Branch: master

Find file Copy path

pybnl / examples / 010-2018-06-15-marks.ipynb

cs224 010-2018-06-15-marks.ipynb: current version is also printing the bnle...

99d5772 on Jun 22, 2018

1 contributor

Raw Blame History

3055 lines (3055 sloc) 114 KB

In [1]:

```
%load_ext watermark
%watermark -a 'Christian Schuhegger' -u -d -v -p n
umpy,xarray,scipy,pandas,sklearn,matplotlib,seabor
n,qgrid,rpy2,libpgm,pgmpy,networkx,graphviz,pybnl,
pytest
```

```
Christian Schuhegger
last updated: 2018-06-22

CPython 3.6.4
IPython 6.2.1

numpy 1.14.2
xarray 0.10.3
scipy 1.0.1
pandas 0.22.0
sklearn 0.19.1
matplotlib 2.2.2
seaborn 0.8.1
qgrid 1.0.2
rpy2 2.9.1
libpgm n
pgmpy n
networkx 2.1
graphviz 0.8.3
pybnl n
pytest 3.5.0
```

Access other versions via nbviwer: [https://nbviewer.jupyter.org/github/cs224/pybnl/blob/4bd08e6e48194dcddeadf0f202b910e3e224753b/exa 2018-06-15-marks.ipynb](https://nbviewer.jupyter.org/github/cs224/pybnl/blob/4bd08e6e48194dcddeadf0f202b910e3e224753b/exa) (https://nbviewer.jupyter.org/github/cs224/pybnl/blob/4bd08e6e48194dcddeadf0f202b910e3e224753b/exa 2018-06-15-marks.ipynb)

In [2]:

```
%matplotlib inline
import numpy as np, pandas as pd, xarray as xr, ma
tplotlib.pyplot as plt, seaborn as sns
import sklearn, sklearn.pipeline
import networkx as nx, graphviz, networkx.algorith
ms.dag
import random

pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
# pd.set_option('display.float_format', lambda x:
 '%.2f' % x)
np.set_printoptions(edgeitems=10)
np.set_printoptions(suppress=True)
np.core.arrayprint._line_width = 180

sns.set()
```

In [3]:

```
from IPython.display import display, HTML

from IPython.display import display_html
def display_side_by_side(*args):
    html_str=''
    for df in args:
        if type(df) == np.ndarray:
            df = pd.DataFrame(df)
        html_str+=df.to_html()
    html_str = html_str.replace('table','table sty
le="display:inline"')
    # print(html_str)
    display_html(html_str,raw=True)

CSS = """
.output {
    flex-direction: row;
}
"""

def display_graphs_side_by_side(*args):
    html_str='<table><tr>'
    for g in args:
        html_str += '<td>'
        html_str += g._repr_svg_()
        html_str += '</td>'
    html_str += '</tr></table>'
    display_html(html_str,raw=True)


display(HTML("<style>.container { width:70% !impor
tant; }</style>"))
```

In [5]:

```
%load_ext rpy2.ipython
```

```
The rpy2.ipython extension is already loaded. To r
eload it, use:
  %reload_ext rpy2.ipython
```

In [6]:

```
%load_ext autoreload
%autoreload 1
%aimport pybnl.bn
```

In [7]:

```
import locale
locale.setlocale(locale.LC_ALL, 'C')

import rpy2, rpy2.rinterface, rpy2.robjects, rpy2.
robjects.packages, rpy2.robjects.lib, rpy2.robject
```

```
s.lib.grid, \
    rpy2.robjects.lib.ggplot2, rpy2.robjects.panda
s2ri, rpy2.interactive.process_revents, \
    rpy2.interactive, rpy2.robjects.lib.grdevices
# rpy2.interactive.process_revents.start()
rpy2.robjects.pandas2ri.activate()
```

In [9]:

```
rpackageversionfn = rpy2.robjects.r('packageVersio
n')
print(rpackageversionfn("bnlearn")[0])
```

```
[1]        4        4 20180620
```

learning.test

Before we look at the `marks` data-set let's look first at a test network provided in the `bnlearn` package: networks (http://www.bnlearn.com/documentation/networks/) <img src='http://www.bnlearn.com/documentation/networks/learning.test.png (http://www.bnlearn.com/documentation/networks/learning.test.png)' width=400>

In [10]:

```
%%R -o rdf_lt
data(learning.test)
rdf_lt = learning.test
```

Converting the R data.frame into a python pd.DataFrame and converting to CategoricalDtype

After loading the data-set we need to convert it so that all variables are of type `CategoricalDtype`: see the pandas documentation about Categorical Data (https://pandas.pydata.org/pandas-docs/stable/categorical.html) for more details.

In [11]:

```
#df_lt = rpy2.robjects.pandas2ri.ri2py(rdf_lt)
df_lt = rdf_lt

ct1 = pd.api.types.CategoricalDtype(['a', 'b', 'c'
], ordered=True)
ct2 = pd.api.types.CategoricalDtype(['a', 'b'], or
dered=True)

for c in 'ABCDE':
    df_lt[c] = df_lt[c].astype(ct1)
```

```
df_lt['F'] = df_lt['F'].astype(ct2)

df_lt.info()
```

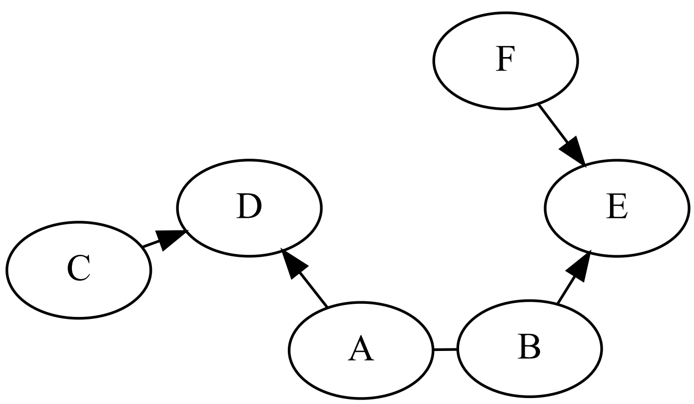
```
<class 'pandas.core.frame.DataFrame'>
Index: 5000 entries, 1 to 5000
Data columns (total 6 columns):
A    5000 non-null category
B    5000 non-null category
C    5000 non-null category
D    5000 non-null category
E    5000 non-null category
F    5000 non-null category
dtypes: category(6)
memory usage: 69.0+ KB
```

ConstraintBasedNetFromDataDiscreteBayesNetwork

In [12]:

```
cbnet = pybnl.bn.ConstraintBasedNetFromDataDiscret
eBayesNetwork(df_lt)
cbnet.fit()
#display_side_by_side(cbnet.structure().dot(),cbne
t.structure().cpdag().dot())
cbnet.structure().cpdag().dot()
```

Out[12]:

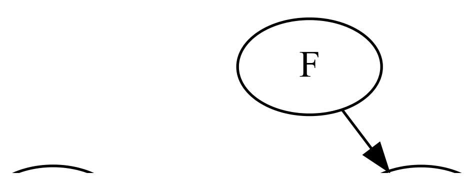


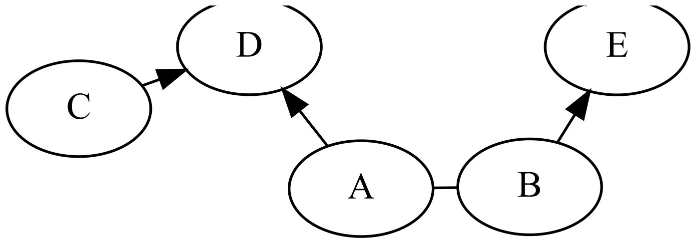
ScoreBasedNetFromDataDiscreteBayesNetwork

In [13]:

```
sbnet = pybnl.bn.ScoreBasedNetFromDataDiscreteBaye
sNetwork(df_lt)
sbnet.fit()
sbnet.structure().cpdag().dot()
```

Out[13]:



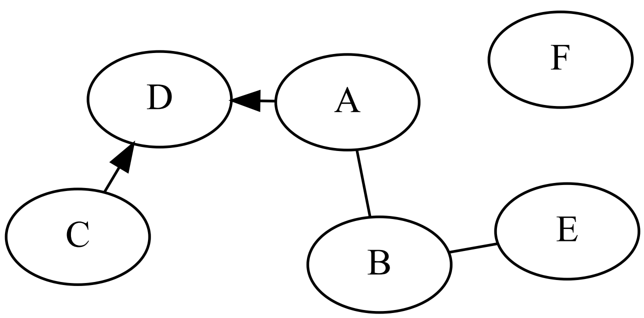


HybridScoreAndConstainedBasedNetFromDataDiscreteBayes

In [14]:

```
hnet1 = pybnl.bn.HybridScoreAndConstainedBasedNetF
romDataDiscreteBayesNetwork(df_lt)
hnet1.fit()
hnet1.structure().cpdag().dot()
```

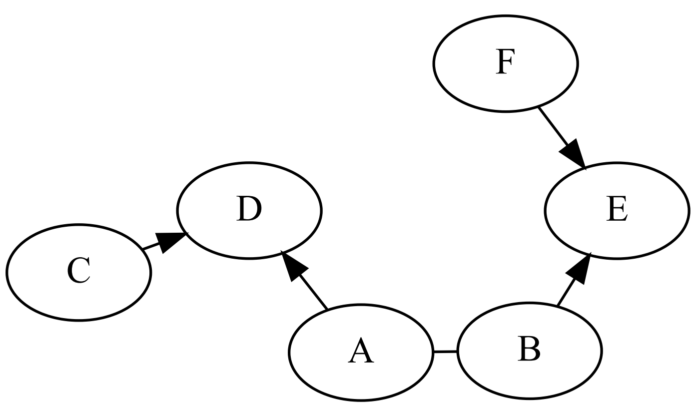
Out[14]:



In [15]:

```
hnet2 = pybnl.bn.HybridScoreAndConstainedBasedNetF
romDataDiscreteBayesNetwork(df_lt, algorithm='rxma
x2_sihitonpc_tabu')
hnet2.fit()
hnet2.structure().cpdag().dot()
```

Out[15]:



In [16]:

```
hnet2.structure().cpdag().vstructs()
```

Out[16]:

	X	Z	Y
0	A	D	C
1	B	E	F

marks

Let's take the detour of loading the `R` data set, writing it to CSV and then loading the CSV via pandas from python. Like that we're sure we have a typical starting position in a `python` data workflow.

In [17]:

```
%%R -o marks
library(bnlearn)
data(marks)
write.csv(marks, file = "marks.csv")
```

In [18]:

```
pd_marks = pd.read_csv('marks.csv', index_col=0).a
stype(np.float64)
pd_marks.head()
```

Out[18]:

	MECH	VECT	ALG	ANL	STAT
1	77.0	82.0	67.0	67.0	81.0
2	63.0	78.0	80.0	70.0	81.0
3	75.0	73.0	71.0	66.0	81.0
4	55.0	72.0	63.0	70.0	68.0
5	63.0	63.0	65.0	70.0	63.0

In [19]:

```
hbt = pybnl.bn.HarteminkBinTransformer(3,ibreaks=1
8)
dmarks = hbt.fit_transform(X=pd_marks)
```

In [20]:

```
# dmarks = pybnl.bn.discretize(pd_marks)
dmarks.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 88 entries, 0 to 87
Data columns (total 5 columns):
MECH    88 non-null category
VECT    88 non-null category
```

```
VECT    88 non-null category
ALG     88 non-null category
ANL     88 non-null category
STAT    88 non-null category
dtypes: category(5)
memory usage: 1.0 KB
```

In [21]:

```
dmarks['MECH'].dtype
```

Out[21]:

```
CategoricalDtype(categories=['(0,35.8]', '(35.8,4
9]', '(49,77]'], ordered=False)
```

In [22]:

```
dmarks.head()
```

Out[22]:

	MECH	VECT	ALG	ANL	STAT
0	(49,77]	(60.7,82]	(58.7,80]	(62.3,70]	(53.7,81]
1	(49,77]	(60.7,82]	(58.7,80]	(62.3,70]	(53.7,81]
2	(49,77]	(60.7,82]	(58.7,80]	(62.3,70]	(53.7,81]
3	(49,77]	(60.7,82]	(58.7,80]	(62.3,70]	(53.7,81]
4	(49,77]	(60.7,82]	(58.7,80]	(62.3,70]	(53.7,81]

Let's also create immediately a marks data-frame that include one additional latent variable that we will need later:

In [23]:

```
ldmarks = dmarks.copy()
pybnl.bn.augment_df_with_latent_variable(ldmarks,
'LAT', 3)
print(pybnl.bn.levels_of_latent_variable(ldmarks,
'LAT'))
ldmarks.head()
```

```
['l000', 'l001', 'l002']
```

Out[23]:

	MECH	VECT	ALG	ANL	STAT	LAT
0	(49,77]	(60.7,82]	(58.7,80]	(62.3,70]	(53.7,81]	NaN
1	(49,77]	(60.7,82]	(58.7,80]	(62.3,70]	(53.7,81]	NaN
2	(49,77]	(60.7,82]	(58.7,80]	(62.3,70]	(53.7,81]	NaN
3	(49,77]	(60.7,82]	(58.7,80]	(62.3,70]	(53.7,81]	NaN
4	(49,77]	(60.7,82]	(58.7,80]	(62.3,70]	(53.7,81]	NaN

NetAndDataDiscreteBayesNetwork

Create network by hand

In [24]:

```
dg = nx.DiGraph()
# G.add_node(1)
dg.add_nodes_from(list(marks.columns))
dg.add_edges_from([
    ['STAT', 'ANL'],
    ['STAT', 'ALG'],
    ['ANL', 'ALG'],
    ['ALG', 'MECH'],
    ['ALG', 'VECT'],
    ['VECT', 'MECH'],
])
```

In [25]:

```
list(nx.connected_components(dg.to_undirected()))
```

Out[25]:

```
[{'ALG', 'ANL', 'MECH', 'STAT', 'VECT'}]
```

In [26]:

```
ns = pybnl.bn.digraph2netstruct(dg)
```

You can display the graph either in the 'dot' format or in the default more compact 'fdp' format

In [27]:

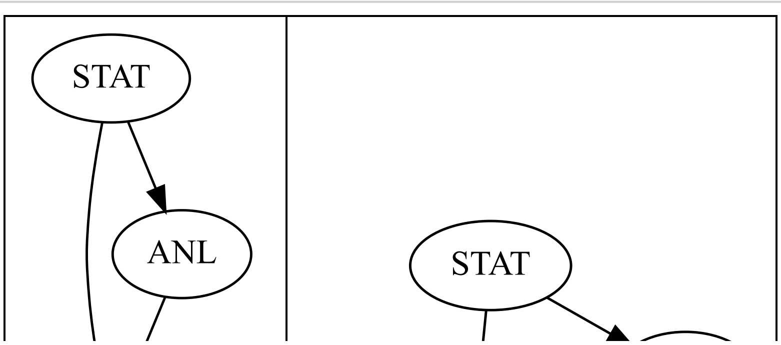
```
type(ns.dot())
```

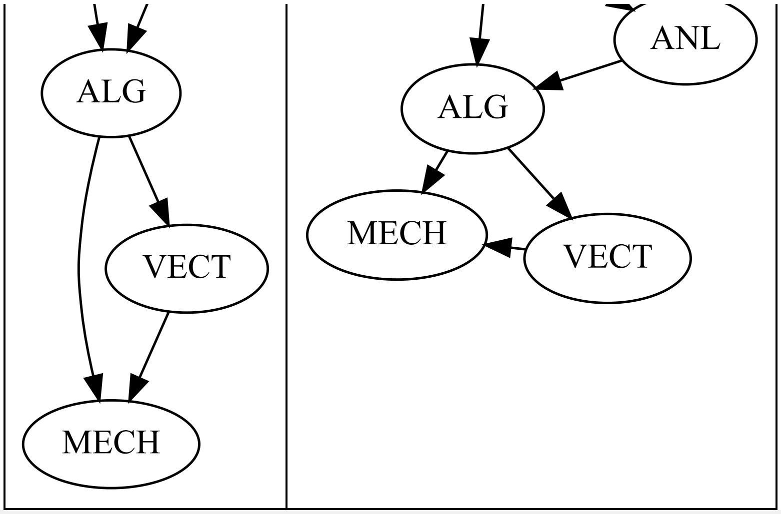
Out[27]:

```
graphviz.dot.Digraph
```

In [28]:

```
display_graphs_side_by_side(ns.dot(engine='dot'), ns.dot())
```





Fit the network

In [29]:

```
net_dmarks = pybnl.bn.NetAndDataDiscreteBayesNetwo
rk(dmarks, rnet=ns.rnet)
net_dmarks.fit()
net_dmarks_net_and_data = net_dmarks
print(net_dmarks.rfit)
```

```
  Bayesian network parameters

  Parameters of node ALG (multinomial distributio
n)

Conditional probability table:

, , STAT = (9,25.5]

             ANL
ALG             (9,39.3] (39.3,62.3] (62.3,70]
  (15,48.8]   1.0000000   0.5000000
  (48.8,58.7] 0.0000000   0.5000000
  (58.7,80]   0.0000000   0.0000000

, , STAT = (25.5,53.7]

             ANL
ALG             (9,39.3] (39.3,62.3] (62.3,70]
  (15,48.8]   0.7692308   0.2250000
  (48.8,58.7] 0.2307692   0.6250000
  (58.7,80]   0.0000000   0.1500000

, , STAT = (53.7,81]

             ANL
ALG             (9,39.3] (39.3,62.3] (62.3,70]
  (15,48.8]   1.0000000   0.1111111 0.0000000
  (48.8,58.7] 0.0000000   0.4444444 0.0000000
  (58.7,80]   0.0000000   0.4444444 1.0000000
```

```
  Parameters of node ANL (multinomial distributio
n)

Conditional probability table:
 
             STAT
ANL             (9,25.5] (25.5,53.7] (53.7,81]
  (9,39.3]    0.7333333   0.2452830 0.0500000
  (39.3,62.3] 0.2666667   0.7547170 0.4500000
  (62.3,70]   0.0000000   0.0000000 0.5000000

  Parameters of node MECH (multinomial distributio
n)

Conditional probability table:
 
, , VECT = (9,42.2]

           ALG
MECH          (15,48.8] (48.8,58.7]  (58.7,80]
  (0,35.8]  0.77777778  0.50000000 0.00000000
  (35.8,49] 0.22222222  0.33333333 1.00000000
  (49,77]   0.00000000  0.16666667 0.00000000

, , VECT = (42.2,60.7]

           ALG
MECH          (15,48.8] (48.8,58.7]  (58.7,80]
  (0,35.8]  0.33333333  0.33333333 0.57142857
  (35.8,49] 0.40000000  0.57142857 0.14285714
  (49,77]   0.26666667  0.09523810 0.28571429

, , VECT = (60.7,82]

           ALG
MECH          (15,48.8] (48.8,58.7]  (58.7,80]
  (0,35.8]  0.00000000  0.14285714 0.00000000
  (35.8,49] 1.00000000  0.57142857 0.08333333
  (49,77]   0.00000000  0.28571429 0.91666667


  Parameters of node STAT (multinomial distributio
n)

Conditional probability table:
    (9,25.5] (25.5,53.7]   (53.7,81] 
  0.1704545   0.6022727   0.2272727 

  Parameters of node VECT (multinomial distributio
n)

Conditional probability table:
 
             ALG
VECT            (15,48.8] (48.8,58.7]  (58.7,80]
  (9,42.2]    0.52941176  0.17647059 0.05000000
```

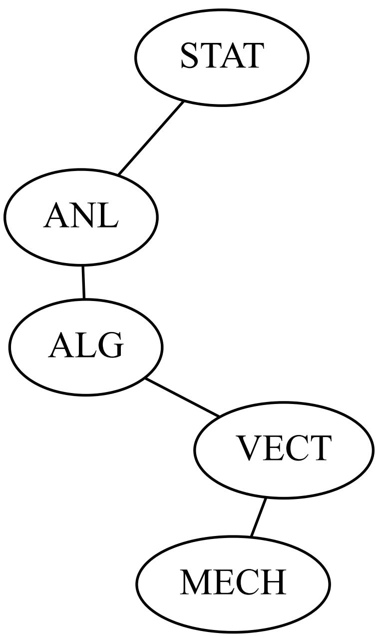
```
(42.2,60.7] 0.44117647  0.61764706 0.35000000
(60.7,82]   0.02941176  0.20588235 0.60000000
```

ConstraintBasedNetFromDataDiscreteBayesNetwork

In [30]:

```
net_dmarks = pybnl.bn.ConstraintBasedNetFromDataDiscreteBayesNetwork(dmarks)
net_dmarks.fit()
net_dmarks_cb = net_dmarks
net_dmarks.structure().cpdag().dot()
```

Out[30]:

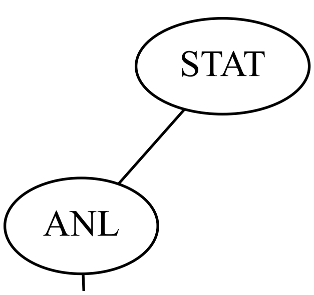


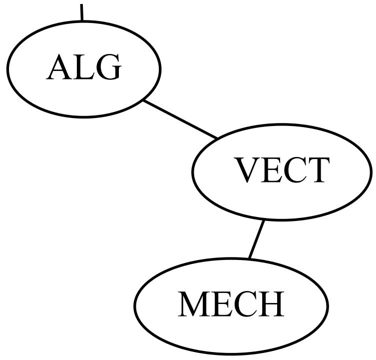
ScoreBasedNetFromDataDiscreteBayesNetwork

In [31]:

```
net_dmarks = pybnl.bn.ScoreBasedNetFromDataDiscreteBayesNetwork(dmarks)
net_dmarks.fit()
net_dmarks_sb = net_dmarks
net_dmarks.structure().cpdag().dot()
```

Out[31]:





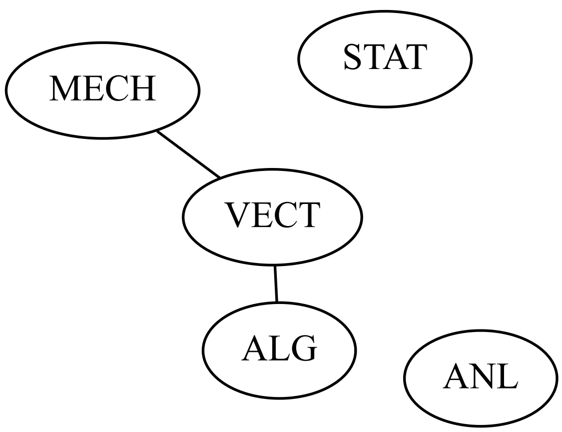
HybridScoreAndConstainedBasedNetFromDataDiscreteBayes

mmhc

In [32]:

```
net_dmarks = pybnl.bn.HybridScoreAndConstainedBase
dNetFromDataDiscreteBayesNetwork(dmarks)
net_dmarks.fit()
net_dmarks_hyb_mmhc = net_dmarks
net_dmarks.structure().cpdag().dot()
```

Out[32]:

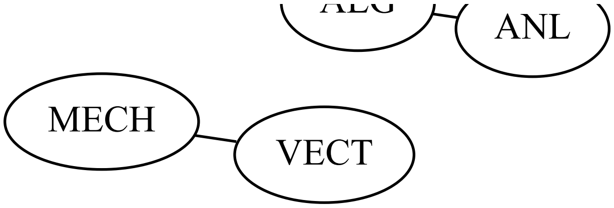


rxmax2_sihitonpc_tabu

In [33]:

```
net_dmarks = pybnl.bn.HybridScoreAndConstainedBase
dNetFromDataDiscreteBayesNetwork(dmarks, algorithm
='rxmax2_sihitonpc_tabu')
net_dmarks.fit()
net_dmarks_hyb_rsmax = net_dmarks
net_dmarks.structure().cpdag().dot()
```

Out[33]:



Comparison

In [34]:

```
def score(type='bic'):
    cdf = pd.DataFrame(columns=['algorithm', 'scor
e'])
    cdf.loc[len(cdf)] = ['ConstraintBased', net_dm
arks_cb.score(type=type)]
    cdf.loc[len(cdf)] = ['ScoreBased', net_dmarks_
sb.score(type=type)]
    cdf.loc[len(cdf)] = ['mmhc', net_dmarks_hyb_mm
hc.score(type=type)]
    cdf.loc[len(cdf)] = ['rsmax', net_dmarks_hyb_r
smax.score(type=type)]
    return cdf

display_side_by_side(score(), score(type='loglik'
))
```

	algorithm	score
0	ConstraintBased	-411.497387
1	ScoreBased	-411.497387
2	mmhc	-452.330225
3	rsmax	-434.906936

	algorithm	score
0	ConstraintBased	-353.292008
1	ScoreBased	-353.292008
2	mmhc	-412.034194
3	rsmax	-394.610904

In [35]:

```
net_dmarks_cb.structure().bf(net_dmarks_hyb_mmhc.s
tructure(), dmarks)
```

Out[35]:

45.10970279456899

In [36]:

```
net_dmarks_cb.bf(net_dmarks_hyb_mmhc, dmarks)
```

Out[36]:

```
45.10970279456899
```

In [37]:

```
net_dmarks_cb.bf(net_dmarks_hyb_mmhc)
```

Out[37]:

```
45.10970279456899
```

sklearn pipeline

In [38]:

```
pl = sklearn.pipeline.Pipeline(
    steps=[
        ('hartemink', pybnl.bn.HarteminkBinTransfo
rmer(3,ibreaks=18)),
        ('score', pybnl.bn.ScoreBasedNetFromDataDi
screteBayesNetwork()),
          ]
)
pl.set_params(hartemink__breaks=3).fit(pd_marks, N
one)
```

Out[38]:

```
Pipeline(memory=None,
     steps=[('hartemink', HarteminkBinTransformer
(breaks=3, ibreaks=18)), ('score', ScoreBasedNetFr
omDataDiscreteBayesNetwork(algorithm=None, ldf=Non
e,
                     whitelist=None))])
```

In [39]:

```
pl.score(dmarks)
```

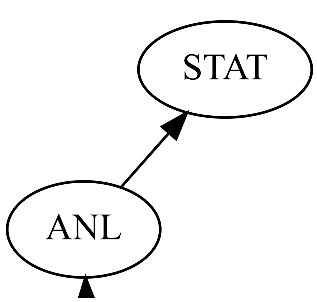
Out[39]:

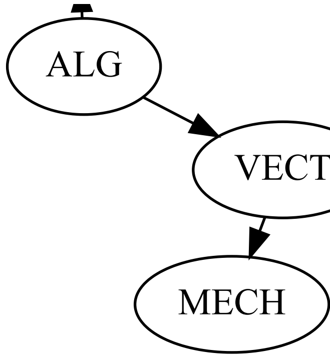
```
-411.49738682714093
```

In [40]:

```
pl.steps[-1][1].structure().dot()
```

Out[40]:





In [41]:

```
pl.set_params(hartemink__breaks=5).fit(pd_marks, None)
pl.score(dmarks)
```

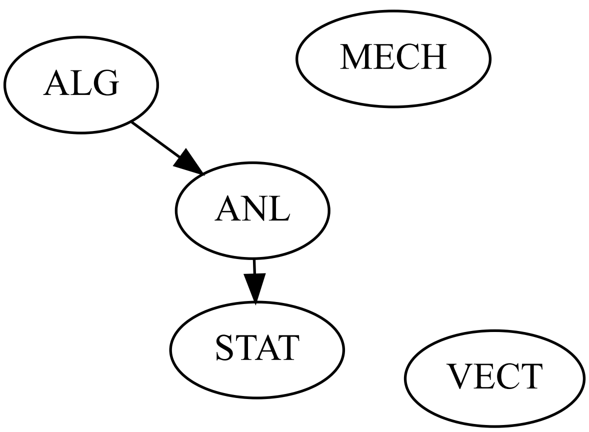
Out[41]:

```
-682.3887051571114
```

In [42]:

```
pl.steps[-1][1].structure().dot()
```

Out[42]:



In [43]:

```
pl.set_params(hartemink__breaks=7).fit(pd_marks, None)
pl.score(dmarks)
```

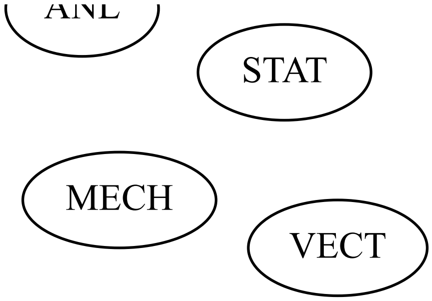
Out[43]:

```
-876.9315865210814
```

In [44]:

```
pl.steps[-1][1].structure().dot()
```

Out[44]:



StructuralEMNetFromDataDiscreteBayesNetwork