Bayesian reasoning and machine learning: opdracht 2

Wilfred Van Casteren
nr:837377516
Open Universiteit
Masteropleiding AI

July 14, 2025

Introduction

In this writing I will be using 2 algorithms for causal discovery, the greedy hill climbing algorithm (GHC) and the Multivariate Information-based Inductive Causation algorithm (MIIC). I will compare generated networks by both of the algorithms, using the ROC-curves and more specifically the Area Under the Curve (AUC) values. I will also add some information that compares structural networks discovered by the already named algorithms. Finally I will add some expert knowledge and compare the AUC values again.

Type of causal discovery algorithms

According to [1], causal discovery algorithms can broadly be split into three categories.

- **Search-and-score algorithms** search for a Bayesian network structure that fits the data best.
- **Constraint-based algorithms** carry out a conditional (in)dependence analysis on the data. Based on this analysis an undirected graph is generated (to be interpreted as a Markov network). Using additional independence tests, some of the arcs can be directed.
- **Hybrid algorithms** combine both approaches.

Multivariate Information-based Inductive Causation (MIIC)

MIIC combines constraint-based and information-theoretic approaches to disentangle direct from indirect effects amongst correlated variables, including cause-effect relationships and the effect of unobserved latent causes.

Greedy Hill Climbing (GHC) algorithm

A GHC algorithm is often used for solving mathematical optimization problems in AI. With a good heuristic function and a large set of inputs, Hill Climbing can find a sufficiently good solution in a reasonable amount of time, although it may not always find the global optimal maximum.

Remarks As saving some models to some formats generated some issues I have changed some of the columns' values, by using values not containing string values like "-", "¡", "1","3". As this kind of code doesn't add anything I have omitted the data fetching procedure from this writing. I have used python version 3.12.8 for my python notebook.

Remark concerning folder setup

As I never completely know how I will tackle an assignment, I have created some overhead code. Code that is superfluous for this writing, code that creates objects that aren't used any more. To save these objects I sometimes create folders.

As sometimes folders are created via Python, and I sometimes create them by hand, it is possible some directory isn't created for some reason, resulting in an error when saving some object.

To make sure all directories are present when needed, the notebook will be zipped with its accompanying folders. I know the code to generate folders is far from optimal, but it's already late, and I want to conclude.

It could however be all folders are created when needed, so no errors will be generated while running the notebook solely, in that case it should run without errors.

Learning the network

Expert network

The network as created by experts is to be seen in Figure 1.

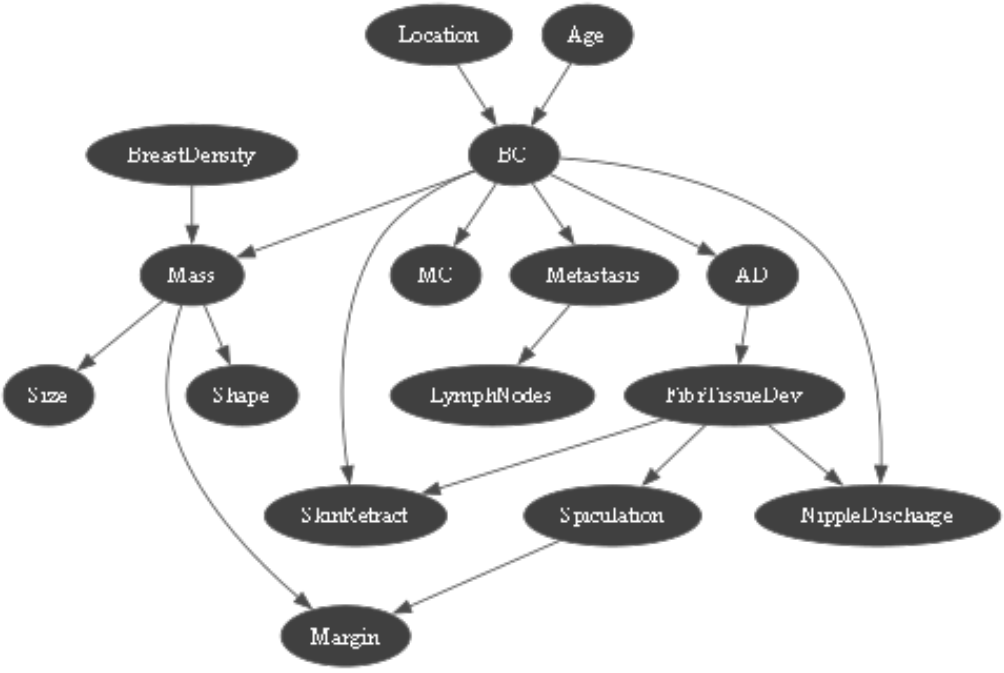


Figure 1: Network as created by experts.

Code to learn the network

In Listing 1 the code to learn the different networks is displayed. The PC algorithm is used to produce the inference for the ground truth network. The code is also used to save networks, create images.

Listing 1: Learning the network

```
def learn(data, template, smoothing, alg, dir=""):
 learner = gum.BNLearner(data,template)

 #code to produce directory structure not shown

 if alg=="pc":
  bn2 = learner.useSmoothingPrior(smoothing).learnBN()
  bn2.saveNET(network+"learned_bn.net", allowModificationWhenSaving=False)
 if alg=="miic":
  bn2=learner.useMIIC().learnBN()
  bn2.saveNET(network+"learned_bn.net", allowModificationWhenSaving=False)
 if alg=="greedy":
  bn2=learner.useGreedyHillClimbing().useNMLCorrection().useScoreBDeu().learnBN()
  bn2.saveNET(network+"learned_bn.net", allowModificationWhenSaving=False)

 print("is constraintbased: "+ str(learner.isConstraintBased()))
 bn3=learner.learnParameters(bn2)
 inf = gnb.getInference(bn3)
 if dir=="":
  export_html_string(inf, "opdracht2/out/inf" )
 else:
  export_html_string(inf, dir+"/inf" )
 gumimage.export(bn3, dir+"/bn-learned.png")
 gum.saveBN(bn3,network+"network.dsl")
 return bn2
```

Remark Some code, in the listing above, is only there to generate images, html or other objects, in other listings some 'metacode' is present too. Some of the generated objects will be presented in the appendices or in later sections.

Code to generate a template

In Listing 1 an argument named template is used, most of the time this template only contains nodes. Listing 2 is used to generate such an edgeless DAG.

When learning the parameters for the expert network, the expert template is used, this to keep the structure developed by experts and thus only learning the parameters from the data.

Listing 2: Generating an edgeless template

```
def get_template(data):
 template = gum.BayesNet()

 for cname in data:
  if not cname=="index":
   labels= data[cname].unique()
   template.add(gum.LabelizedVariable(cname, cname, labels))

 gumimage.export(template, "out/null-template.png", size=30)
 return template
```

Structural difference between MIIC Learned network and Greedy hill climbing (GHC) algorithm

When comparing learned DAGs generated by the MIIC and GHC algorithm, a lot of differences can be seen. In Figure 2 only a few edges are the same for both algorithms. These few edges are grey. The purple edges are reversed or superfluous (dashed).

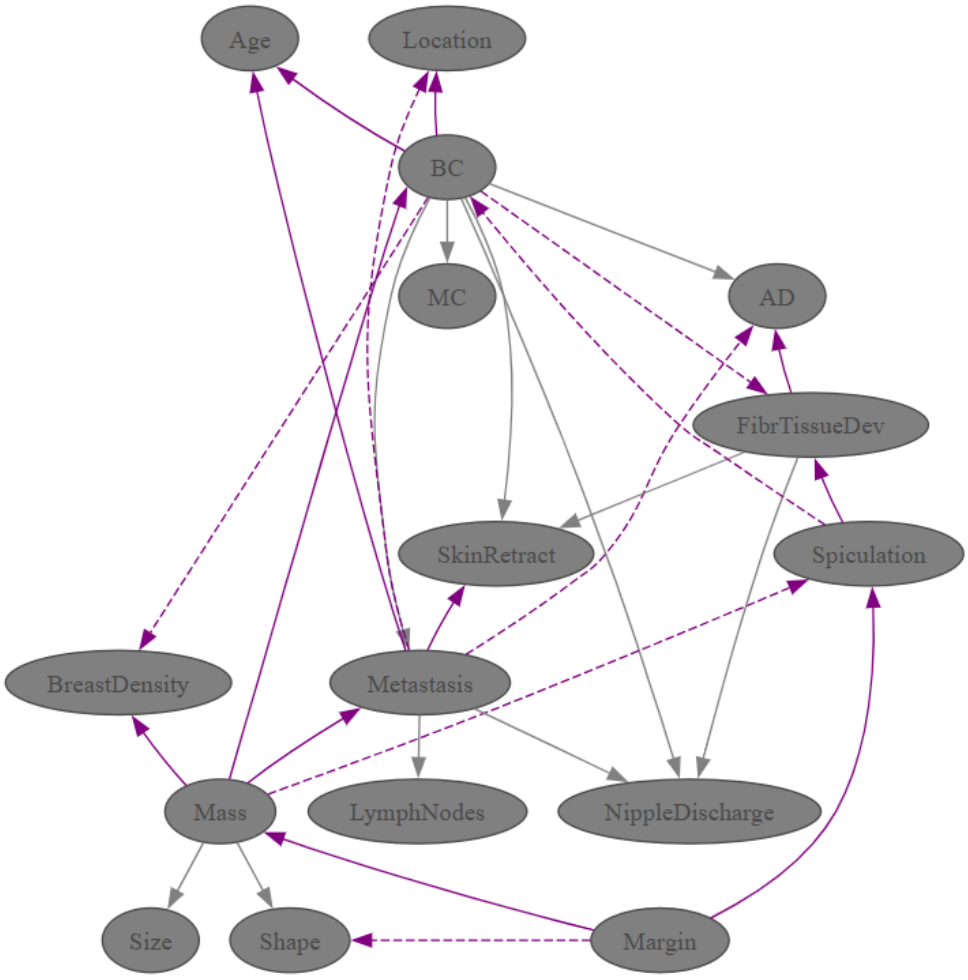


Figure 2: Differences between learned DAGs.

Hamming distance

Hamming distance is the difference of edges comparing the 2 skeletons, and structural Hamming difference is the difference comparing the cpdags, including the arcs' orientation. For the two learned DAGs the hamming distance is 7, and the structural hamming distance is 23.

Comparing networks

To compare the MIIC and GHC generated networks it is a good idea to generate examples from the expert network, the generated examples are being saved in a csv-file. Likewise a validation csv-file is being created too.

Exporting examples to a csv-file

In Listing 3 code is displayed to create csv-files from a network. In this case a normal and validation file is being created.

Listing 3: Generating and exporting examples

```
import pyagrum as gum
gum.generateSample(gt_learned_bn,10000,"data/out/pc/pc.csv",show_progress=True,with_labels=
    ↪ True)
gum.generateSample(gt_learned_bn,10000,"data/out/pc/pc_validation.csv",show_progress=True,
    ↪ with_labels=True)
```

Generating and returning a balanced dataset

To generate a balanced sample from the generated files, Listing 4 is used. Some column manipulations have been omitted for simplicity.

Listing 4: Get a balanced sample from the datasource

```
def get_gen_data(alg, number, validation=False):
 if validation:
  data=pd.read_csv("data/out/"+alg+"/"+alg+".csv")
 else:
  data=pd.read_csv("data/out/"+alg+"/"+alg+"_validation.csv")

 bc_data=data[data['BC']=='Yes']
 nobc_data=data[data['BC']=='No']
 bal_data=pd.concat([bc_data.sample(int(number/2)), nobc_data.sample(int(number/2))])

 return bal_data
```

Statements to get datasets

As all files containing generated data have been created, data can be prepared for learning and validating models, see Listing 5. Samples are being created containing 100, 500, 1000 items, a validation sample is created too.

Listing 5: Get sample for learning and testing

```
validation=get_gen_data("pc", 1000, validation=True)
smp100= get_gen_data("pc", 100)
smp500= get_gen_data("pc", 500)
smp1000= get_gen_data("pc", 1000)
```

Statements to learn networks

In Listing 6 the statements to learn the Bayesian Networks is displayed. For every sample a network is being learned, in this case the network is learned using the MIIC algorithm.

Listing 6: Learning a network

```
miic_1000_learned_bn=learn(smp1000, template,0,"miic", "miic1000")
miic_500_learned_bn=learn(smp500, template,0,"miic", "miic500")
miic_100_learned_bn=learn(smp100, template,0,"miic", "miic100")
```

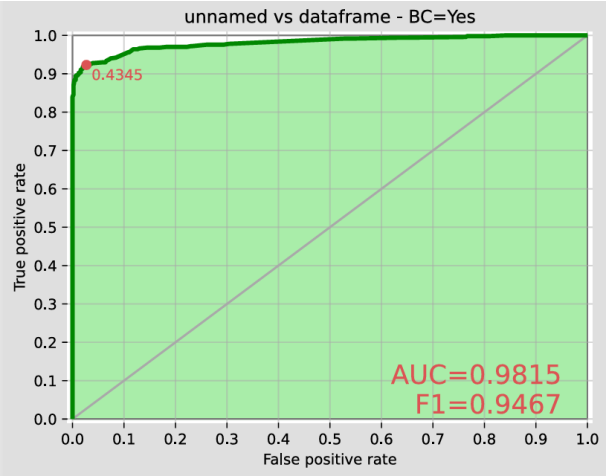
The learn function as used in Listing 6 To have an idea what kind of arguments the learn statement expects, a small explication is in place.

$$miic\_1000\_learned\_bn = learn(smp\_1000, template, 0, "miic", "miic1000")$$

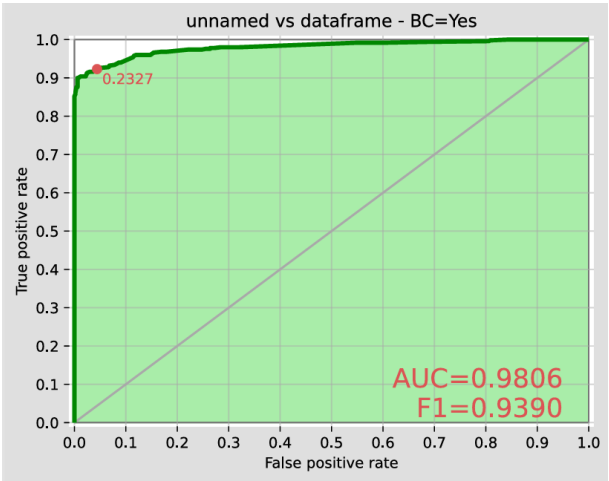
In the above statement, the "smp_1000" argument refers to a dataset containing 1000 rows, generated using the original expert network, the "template" argument refers to a network containing only nodes, "miic" is the algorithm used. And the last argument is a directory, used for saving objects.

Comparing ROC-curves

The code in Listing 7, generates the ROC-curves that are shown in Figure 3. The area under the curve (AUC) values are somewhat different. For both algorithms, the network, learned using 1000 items, performs slightly better than the network learned, using only 500 items. Globally the 'GHC-trained' network performs better than the 'MIIC-trained' one, as it has a higher AUC value.

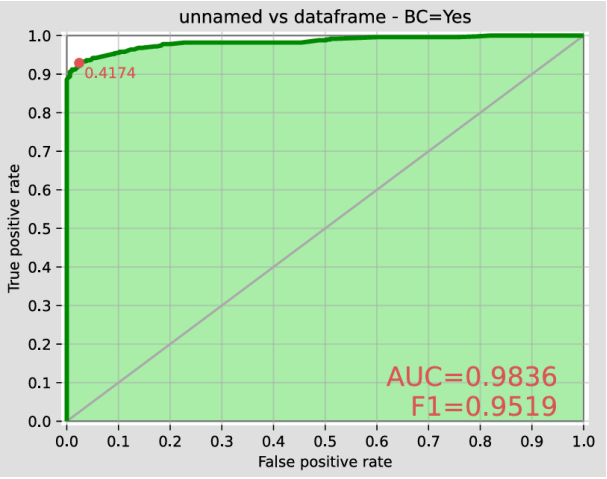


(a) The ROC-curve for a network trained with a balanced sample containing 1000 examples

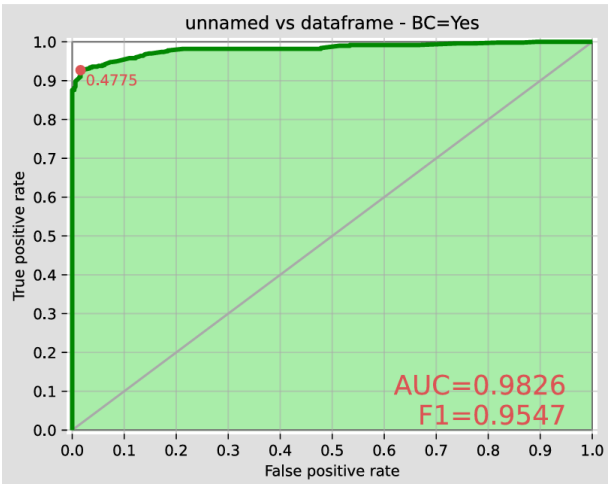


(b) The ROC-curve for a network trained with a balanced sample containing 500 examples

Figure 3: ROC-curves for 2 'MIIC-trained' networks using 1000 and 500 examples



(a) The ROC-curve for a network 'trained' with a balanced sample containing 1000 examples



(b) The ROC-curve for a network 'trained' with a balanced sample containing 500 examples

networks

Figure 4: ROC-curves for 2 'GHC-trained' networks using 1000 and 500 examples

Listing 7: Generating ROC-curves

```
showROC(miic_1000_learned_bn,validation,'BC','Yes',show_progress=False, show_fig=True,
    ↪ save_fig=False, with_labels=True, significant_digits=4)
showROC(miic_500_learned_bn,validation,'BC','Yes',show_progress=False, show_fig=True,
    ↪ save_fig=False, with_labels=True, significant_digits=4)
```

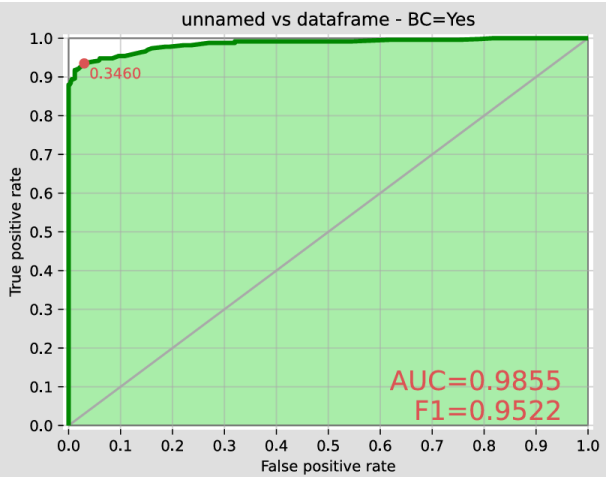
Add some expert knowledge

As the MIIC algorithm introduced edges that are superfluous, see Figure 7, I will add some forbidden arcs to the learner (defined in an earlier section) when using the MIIC algorithm by replacing line 10 of Listing 1 with this line of code.

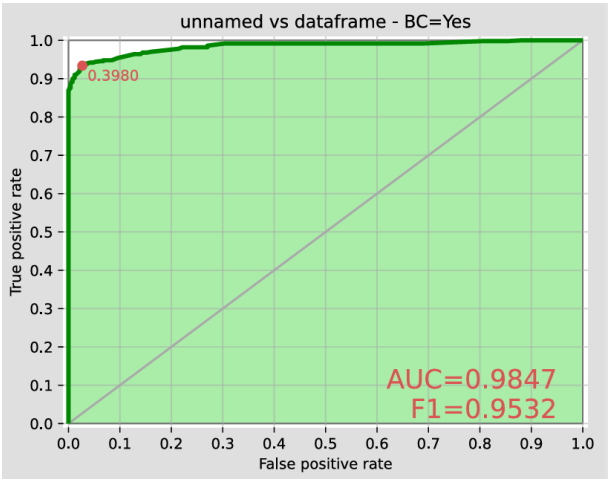
$$bn2 = learner.useMIIC().addForbiddenArc("Age","Metastasis")$$
$$.addForbiddenArc("SkinRetract","Metastasis").learnBN()$$

Comparing ROC-curves

The ROC-curves for the adapted learner's networks are slightly better than without adding expert knowledge, see Figure 5. Some caution is appropriate as in this case data is not generated using the same seed as in an earlier experiment.



(a) The ROC-curve for a network 'trained' with a balanced sample containing 1000 examples



(b) The ROC-curve for a network 'trained' with a balanced sample containing 500 examples

Figure 5: ROC-curves for 2 'MIIC-trained' networks using 1000 and 500 examples, while adding some expert knowledge

Conclusion

In this writing only some of the techniques used when doing causal discovery were shown. And the techniques that were exposed, were only exposed superficially. Furthermore it would be highly inappropriate to claim the GHC algorithm is the best for this kind of dataset. Only adding some expert information showed the MIIC algorithm outperformed the GHC algorithm. Furthermore, there are still a lot of algorithms available and unused in this writing. Also some experiments should be done using some cross-validation techniques and should be accompanied by statistical and other numerical data. It wouldn't hurt to use other software packages to get a broader/other insight in the data and its possible DAGs.

Bibliography

[1] Marco Scutari, Catharina Elisabeth Graafland, and José Manuel Gutiérrez. Who learns better bayesian network structures: Constraint-based, score-based or hybrid algorithms? In Václav Kratochvíl and Milan Studený, editors, *Proceedings of the Ninth International Conference on Probabilistic Graphical Models*, volume 72 of *Proceedings of Machine Learning Research*, pages 416–427. PMLR, 11–14 Sep 2018.

Expert DAG

The expert DAG is to be seen in Figure 6.

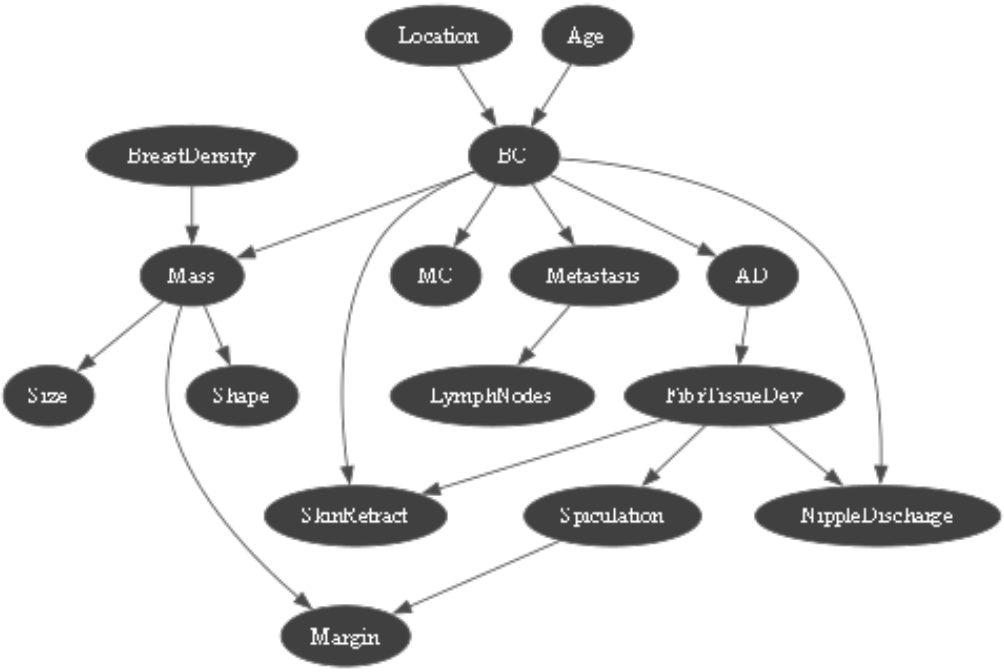


Figure 6: Expert DAG.

Difference MIIC from expert DAG

The MIIC algorithm does a pretty good job, Only 4 edged are superfluous, the dashed purple ones.

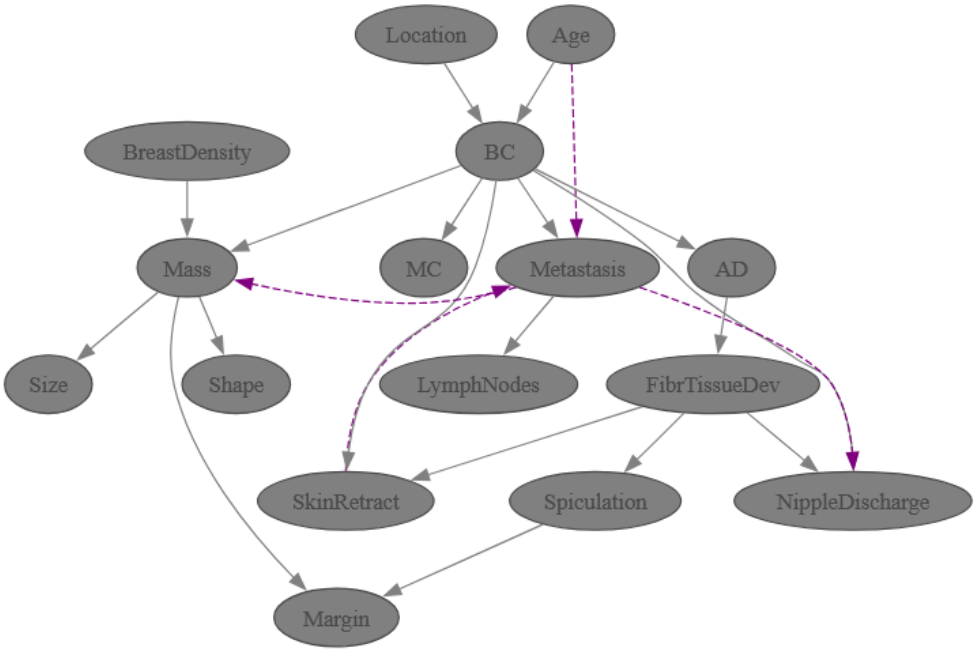


Figure 7: Difference between MIIC DAG and expert DAG.

Difference GHC from expert DAG

This algorithm's network doesn't even come close to the expert DAG, most edges are wrong, either they are superfluous (purple dashed) or reversed (purple).

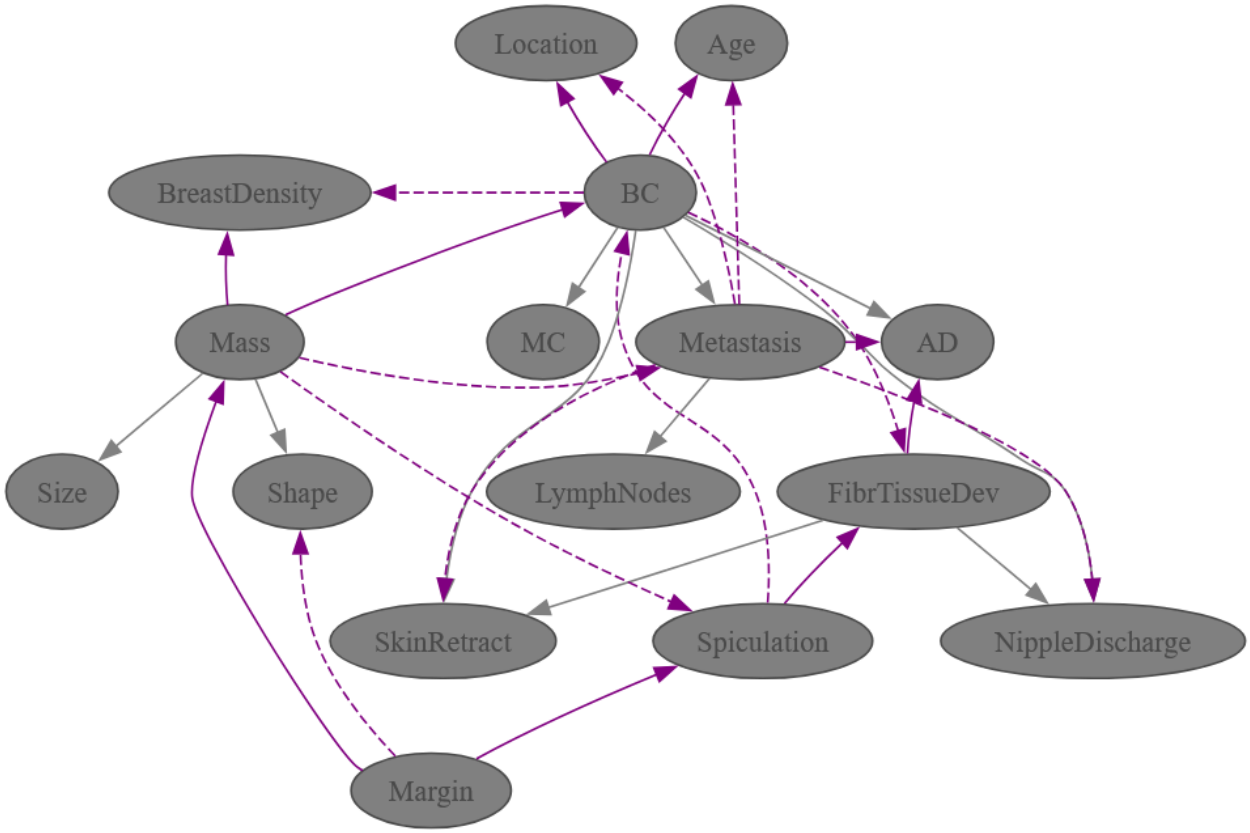


Figure 8: Difference between GHC DAG and expert DAG.

Learned inference for MIIC,GHC algorithm and PC algorithm

This section contains parameters as learned when combining the DAGS as learned using MIIC and GHC with the data that accompanied the assignment. It also contains the so-called ground truth inference using the expert DAG combined with the data.

MIIC inference

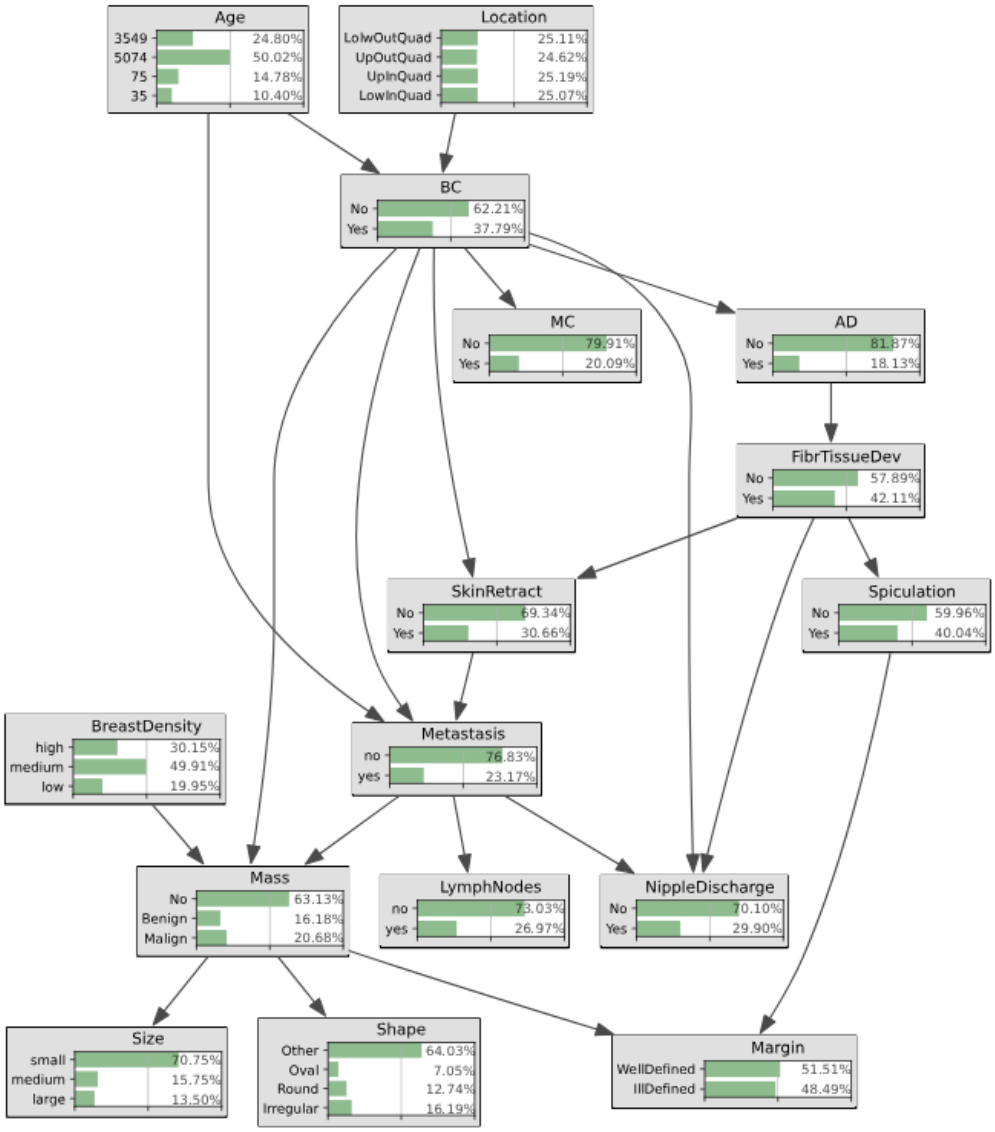


Figure 9: Inference when using MIIC.

GHC inference

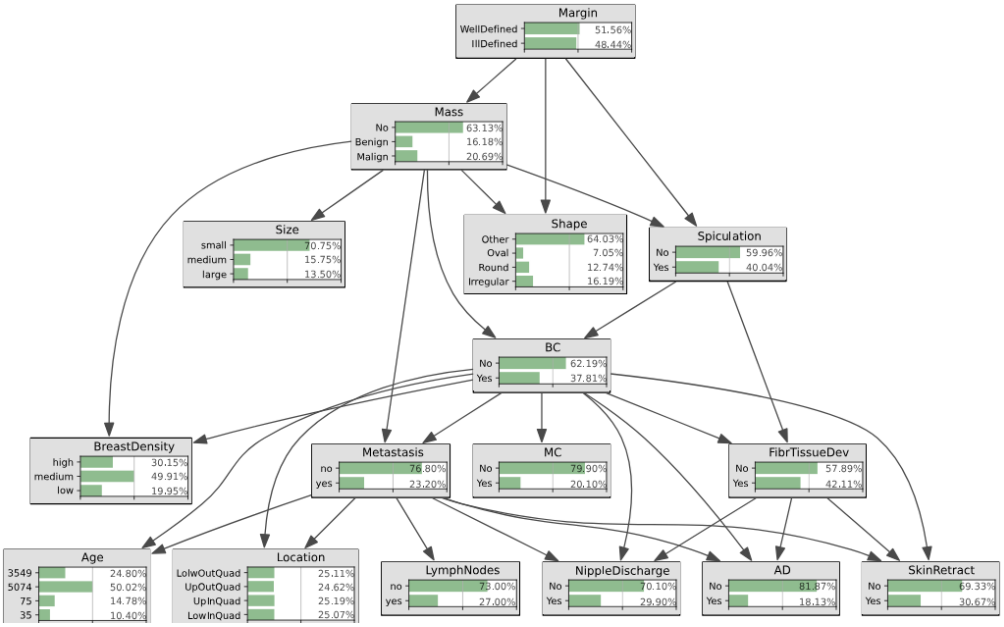


Figure 10: Inference when using GHC.

Expert inference

Figure 11: Expert inference.