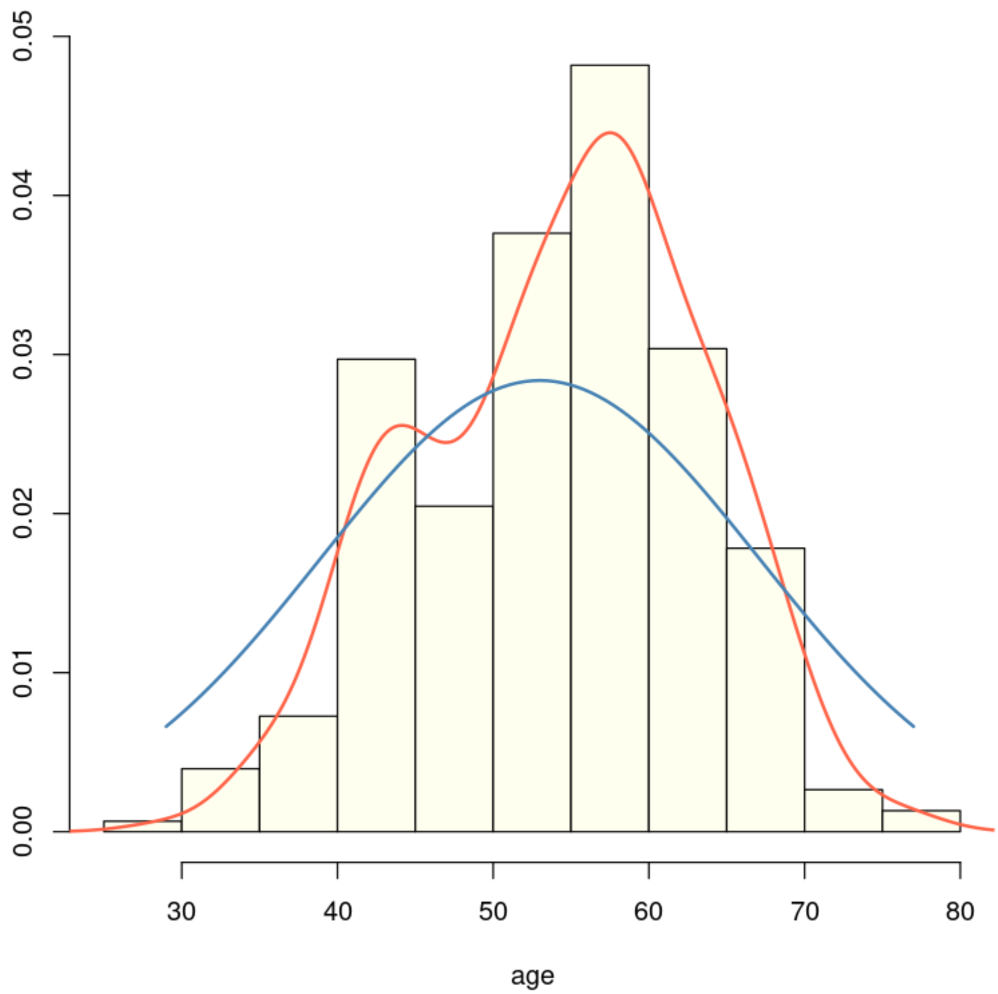
A data.frame: 6 × 5

	age <dbl>	trestbps <dbl>	chol <dbl>	thalach <dbl>	oldpeak <dbl>
1	63	145	233	150	2.3
2	67	160	286	108	1.5
3	67	120	229	129	2.6
4	37	130	250	187	3.5
5	41	130	204	172	1.4
6	56	120	236	178	0.8

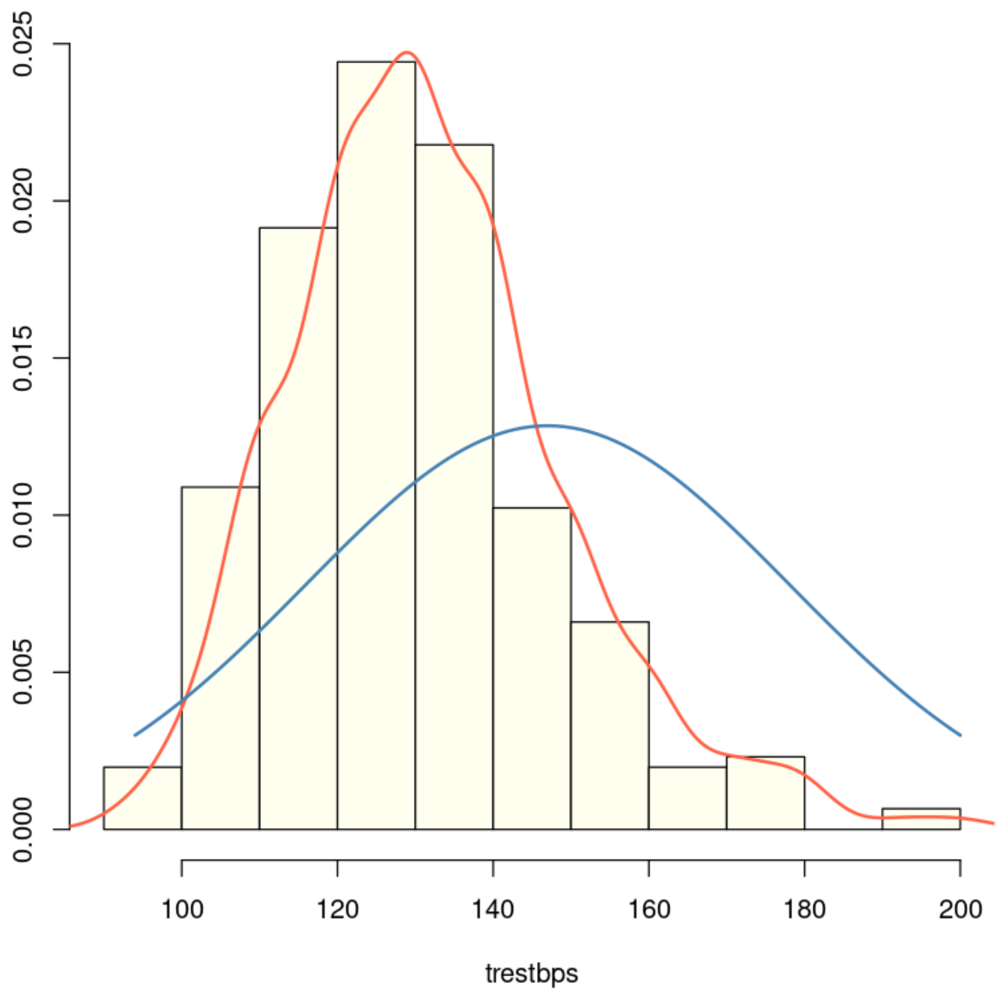
1. 303 2. 5

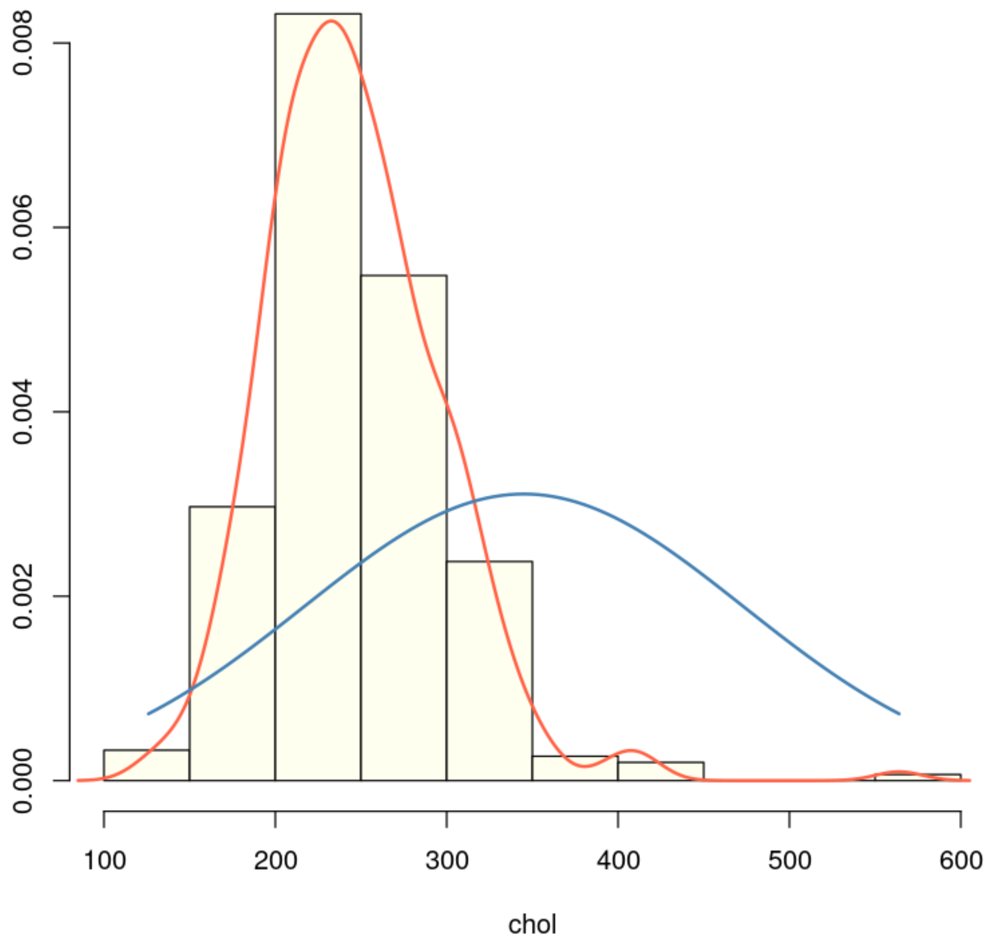
[14]:
```
for (var in continous_col_names) {

    x = data[, var]
    hist(x, prob = TRUE, xlab = var, ylab = "", main = "", col = "ivory")
    lines(density(x), lwd = 2, col = "tomato")
    curve(dnorm(x, mean = mean(x), sd = sd(x)), from = min(x), to = max(x),add␣
 ↪= TRUE, lwd = 2, col = "steelblue")
```

```
}
```

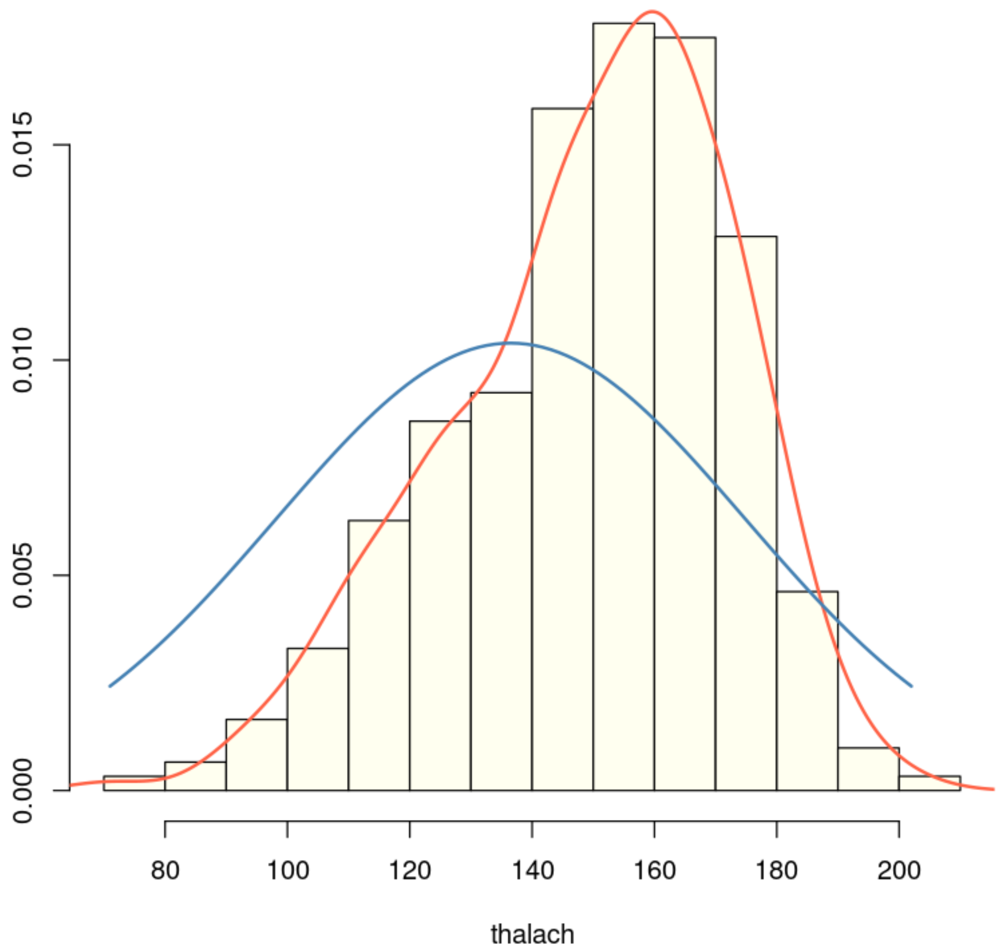


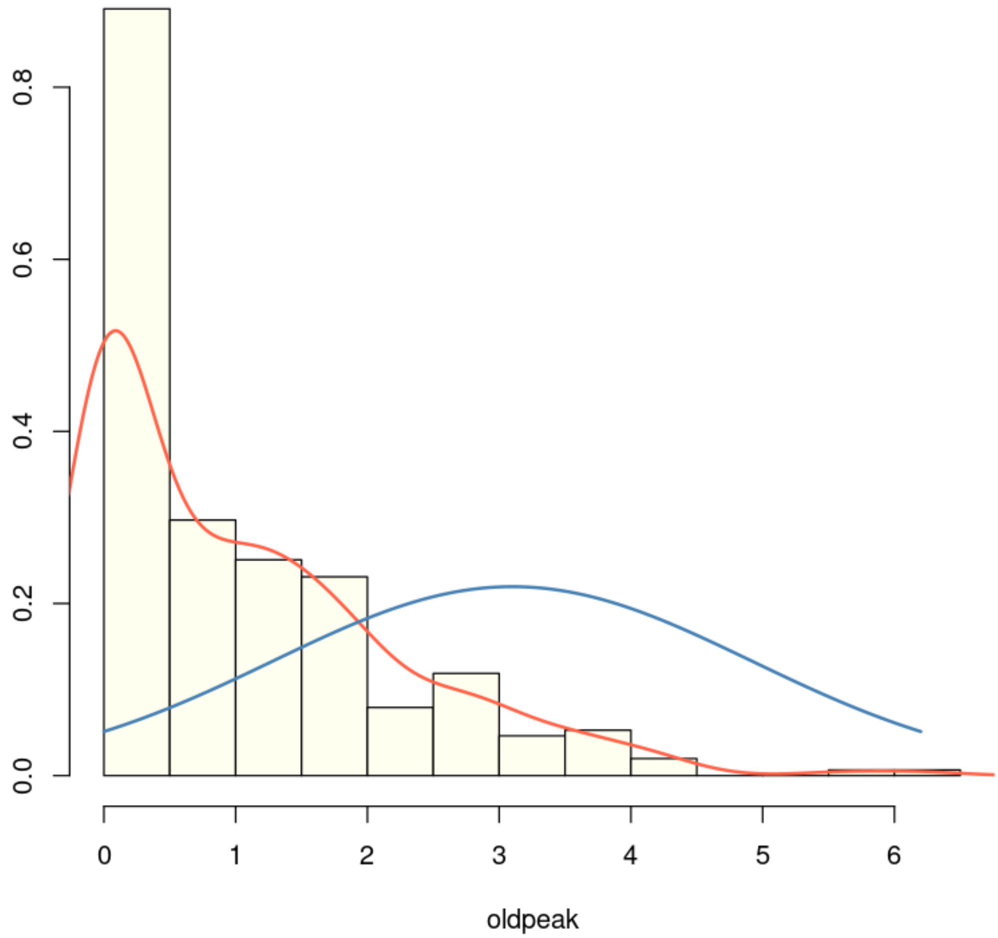
0.00
0.01
0.02
0.03
0.04
0.05
30
40
50
60
70
80
age


0.000
0.005
0.010
0.015
0.020
0.025
100
120
140
160
180
200
trestbps



0.000
0.002
0.004
0.006
0.008
100
200
300
400
500
600
chol


0.000
0.005
0.010
0.015
80
100
120
140
160
180
200
thalach



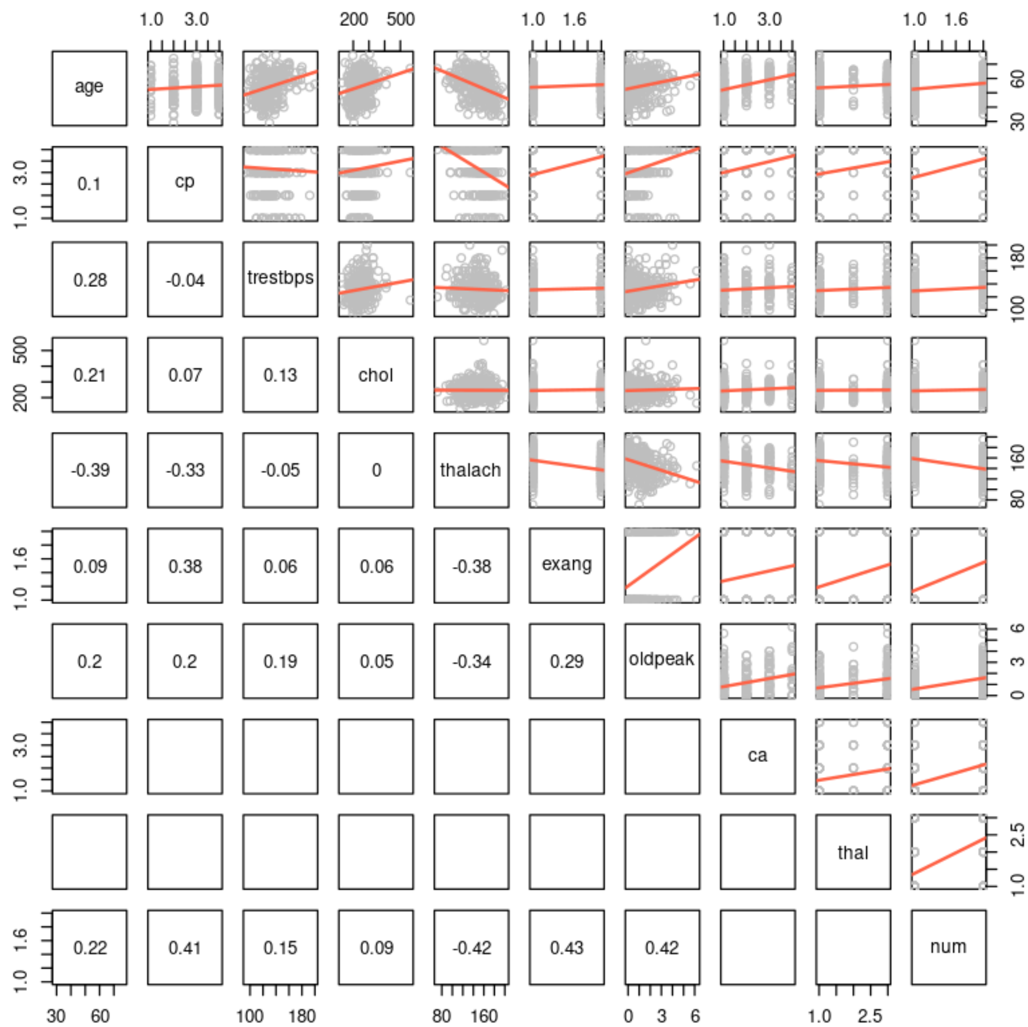
Most of the continous attributes are close to a Gaussian distribution, but some e.g. oldpeak are not.

We will deal with that later.

Checking relationship between all the feature-pairs

[15]:
```
pairs(data,
upper.panel = function(x, y, ...) {
 points(x = x, y = y, col = "grey")
 abline(coef(lm(y ~ x)), col = "tomato", lwd = 2)
},
lower.panel = function(x, y, ...) {
 par(usr = c(0, 1, 0, 1))
```

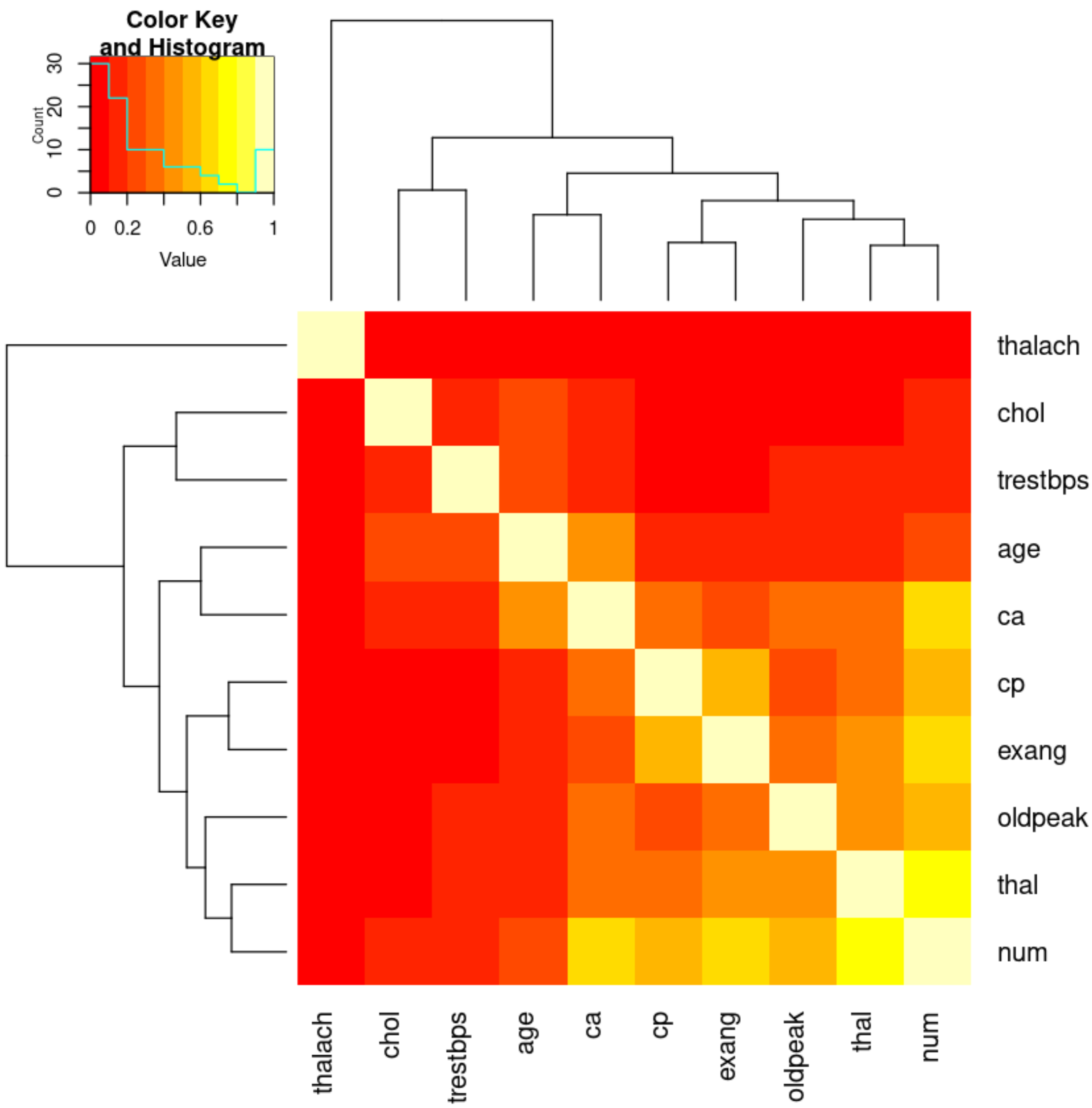
```
 text(x = 0.5, y = 0.5, round(cor(x, y), 2), cex = 1)
}
)
```



Checking correlations between features to get a raw idea of the bayesian-network

[16]:
```
correlations_all = hetcor(data,std.err = FALSE)$correlations
```

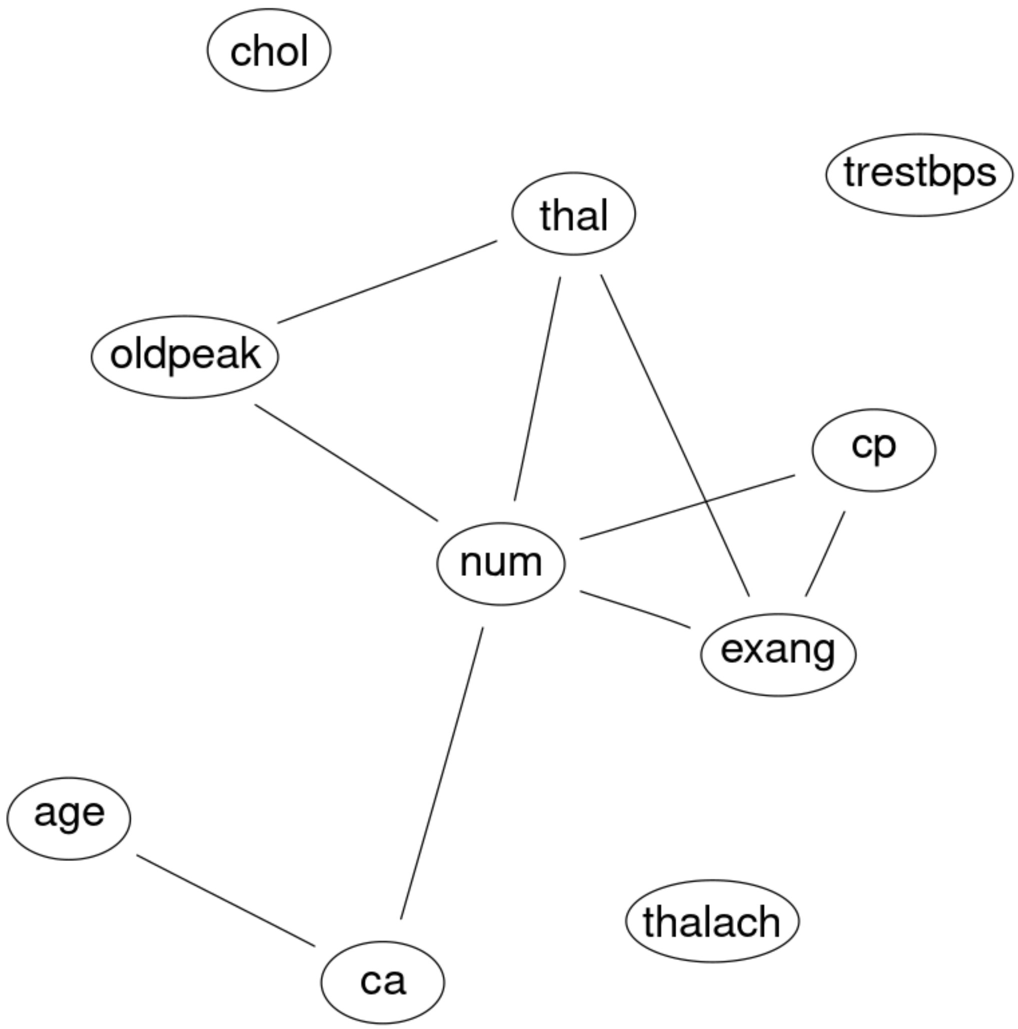
[17]:
```
heatmap.2(correlations_all,scale = "none", trace = "none", revC = TRUE, breaks␣
↪= seq(0, 1, 0.1))
```



There is definitely a cluster of variables as seen from the heatmap, same is visualized in the graph below.

[18]:
```
ug = empty.graph(colnames(correlations_all))
amat(ug) = (correlations_all > 0.4) + 0L - diag(1L, nrow(correlations_all))
graphviz.plot(ug, layout = "fdp", shape = "ellipse")
```

```
Loading required namespace: Rgraphviz
```



0.0.2 Structure Learning

We will learn the structure of the Bayesian Network before learning the parameters Using some information from https://archive.ics.uci.edu/ml/datasets/Heart+Disease one can say that these features should not change when one is changed, so adding a black-list in the network corresponding to these attributes.

[19]:
```
bl = tiers2blacklist(list("exang",c("chol", "trestbps")))
bl = rbind(bl, c("exang", "chol"), c("chol", "exang"))
```

[20]:
```
bl
```

A matrix: 4 × 2 of type chr

from	to
chol	exang
trestbps	exang
exang	chol
chol	exang

Similarly, adding a white-list for the attributes.

[21]:
```
wl = matrix(c("thalach", "trestbps"),ncol = 2, byrow = TRUE, dimnames =
 ↪list(NULL, c("from", "to")))
```

[22]:
```
wl
```

A matrix: 1 × 2 of type chr

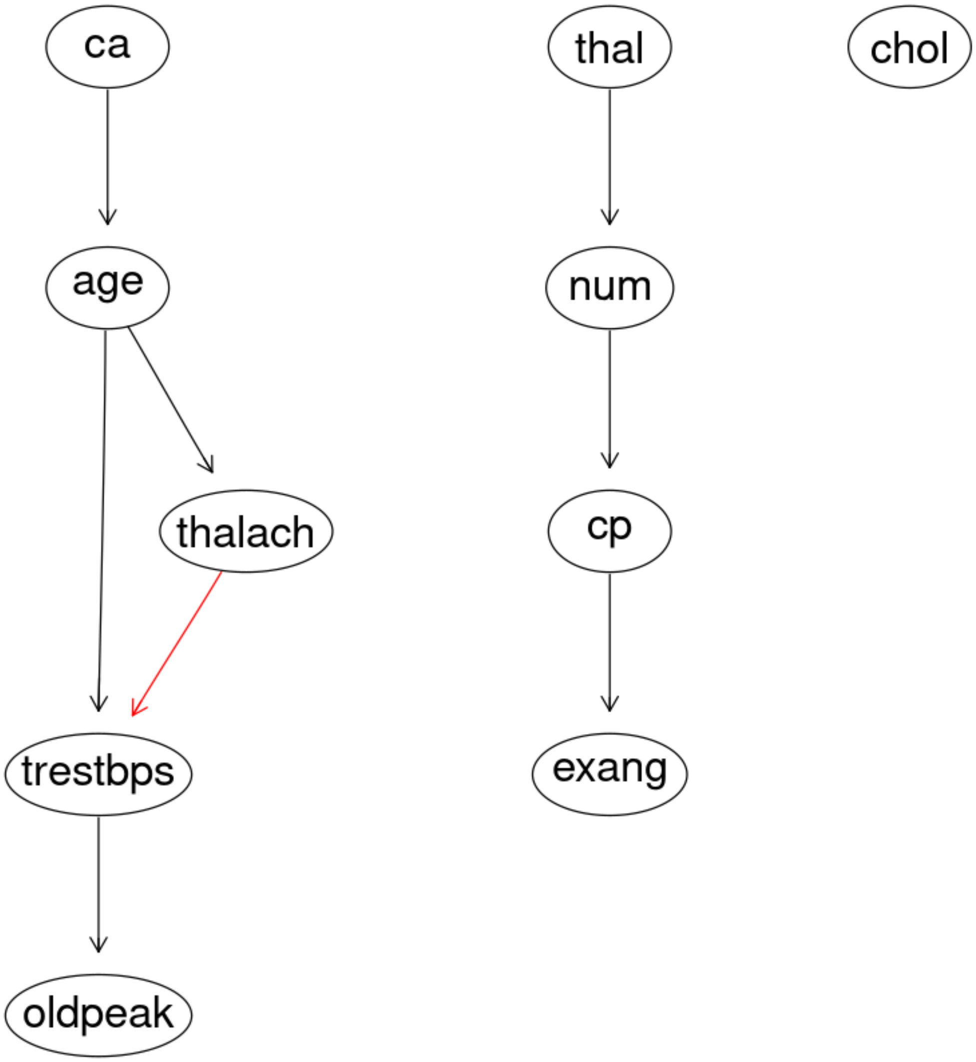
from	to
thalach	trestbps

[23]:
```
x_data = data[,x_col_names]
```

[24]:
```
data_no_na = data[complete.cases(data), ]
```

[25]:
```
dag = mmhc(data_no_na, whitelist = wl, blacklist = bl)
```

[26]:
```
graphviz.plot(dag, shape = "ellipse", highlight = list(arcs = wl))
```



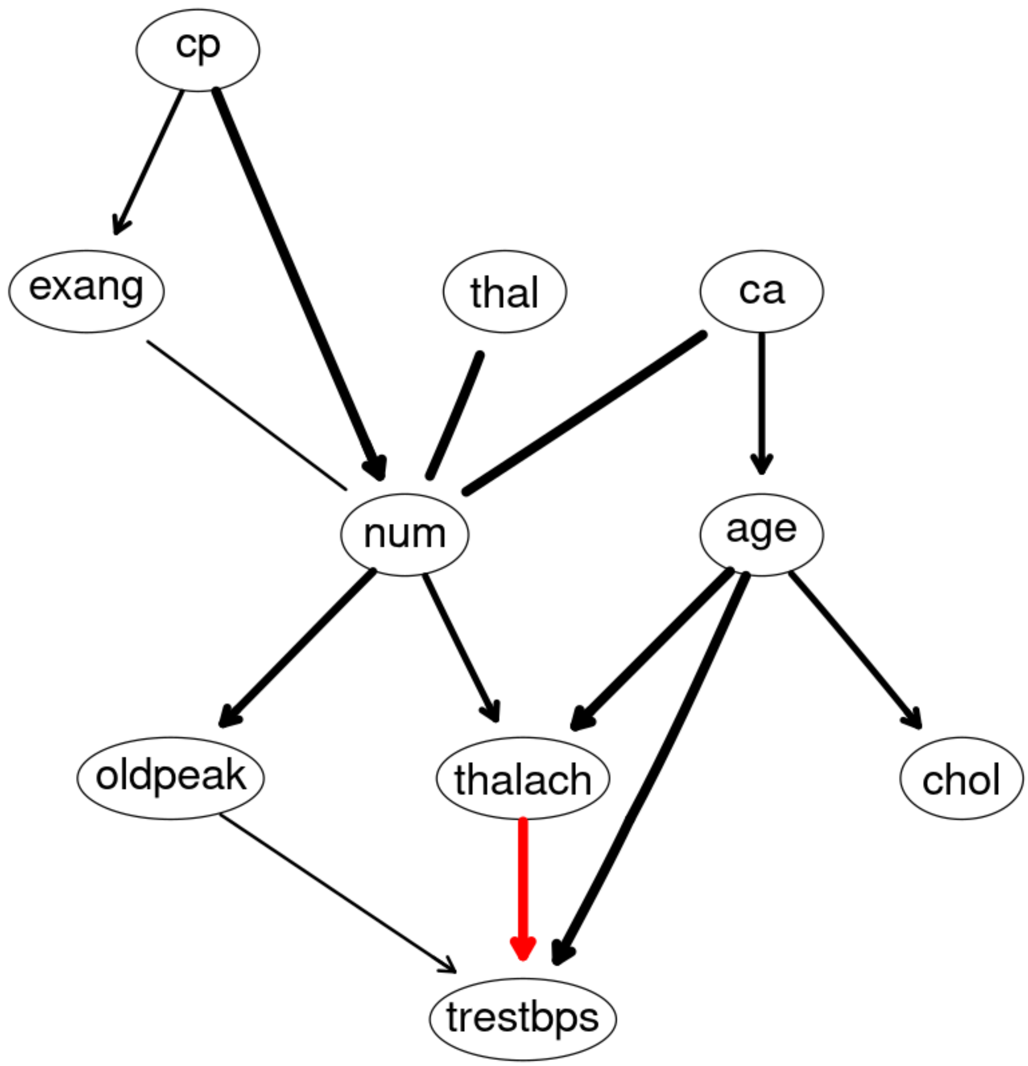
As we saw, above that some of the continous attributes do not strictly follow the Gaussian distribution, so we use boot.strength to resample data using bootstrapping and to get a better respresentation of the bayesian-network.

[27]:
```
strength = boot.strength(data_no_na, R = 200, algorithm = "hc",
                         algorithm.args = list(whitelist = wl, blacklist = bl))
```

[28]:
```
avg = averaged.network(strength)
```

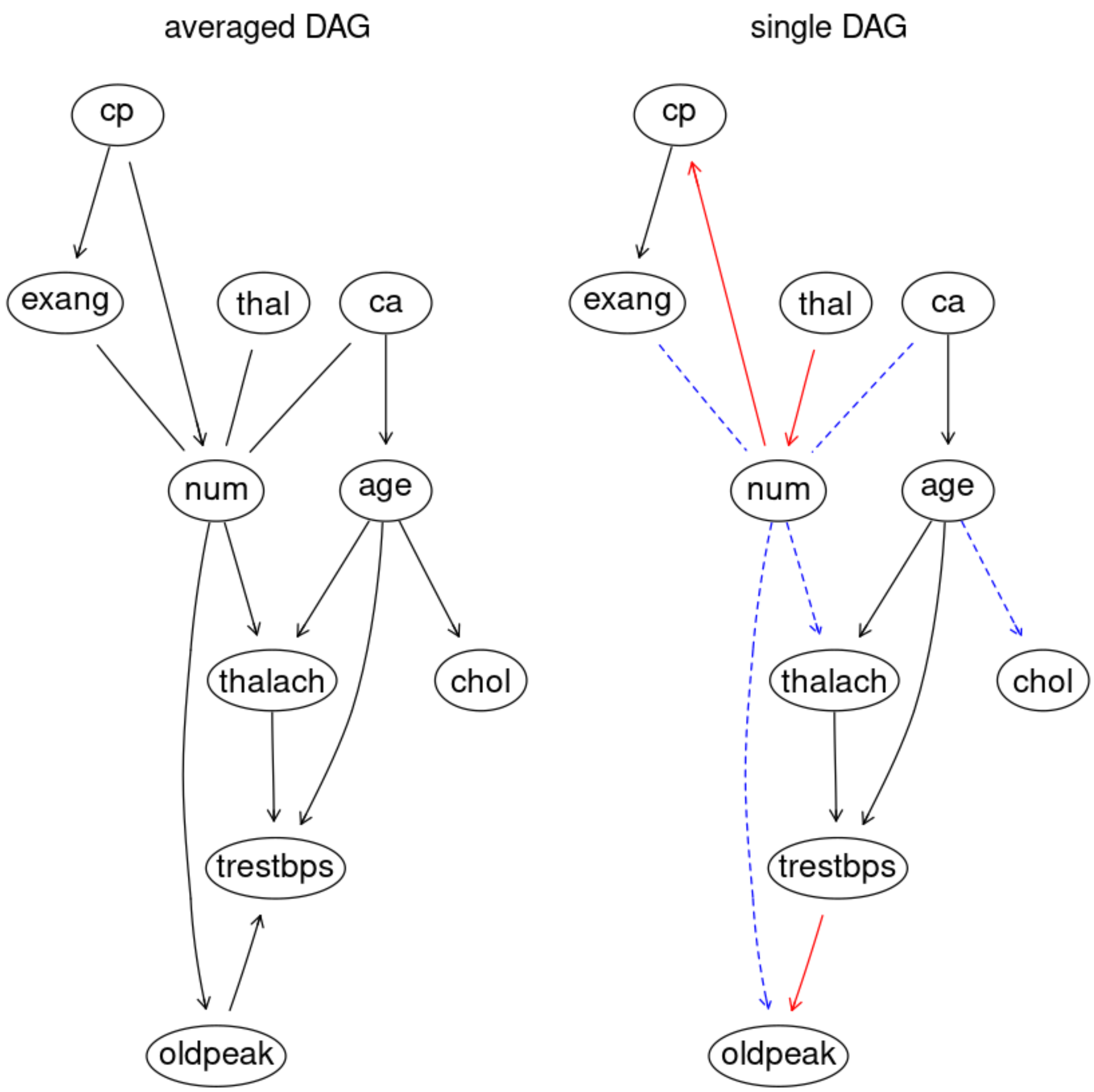
```
Warning message in averaged.network.backend(strength = strength, nodes = nodes,
:
"arc num -> cp would introduce cycles in the graph, ignoring."
```

[29]:
```
strength.plot(avg, strength, shape = "ellipse", highlight = list(arcs = wl))
```



We try to compare both the networks - learned from data and learned using boot.strength

[30]:
```
par(mfrow = c(1, 2))
graphviz.compare(avg, dag, shape = "ellipse", main = c("averaged DAG", "single␣
 ↪DAG"))
```



[31]:
```
compare(avg, dag)
```

$tp 5

$fp 3

$fn 8

[32]:
```
compare(cpdag(avg, wlbl = TRUE), cpdag(dag, wlbl = TRUE))
```

$tp 7

$fp 1

$fn 6

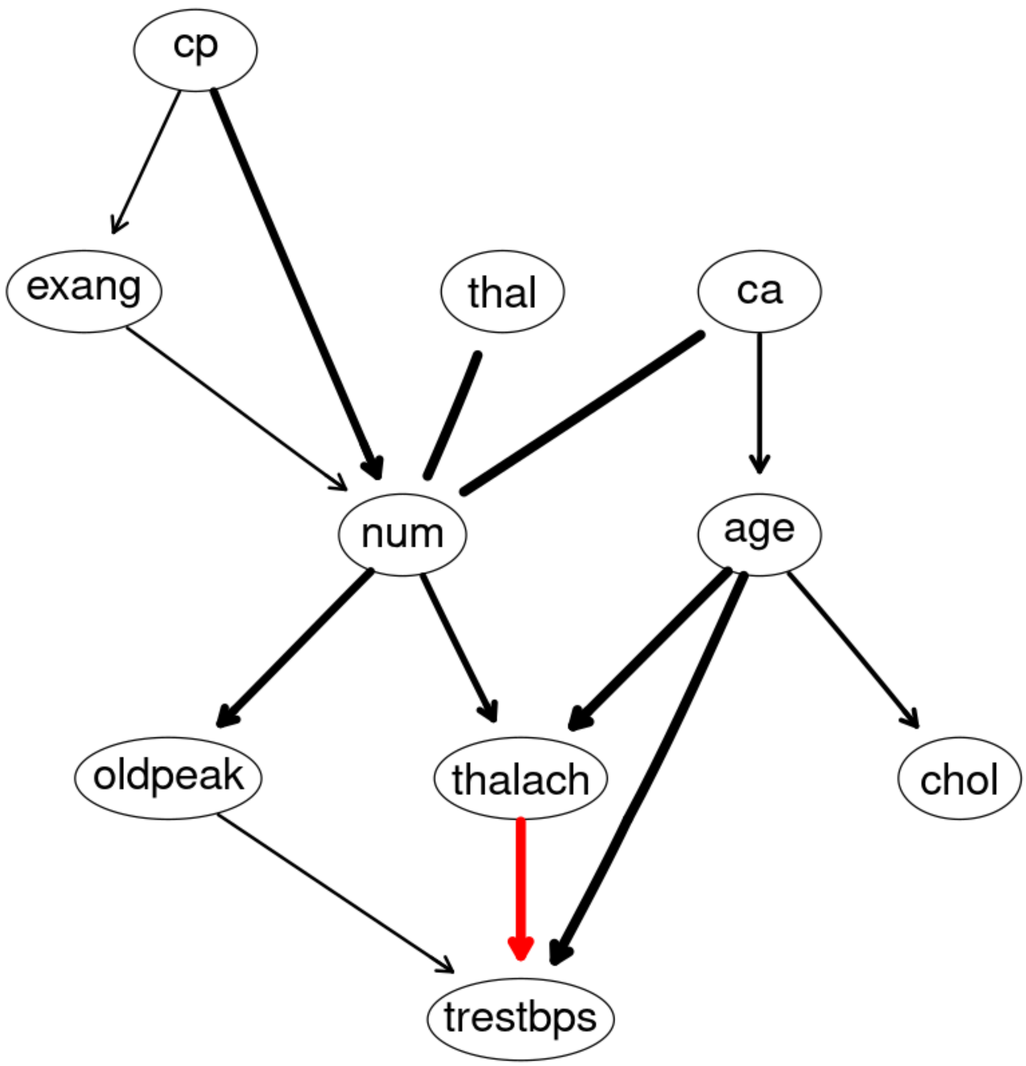
[33]:
```r
nrow(strength[strength$strength > 0.5 & strength$direction > 0.5, ])
```

9

[34]:
```r
simpler = averaged.network(strength, threshold = 0.6)
```

```
Warning message in averaged.network.backend(strength = strength, nodes = nodes,
:
"arc num -> cp would introduce cycles in the graph, ignoring."
```

[44]:
```r
strength.plot(simpler, strength, shape = "ellipse", highlight = list(arcs =
 ↪wl), threshold = 0.6)
```



[47]:
```
undirected.arcs(simpler)
```

A matrix: 4 × 2 of type chr

from	to
ca	num
thal	num
num	ca
num	thal

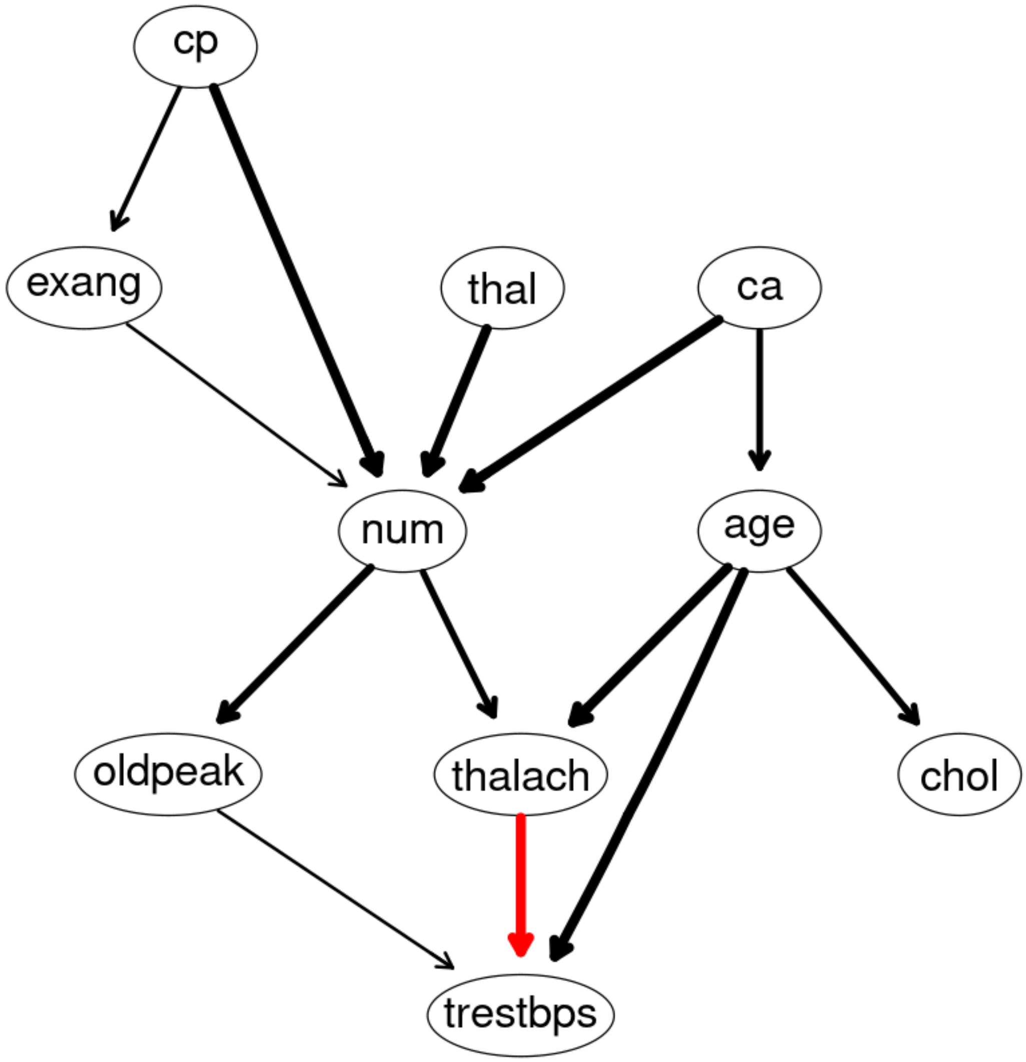
[48]:
```
simpler = set.arc(simpler, from="ca", to="num")
simpler = set.arc(simpler, from="thal", to="num")
# simpler = set.arc(simpler, from="thal", to="num")
# simpler = set.arc(simpler, from="cp", to="num")
```

[38]:
```
continous_col_names
discrete_col_names
```

1. 'age' 2. 'trestbps' 3. 'chol' 4. 'thalach' 5. 'oldpeak'

1. 'cp' 2. 'exang' 3. 'ca' 4. 'thal' 5. 'num'

[56]:
```
strength.plot(simpler, strength, shape = "ellipse", highlight = list(arcs = wl))
```



0.0.3 Parameter Learning

[149]:
```
fitted
```

```
  Bayesian network parameters

  Parameters of node age (conditional Gaussian distribution)

Conditional density: age | ca
Coefficients:
                    0          1          2          3
(Intercept)  51.68391  57.47692  59.78947  59.90000
Standard deviation of the residuals:
```

```
      0        1        2        3
9.226005  6.876381  6.576608  8.232797
Discrete parents' configurations:
   ca
0   0
1   1
2   2
3   3

  Parameters of node cp (multinomial distribution)

Conditional probability table:
         1          2          3          4
0.07744108 0.16498316 0.27946128 0.47811448

  Parameters of node trestbps (Gaussian distribution)

Conditional density: trestbps | age + thalach + oldpeak
Coefficients:
(Intercept)          age      thalach      oldpeak
 80.2769148    0.6055481    0.1038890    2.6972546
Standard deviation of the residuals: 16.77764

  Parameters of node chol (Gaussian distribution)

Conditional density: chol | age
Coefficients:
(Intercept)          age
 183.844602     1.164341
Standard deviation of the residuals: 51.005

  Parameters of node thalach (conditional Gaussian distribution)

Conditional density: thalach | age + num
Coefficients:
                      0            1
(Intercept)  213.9876303  159.6233506
age           -1.0524778   -0.3614196
Standard deviation of the residuals:
       0        1
16.22507  22.61379
Discrete parents' configurations:
   num
0    0
1    1

  Parameters of node exang (multinomial distribution)
```

```
Conditional probability table:

     cp
exang          1          2          3          4
    0 0.82608696 0.91836735 0.86746988 0.45070423
    1 0.17391304 0.08163265 0.13253012 0.54929577

  Parameters of node oldpeak (conditional Gaussian distribution)

Conditional density: oldpeak | num
Coefficients:
                    0         1
(Intercept)  0.598750  1.589051
Standard deviation of the residuals:
        0          1
0.7871601  1.3050061
Discrete parents' configurations:
   num
0    0
1    1

  Parameters of node ca (multinomial distribution)

Conditional probability table:
          0          1          2          3
0.58585859 0.21885522 0.12794613 0.06734007

  Parameters of node thal (multinomial distribution)

Conditional probability table:
          3          6          7
0.55218855 0.06060606 0.38720539

  Parameters of node num (multinomial distribution)

Conditional probability table:

, , exang = 0, ca = 0, thal = 3

   cp
num          1          2          3          4
  0 0.80000000 0.96551724 0.90243902 0.86956522
  1 0.20000000 0.03448276 0.09756098 0.13043478

, , exang = 1, ca = 0, thal = 3

   cp
num          1          2          3          4
```

```
  0 1.00000000 1.00000000 1.00000000 0.60000000
  1 0.00000000 0.00000000 0.00000000 0.40000000

, , exang = 0, ca = 1, thal = 3

   cp
num          1          2          3          4
  0 0.50000000 0.50000000 1.00000000 0.25000000
  1 0.50000000 0.50000000 0.00000000 0.75000000

, , exang = 1, ca = 1, thal = 3

   cp
num          1          2 3          4
  0 1.00000000 1.00000000   0.00000000
  1 0.00000000 0.00000000   1.00000000

, , exang = 0, ca = 2, thal = 3

   cp
num          1          2          3          4
  0 0.50000000 0.66666667 1.00000000 0.66666667
  1 0.50000000 0.33333333 0.00000000 0.33333333

, , exang = 1, ca = 2, thal = 3

   cp
num 1 2 3          4
  0       0.00000000
  1       1.00000000

, , exang = 0, ca = 3, thal = 3

   cp
num 1 2          3          4
  0     0.50000000 0.00000000
  1     0.50000000 1.00000000

, , exang = 1, ca = 3, thal = 3

   cp
num 1 2 3          4
  0       0.00000000
  1       1.00000000

, , exang = 0, ca = 0, thal = 6

   cp
```

```
num          1          2 3          4
  0 1.00000000 1.00000000   1.00000000
  1 0.00000000 0.00000000   0.00000000

, , exang = 1, ca = 0, thal = 6

   cp
num 1 2 3          4
  0       0.33333333
  1       0.66666667

, , exang = 0, ca = 1, thal = 6

   cp
num 1 2          3 4
  0     0.00000000
  1     1.00000000

, , exang = 1, ca = 1, thal = 6

   cp
num 1 2          3          4
  0     0.00000000 0.00000000
  1     1.00000000 1.00000000

, , exang = 0, ca = 2, thal = 6

   cp
num 1 2 3          4
  0       0.00000000
  1       1.00000000

, , exang = 1, ca = 2, thal = 6

   cp
num 1 2 3          4
  0       0.00000000
  1       1.00000000

, , exang = 0, ca = 3, thal = 6

   cp
num 1 2 3          4
  0       0.00000000
  1       1.00000000

, , exang = 1, ca = 3, thal = 6
```

```
   cp
num 1          2 3 4
  0   0.00000000
  1   1.00000000

, , exang = 0, ca = 0, thal = 7

   cp
num          1          2          3          4
  0 0.66666667 0.66666667 0.85714286 0.33333333
  1 0.33333333 0.33333333 0.14285714 0.66666667

, , exang = 1, ca = 0, thal = 7

   cp
num          1 2          3          4
  0 0.50000000   0.33333333 0.11111111
  1 0.50000000   0.66666667 0.88888889

, , exang = 0, ca = 1, thal = 7

   cp
num 1          2          3          4
  0   0.00000000 0.33333333 0.00000000
  1   1.00000000 0.66666667 1.00000000

, , exang = 1, ca = 1, thal = 7

   cp
num 1 2          3          4
  0     0.50000000 0.06250000
  1     0.50000000 0.93750000

, , exang = 0, ca = 2, thal = 7

   cp
num 1 2          3          4
  0     0.00000000 0.00000000
  1     1.00000000 1.00000000

, , exang = 1, ca = 2, thal = 7

   cp
num 1 2 3          4
  0       0.00000000
  1       1.00000000

, , exang = 0, ca = 3, thal = 7
```

```
   cp
num 1 2          3          4
  0     0.33333333 0.20000000
  1     0.66666667 0.80000000

, , exang = 1, ca = 3, thal = 7

   cp
num 1 2 3          4
  0       0.00000000
  1       1.00000000
```

[150]:
```
fitted = bn.fit(simpler, data_no_na)
```

[151]:
```
imputed_data = impute(fitted, data)
```

[152]:
```
dim(data)
dim(imputed_data)
```

1. 303 2. 10

1. 303 2. 10

[205]:
```
imputed_data =  imputed_data[sample(nrow(imputed_data)),]
```

[206]:
```
test_data = tail(imputed_data,50)
train_data = imputed_data
```

[207]:
```
fitted_all = bn.fit(simpler, train_data)

predicted_test = predict(fitted_all, node = "num", data = test_data)
accuracy_test = sum(test_data[,"num"] == predicted_test) /
 ↪length(predicted_test)

predicted_train = predict(fitted_all, node = "num", data = train_data)
accuracy_train = sum(train_data[,"num"] == predicted_train) /
 ↪length(predicted_train)
```

[208]:
```
accuracy_train
accuracy_test
```

0.867986798679868

0.84