GRAPHICAL METHODS FOR DIAGNOSTICS IN MEDICINE.

GIANPIERO CEA

Abstract

In this paper we discuss the uses of a special class of Machine Learning methods called Graphical Methods for Diagnostics in Medicine. We demonstrate this by applying GMs to the Wisconsin Breast Cancer Dataset.

1 INTRODUCTION

Machine Learning is one of the most prolific area of research in recent years. This is due in part to the enormous amount of data we are able to collect nowadays from various sources (e.g. smart phones, internet of things etc..) and to improvements in raw computational power that can be used to train very sophisticated models that can learn highly specialized data.

One field where Machine Learning can have a huge impact is Medicine and Health care: tools coming from Data Analysis and Machine Learning can help doctors and patients to simplify and significantly improve the accomplishing of an enormous variety of tasks: personalised medicine, automatic recognition of tumor cells from scans, intelligent track of patients' health data and so on.

One area in particular where Machine Learning is widely used in the medical field is for doing diagnostics, which can be defined as the subfield of Medicine whose goal is to determine the possible causes for a certain symptom in a patient and the relations between these. From a Machine Learning perspective, solving a diagnostics problem can mean solving at the same time a lot of other sub-problems such as classification or prediction, since we can use our understanding of the relations in the data to achieve this.

A lot of the Machine Learning algorithms are so called "Black Box" methods, which means the model is optimized in some way to learn from the data but the exact algorithm that it follows is in most cases hidden to the user: most implementations of Deep Neural Networks belong to this family of algorithms for example. Clearly, even if some of these methods can have very remarkable predictive power, in a Medical setting they carry a substantial disadvantage: if for example a model is to be used by a doctor to help him making a decision about a patient treatment, the doctor himself need to first have a clear understanding of what the model is doing before he can "trust" it and use the result is getting from it when taking his decision.

These are the reasons why "White Box" approaches are generally preferred in Medicine: these are methods where the actual "reasoning" that the model uses is somewhat explicit or transparent to the the expert using it. One class of such methods are Graphical Methods: they are called graphical because they all have an underlined associated graph that is representing the "relations" between the various variables. As we will see in more detail later on, learning a graphical model like a Bayesian Network (BN) means most of the times to learn both a structure (i.e. the graph) and the best parameters associated to each node of the graph.

Our focus in this paper will mainly be practical: for this reason we exemplify the use of two of the main Graphical Methods, Bayesian Networks and Markov networks by applying them to the very popular Wisconsin

Breast Cancer dataset. We decided to implement our code in R since this provided the most useful libraries, in particular bnlearn.

1.1 *Theory*

Graphical Methods are a class of probabilistic models, as such some basic knowledge of probability is assumed. In particular of great importance is Bayesian probability. We quickly recall the fundamental Bayes Rule:

Theorem 1. Let X, Y be random variables on a probability space (S,$\mathscr{A}$,P) Then

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

Definition 1.1. A Bayesian Network is a tuple ($\mathscr{P}$,$\mathscr{X}$,$\vec{G}$) where $\mathscr{P}$ = (S,$\mathscr{A}$,P) is a probability space, $\mathscr{X} = \{X_1, ..., X_n\}$ is a set of random variables defined on the P-space and $\vec{G}$ is a simple acyclic directed graph (DAG).

Then the arrows of $\vec{G}$ represents probabilistic independency assumption. In a way, Bayesian Networks are a very compact way to represents joint distribution on a set of variables. We must have:

$$P(X_1, ..., X_n) = \prod_{i=1}^{n} P(X_i|Pa(X_i))$$

where $Pa(X_i)$ is the set of parents of X_i, i.e. the set of those r.v. in $\mathscr{X}$ that have an arrow in $\vec{G}$ pointing towards X_i

Definition 1.2. A Markov Network (also Markov Random Field) is a tuple ($\mathscr{P}$,$\mathscr{X}$,G) where $\mathscr{P}$ = (S,$\mathscr{A}$,P) is a probability space, $\mathscr{X} = \{X_1, ..., X_n\}$ is a set of random variables defined on the P-space and G is a simple undirected graph.

Here instead of having a CPD at each node, we just have a generic positive potential function *phi* over set of variables associated with graphs cliques C. We then have:

$$P(X_1, ..., X_n) = \frac{1}{Z} \prod_{c \in C} \phi(x_c)$$

where Z is a normalization variable.

1.2 *The Wisconsin Breast Cancer Dataset*

The dataset we are going to use comes from the ML UCI datasets. This can be found at `https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original)`.
It contains 699 instances for 10 attributes. This is the attribute information:

1. Sample code number: id number
2. Clump Thickness: 1 - 10
3. Uniformity of Cell Size: 1 - 10
4. Uniformity of Cell Shape: 1 - 10
5. Marginal Adhesion: 1 - 10
6. Single Epithelial Cell Size: 1 - 10
7. Bare Nuclei: 1 - 10
8. Bland Chromatin: 1 - 10
9. Normal Nucleoli: 1 - 10
10. Mitoses: 1 - 10
11. Class: (2 for benign, 4 for malignant)

1.3 *The `bnlearn` package in R*

The main package for R we will use is `bnlearn` , a package written by Marco Scutari. It has many features for both structural and parameters learning of Bayesian network from data. We refer to `www.bnlearn.com` for Documentation and Installation in R. We use the code at lines 12-21 CODE to install and load this and all the other required packages.

2 DATA ANALYSIS

2.1 *Data exploration and clearing*

We could download the cvs data directly from the UCI website, but instead we preferred to use the R library `mlbench` that already has the data implemented. Install and load it with:

```
install.library("mlbench")
library(mlbench)
```

Then we import the data with

```
data(BreastCancer)
```

By running a basic statistical analysis we see that some data is missing: there are 16 missing values and they all corresponds to the 'Bare Nuclei' attribute. The amount of data missing is quite small when compared to the dataset: becuase of this we could either disregard all rows where there is a missing value or try to impute them using an algorithm like the Expectation-Maximisation [3]. We decided to try three distinct approaches: the first one is simply omitting the 16 rows corresponding to the missing values. This has been done at line 57 CODE. The other approach is to use Amelia II, a popular statistical package in R, to impute the missing values [3]. We should point out that this is potentially wrong since our dataset is not assumed to be continuous while the methods of imputation in Amelia assume this. Nonetheless it has been shown in the literature [5],[6] how the imputation is in many cases successful regardless of the kind of data so we have tried it anyway in lines 62-77 CODE. The third and last approach we used is to make use of *bnlearn* built-in structural.em method that uses the Structural EM algorithm to find the best structure from missing values(and parameters,but we are only interested in the structure for this first step). This can be seen in the code at lines 80-84 CODE.
So because of this we will always work in what follows by applying our algorithms to three different datasets: the "omit" one (where we omitted the rows with a missing value) , the "imputed" one where we have used Amelia to do the imputation and finally "imputed2" which uses bnlearn builtin method to reconstruct missing data. Even if it is not required, it is usually beneficial to somehow visualise the data we are working with to have a better understanding of how we could learn interesting features from it. So we used a basic mixed correlation plot at lines 87-89 CODE as found in the *corrplot* library. We obtain the following plot:

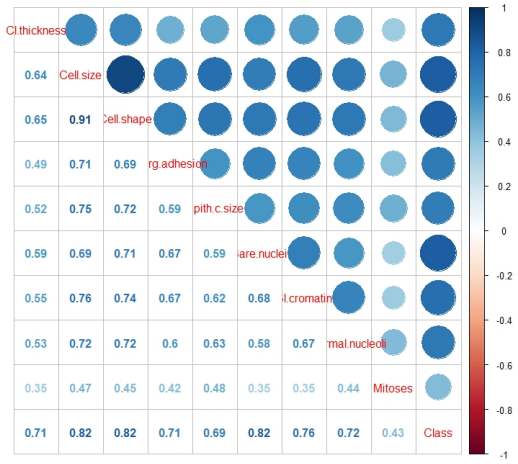

Figure 1: A correlation plot of the variables in our dataset

We see from the plot 5 that none of the variables seem to be statistically insignificant in terms of the 'Class' variables, with the properties "Bare.nuclei", "Cell shape" and "Cell size" being the most correlated and "Mitoses" being the least. We also see that in general the parameter "Mitoses" doesn't correlate much with any of the other attributes and that , not surprisingly perhaps, "Cell.size" and "Cell.shape" are highly correlated.

We end this preliminary step by splitting our omit and two imputed datasets into a training and test set. Here we picked a 80-20 split.The splitting happens at lines 92-106 CODE of the code. It would also make sense to try to learn the best structures by means of a k-fold cross validation but since our aim is just to investigate the main features of graphical models we do not go into this.

2.2 *Structural Learning*

As a first step, we want to use our data to somehow inference an optimal graph that represent the probabilistic relation between all the variables. `bnlearn` for example implements various algorithms for structural learning. With Bayes Network there are two main methods for structural learning: score based learning and constraint based structural learning, and we have both kinds of algorithms readily implemented in `bnlearn`. For constraint based learning these are the algorithms implemented in `bnlearn`:

1. Grow-Shrink (GS)
2. Incremental Association Markov Blanket (IAMB)
3. Fast-Incremental Association (Fast-IAMB)
4. Interleaved Incremental Association (Inter-IAMB)
5. Max-Min Parents & Children (MMPC)
6. Semi-Interleaved Hiton-PC (SI-HITON-PC)

whereas for score based learning we have:

1. Hill Climbing (HC)
2. Tabu Search (Tabu)

It would be too technical to go into the details of each algorithm and we can refer to the specific papers were they were first proposed. Instead we give a theoretical overview of how structural learning works for score based learning [8]: as the name suggests, we define a certain score function on the

set of all possible pairs of graphs and data $(\vec{G}, \mathscr{D})$. This generically takes the form:

$$\text{Score}(G|\mathscr{D}) = LL(G|\mathscr{D}) - \phi(|\mathscr{D}|)|G|$$

where $LL(G|\mathscr{D})$ refers to the log-likelihood of the data $\mathscr{D}$ to be observed under the graph structure $\vec{G}$. It should be noticed here that the second term in this equation is simply to be considered as a regularization argument that prevent the model from over-fitting (and hence always learn a complete graph). Having defined a score function we can now use different methods of searching through the search space: the Hill-Climbing one is the classical one of following a greedy approach where we move along the direction of greatest ascent. Tabu search is also another kind of local search as HC but here we allow to take sometimes some worsening moves as long as there is no improving moves. The name Tabu refers to the fact that the algorithm temporarily marks as "tabu" (i.e. forbidden) some moves if they have already been taken in a certain short time.
On the other hand constraint based algorithms in `bnlearn` are all different optimized derivatives of the so called Inductive Causation Algorithm (IC) [7]. These in general tends to do very well with large datasets, so we don't expect them to perform really well out of the box on our data, since the Wisconsin Breast Cancer can be considered quite small. The basic idea of the IC algorithm is to first find all possible pairs of variable that are dependent of each other. The specifics of the statistical methods used to determine this is what gives us the variety of structural algorithms. In the next step we "prune" the structure, that is we remove as many indirect dependencies between variables as possible: this makes sense, since we want to have a model that is as simple as possible whilst explain the causal relationship between the random variables. In the final step we determine the best direction to assign to the edges in the graph. Is important to notice here how structural learning is essentially an undirected graph process and therefore most of the algorithms we are using here could be used for the Markov network case without much difference (except for allowing cycles, of course).

In the code at lines 116-136 CODE we define a function *struc.learn* that applies several of the above mentioned structural learning algorithms to the dataset given as a argument. Then in lines 141-150 CODE we apply this to the three training sets we have defined earlier. The package `bnlearn` has various builtin utilities to test and compare different learned structures. But before investigating some of them in details we preferred to plot all of them and have a first comparison of the different models in a visual manner. These plots can be seen in the three full scale pictures at the end. Figure 6, Figure 7, Figure 8.

From scanning the three different images we see that the three different datasets(omit,imputed and imputed2) behaved differently only in terms of the `tree.bayes` algorithm while it stayed the same in all other cases: this is not too weird to see considering that in every case we only changed or removed 16 rows out of 699. It is also clear that the structural learning algorithms (gs, iamb, fast.iamb, mmpc, si.hiton.pc) all returned structures that are suspiciously too simple to represent the causal relationship between variables: perhaps a tweaking of some of the parameters like the `test` parameter that changes the conditional independence method (the default one is the *mutual information*) or the `alpha` parameter, the error rate, might have resulted in better structures, but we do not explore this further since the dataset we are working with is quite small so we know constraint based methods are not really good.

We also observe that both the Hill Climbing and the Tabu search algorithm gave us the same model (for the three training sets). Following the same notation as in 1.1 the model joint probability is summarised as follows:

$$[Cell.size][Class|Cell.size][Cl.thickness|Class][Cell.shape|Class][Marg.adhesion|Class]$$
$$[Epith.c.size|Class][Bare.nuclei|Class][Bl.cromatin|Class][Normal.nucleoli|Class][Mitoses|Class]$$

This corresponds graphically to the following graph:

Figure 2: The structure learned via HC or Tabu

We can say that the model, without further expert knowledge, seem to at least make sense and can be considered as probably a good model: it states that the 'Class' variable only depends on the 'Cell Size' variable and that in turn the 'Class' is the only dependency for all the remaining variables. We may ask if perhaps the role of the arrow should be reversed: this may as well be possible but lacking any sort of medical knowledge we cannot say anything about this and the model would be anyway equally as predictive.

Since our main purpose with the Breast Cancer Dataset is to classify a given sample as being "benignant" or "malignant", of special interest are Bayesian Classifiers algorithms, whose main goals is that of finding the best predictive structure (in terms of classification accuracy) rather than necessarily the best causal or probabilistic dependency between variables.
The two Bayesian classifiers used in our code are the Naive Bayes and the Tree Bayes. A Naive Bayes classifier is simply the model that assume that the target variable (in this case 'Class') is directly affecting all or some of the remaining variables, called the explanatory variables. It is a very simple model , nonetheless it seems to be very good for our dataset as we will see later. The Tree Bayes dataset is certainly more involved: the specific algorithm used is called Tree-Augmented Naive Bayes algorithm (TAN). The idea is to relax the independence assumption between the explanatory variables by allowing them to have another parent attribute, apart from the target variable, and In a way that the structure is a tree. We see that we learned a slightly different structure for the three training set.
At line 174 CODE we use the `graphviz.compare` included in `bnlearn` to visualise the difference between the tabu.bayes and naive.bayes structures. From the plots we see that a lot of the added edges to the naive structure are in common with all the tree plots, suggesting that there is probably a strong correlation between these nodes that could be investigated further by an expert.

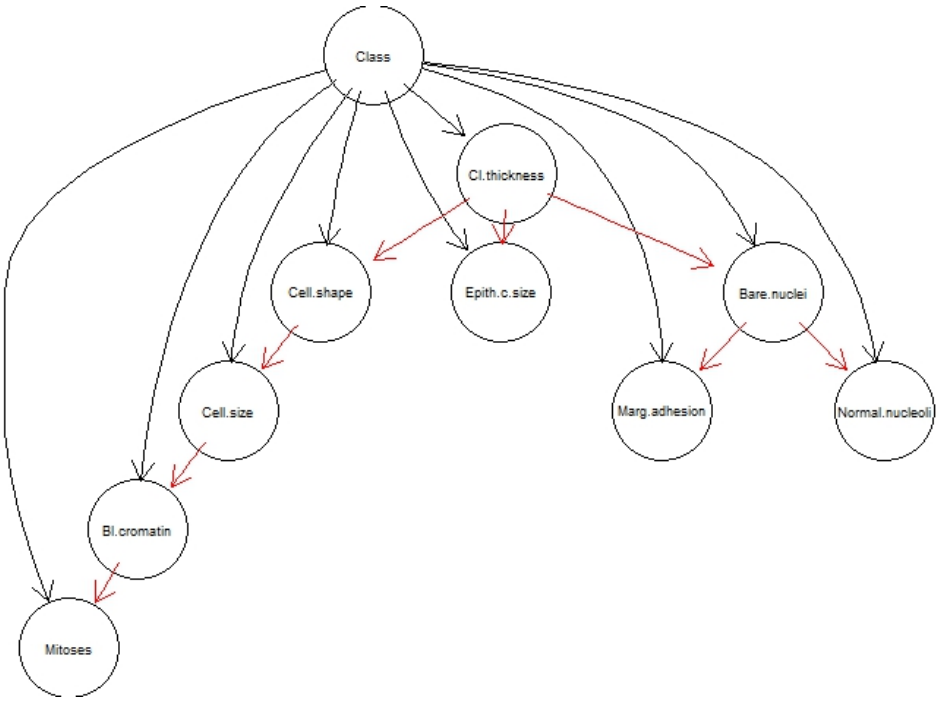

Figure 3: The comparison plot between the structure of the naive bayes classifier and the one of the tree bayes classifier

3 PARAMETER LEARNING

Having found some possible structures, we now move on to one of the most important part of Bayesian Network training: parameter learning. Parameter learning means finding, for discrete data, the best probability distributions for the nodes in the structure conditionally dependent on its parents. Written more formally, we have a collection D of data with $D = \{d_1, \ldots, d_n\}$. We look at these data as a realisation corresponding to the n random variables $X = \{X_1, \ldots, X_n\}$. We have a universe set Θ of all possible parameters. By Bayes Theorem 1 , we have that:

$$P(\theta|D) \propto P(D|\theta)P(\theta)$$

There are essentially two algorithmic approaches for doing parameter learning and both of these are also implemented in `bnlearn`. We remark here that we are restricting to the case of discrete datasets, since this is our case with the Wisconsin Breast Cancer dataset. The first method is the Maximum Likelihood: ideally the approach we would like to take is the so called Maximum a Posteriori approach which means:

$$\theta_{MAP} := \arg\max_{\theta \in \Theta} P(\theta|D)$$

But for most cases this is practically unfeasible. Instead we assume we have no prior knoweldge on the parameters, which is assuming the priors $P(\theta)$ are constants. Then by the equation from before we obtain the Maximum Likelihood Approach, which is we maximise the likelihood of seeing our data D under the assumption that the parameters are θ. This means the following:

$$\theta_{ML} := \arg\max_{\theta \in \Theta} P(D|\theta)$$

The other approach is the Bayesian parameter estimate: this means that we try to approximate the posterior probability distribution given by Bayes rule: this is the purely Bayesian way of treating uncertainties as random distribution where we use our initial knowledge to decide on the priors and

then we leverage Bayes Rule to compute the posteriors. Given an acylic structure graph `dag`, the command to do parameter learning in `bnlearn` is `bn.fit(dag)`. The optional parameter `method` can be either `mle` or `bayes` for respectively Maximum Likelihood Estimation or Bayes estimation. Since all the structures we learned in the previous section where we used constraint based algorithms did not have a direction, we first had to assign a direction to all of these. This is done in the code at lines 184-191 CODE where we decided somewhat arbitrarily to use the ordering derived by the Tabu search algorithm on the imputed dataset. The actual parameter learning, done in both ways and for all three training sets is done in lines 211-219 CODE. Now these list will contain the different conditional probabilities tables associated to each graph and we can do all sorts of manipulation or visualisation with them. As an example we can look at the table for the Tabu structure using the `imputed2` data with respect to the node `Cl.thickness`. This can be done in `bnlearn` with the following command:

```
bn.fit.barchart(bn.fit.imputed2$tabu$Cl.thickness)
```

The resulting plot we end up with it is:

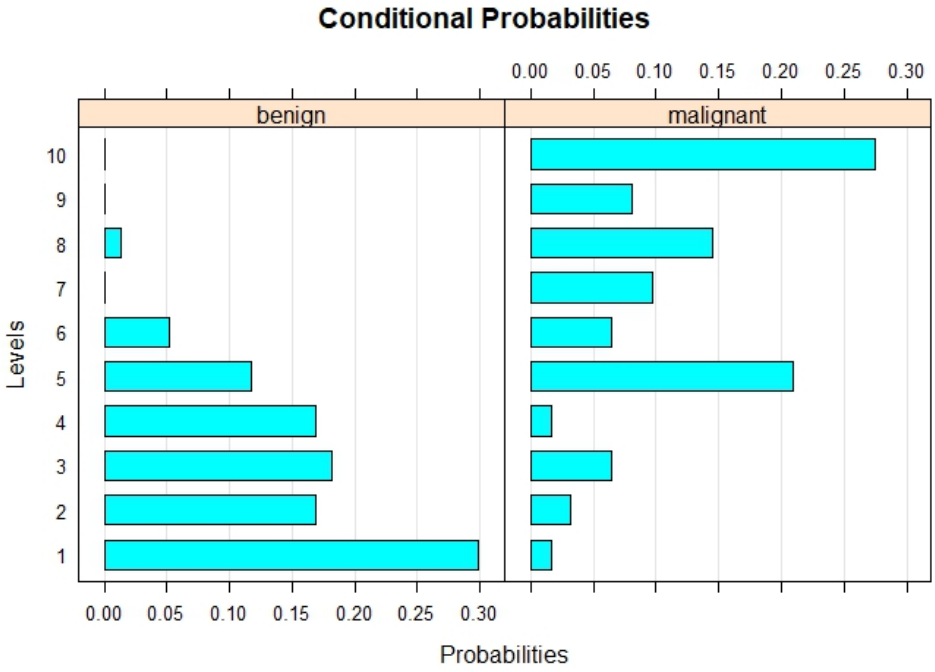

Figure 4: The Conditional table linked to Tabu

from which it is quite clear that the `Cl.thickness` has a clear correlation with the target variable `Class`, with smaller values of the parameters suggesting that the tumor is benignant.

4 MODEL SELECTION

Now we have various possible models that explain the relation between the variables in our datasets for the three different versions of it (omit, imputed and imputed2). We will now do various check of several parameters like accuracy and precision to decide on a preferred model and to see what models has the best predictive power. In the lines 242-270 CODE we have computed various parameters that are measuring the different models we have learned in the structural and parameter learning phase , each of them applied to the three different test sets (imputed2,omit,imputed). Moreover we also computed the "Confusion Matrix": this is a matrix that displays the

number of True Negatives, False Positives, False Negatives and True Positives and gives us a good metric for the performance of the algorithm on the test set. Clearly in our case, since we are dealing with a medical dataset we are not only interested in Accuracy, but also we would ideally have as least false negatives as possible since this are the most dangerous case in a practical setting: having many false negatives would mean that our model is in a way ”reassuring” the patient by telling him it has a benign tumour when in reality he has malign tumor! We decided to print out all the results we obtained. These can be examined in detail inside R output screen. Since we had an already big number of models applied to six different kind of datasets in each case, we summarise here all the result that we have found from this sistematic search. In general we noticed how the constraint based algorithms (gs,iamb,..ecc) all did really badly as we expected, considering the relatively small size of our dataset. In fact it many cases the model seemed to have just assigned a flat target of ”benign” to all our test cases, making it completely useless as a model. Also it generally looks like the distinction in accuracy between the omit dataset and the two imputed datasets was not massive although we gained a bit in accuracy with most algorithms, but interestingly enough the best model on the train and test set we have used was coming from the omit one. Using the ”bayes” method in parameter learning also seemed to have made a distinction in terms of accuracy.

The models that seemed to have done the best in general in terms of accuracy are the naive bayes algorithm and the score based algorithms. The most accurate model is the naive.bayes classifier with parameter learned via the ”bayes” method for parameter learning: this achieves a very high accuracy of 97.4 percent. Obviously we need to make sure that our model has not overfitted but we not have reasons for believing so. The confusion matrix associated to this model is the following:

Prediction	benign	malignant
benign	360	14
malignant	21	165

Although not spectacular, this model is already quite powerful in terms of accuracy and shows the power of graphical methods. It is quite possibly that we could still improve the model, especially when dealing with false positives (14 is quite an high number when compared to 699): perhaps the use of methods like cross validation could have been helpful or model averaging [9]. Since the focus of this paper is mostly to show how Graphical methods are used in medicine we do not investigate this further. We also notice that in general the tree bayes algorithm for structural learning doesn't seem to have done very well: evidently a more complex model has not helped in terms of accuracy.

5 BAYESIAN NETWORK INFERENCE

Now that we can assume that we have a working Bayesian network model we would like to use it to perform various kind of ”queries” on the model. This is called Bayesian inference and can be defined as the process of inferring the value of a set of variables by using the state of some other variables as evidence. What follows is taken from [1][2]. This is the power of graphical methods in general: we can get a probabilistic estimate of the value of some certain variables even in the case when this variable maybe ”hidden”, i.e. if it is expensive or too difficult to calculate in a real world setting. The process of inferencing is often also called ”belief updating” or ”probabilistic reasoning”. To define rigorously this process, assume we have a have learned a bayesian network B with graph structure G and parameters Θ that explains some joint probability X. Then we have some evidence E on the distribution of the joint probability of X and the process of inferring from this evidence is nothing but compute or approximate the value of the posterior probability

$P(X|E,B) = P(X|E,G,\Theta)$. Theoretically our evidence E can be of two different kinds :

- Hard evidence: this is evidence of the form

  $$E = \{X_{i_1} = x_{i_1}, \ldots, X_{i_n} = x_{i_n}\} \text{ for } \{i_1, \ldots, i_n\} \subseteq \{1, \ldots, n\}$$

  Clearly this is the kind of evidence we have when we obtain some new information about our variables. Often in this case we have already learned the Bayesian network.

- Soft evidence: this is evidence of the form

  $$E = \{X_{i_1} \sim (\Theta_{X_{i_1}}), \ldots, X_{i_n} \sim (\Theta_{X_{i_n}})\} \text{ for } \{i_1, \ldots, i_n\} \subseteq \{1, \ldots, n\}$$

  i.e. when we are given information on the kind of distribution that some of the variables follows. Often we take this evidence as new variables.

Since our dataset is discrete we will mainly be interested in queries related to hard evidence, since this is strictly related to the kind of problem we would like to address with our dataset: for example we would like to know how likely it is for a patient to have a malign breast cancer if we know that as a consequence of a test it has ”Clump.Thickness” equal to 6 and ”Bare.Nuclei” equal to 3. Following the distinction as shown in [1] we have two main distinct class of queries that we can perform:

- Conditional probabilities queries (CPQ):
  This is the case where we want to know the posterior probabilities of a finite set Q of variables given a certain hard evidence E on some other variables of X. In most cases it is assumed that the two sets of variables are disjoint. In our case of discrete variables we want to compute:

  $$\text{CPQ}(Q|E,B) = P(Q|E,G,\Theta)$$

  This can be obtained exactly by marginalisation of the variable Q:

  $$P(Q|E,G,\Theta) = (X|E,G,\Theta)d(X/Q)$$

- Maximum a posteriori queries (MAP):
  This is the case when we would like to find the optimal configuration $q*$ for the variable in Q such that the posterior probabilty is maximised. Specifically:

  $$MAP(Q|E,B) = q* = \arg\max P(Q = q|E,G,\Theta)$$

  i.e. when we are given information on the kind of distribution that some of the variables follows.

It is evident that in general the problem of computing exactly these conditional probabilities is quite intractable , especially when dealing with large networks or when dealing with small probabilities. Of course we can exploit the local nature of Bayesian Networks (and more generally Graphical Models) to simplify computations. For example, the marginalisation would become:

$$\begin{aligned} P(Q|E,G,\Theta) &= (X|E,G,\Theta)d(X/Q) \\ &= \int \left(\prod_{i=1}^{n} P(X_i|E, Pa(X_i), \Theta_{X_i}) \right) \\ &= \prod_{i:X_i \in Q} \int P(X_i|E, Pa(X_i), \Theta_{X_i}) d(X_i) \end{aligned} \tag{1}$$

Even by exploiting this and other local separation rules (for example ”d-separation” which is a particular kind of conditional independence) it

is still difficult in general to compute these values. This is why we often distinguish between exact and approximate inference. There are two main algorithms that are used to do exact inference: variable elimination, which just means storing some intermediate values to avoid recomputations, and junction trees, which is an algorithm that more or less try to cluster some variables to turn them into a tree.

For approximate inference the idea is that of using Monte Carlo simulations to sample from the local distributions and use that to estimate $P(Q|E, G, \Theta)$: in practice we generate a great number of samples from the bayesian Network B and then we just take the ratio of the samples including the evidence E and $Q = q$. The package bnlearn implements only approximate inference through the command "cpquery". At lines 279-285 CODE we compute the probability that a patient who was measured to have a "Cell.size" of 4 and a "Bare.nuclei" of 2 has a malignant theorem. In the code we had to do 2000 computations, save them into a list and take the mean in order to overcome the fact the method "cpquery" is stochastic and will result in slight different value in each iteration, whereas by taking the mean of multiple trials should get closer to the actual result. We see now how this sort of queries can be helpful for a doctor: he can asks all sort of questions about the relationship between some measurements he makes by means of exams and get an estimate of the probability of that event he is investigating. In particular there are mainly two kinds of inference we can do in general:

- Predictive: this is when we measure some parameter and we want to predict the value of another variable This is the classic case of when we want to predict if the tumor is benignant or malignant, in relation to our dataset.
- Diagnostics: this is when we have the outputs and we want to investigate the inputs. In our dataset we could for example ask for the most likely value of "Cell.size" if already know that the tumor is malignant.

It is clear how both kind of inference can be very valuable in a medical setting to analyse and take informed decision about some exam or illnesses or just in general to get a better understanding of a medical phenomenon.

6 UNDIRECTED GRAPHICAL METHODS AND MARKOV NETWORKS

So far we have been mostly being work with graphical methods were we arbitrarily imposed the constraint that our underline graphical structure was an acyclic directed graph: clearly in many occasions this is a sensible model to pick if we are trying for example trying to approximate the causal relationship between some probabilistic events, since ideally every "cause" has a direct "consequence" and we hope that the consequence directly or indirectly influence the root cause. This is immediately roots out all sort of situations where there is a so called "feedback loop", which is a process that can influences himself. A lot of processes in reality though, especially dynamical ones, do tend to have one more feedback loops built in and so we would probably need to consider more general graphical models that also allow and undirected structures and possibly with cycles, which is why we use Markov Random Fields, which were introduced at the beginning. Markov Random Field, being more flexible, are also much more difficult to optimize since the parameter space is more vast. It should also be noticed that the functions θ relative to each node can not be interpreted anymore as clearly as before as conditional probability distributions, so in a sense we also loose some intuitiveness when working with undirected models. Unfortunately from our research there weren't many libraries that did structural and parameter learning for undirected graph models and of the existing ones, many focused on dataset with a huge number of data points or where the data was assumed to be a time series. The best package that fitted our needs was 'BDgraph'[10][11]. This package uses Birth-Death

MCMC algorithm to learn a structure for our graph. It should be noticed here as well that a lot of the algorithms that we have described in the section related with Bayesian Networks did not strictly require the assumption that the underlying structure was acyclic and so apply also in the case of Markov networks. At line 291 CODE we therefore use BDgraph to learn an undirected structure. At line 295 CODE we also use the Marginal Pseudo Likelihood algorithm to learn the Markov network structure. The graph has less edges than the previous one. For both the structures in lines 292,296 CODE we printed to the R output a summary of the graphs that gives also all the information about the posterior "probabilities" of all the links (this is stored in the plinks upper triangular matrix).

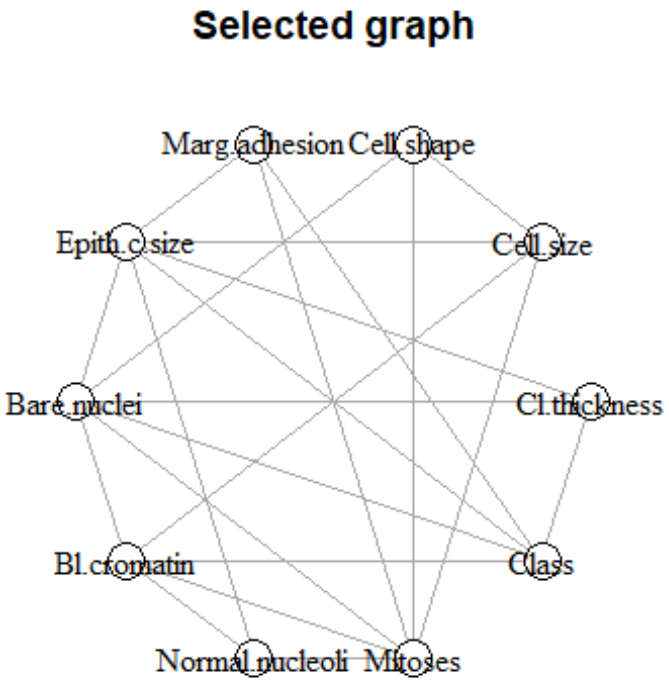

Figure 5: The undirected graph structure learned from the BDgraph algorithm

The result doesn't look very convincing: this is because we probably didn't use many iterations of the sampling algorithms so we did not converge to a near optimal solution. Also our dataset is relatively small so the methods used by BDgraph don't really apply well. Moreover we can conclude by saying that the assumption that the relationship between the features in our dataset can be described well with a DAG is actually not that restrictive and well motivated that these variables do not seem to come from a dynamical process so we don't expect to see any cycles in the underlying cycles. The cycles learned via this method could very possibly be just overfitting to the noise: we would need an expert opinion to see if the new relations learned between the variables could be somewhat justified and might give some deeper insight into the features.

Bibliography

1 Marco Scutari (2010). Learning Bayesian Networks with the bnlearn R Package. Journal of Statistical Software, 35(3), 1-22. URL http://www.jstatsoft.org/v35/i03/.

2 Marco Scutari, Jean-Baptiste Denis. (2014) Bayesian Networks with Examples in R. Chapman and Hall, Boca Raton. ISBN 978-1-4822-2558-7.

3 James Honaker, Gary King, Matthew Blackwell (2011). Amelia II: A Program for Missing Data. Journal of Statistical Software, 45(7), 1-47. URL http://www.jstatsoft.org/v45/i07/.

4 Nagarajan, Radhakrishnan, Marco Scutari, and Sophie Lbre. "Bayesian networks in R." Springer 122 (2013): 125-127.

5 Schafer, Joseph L. 1997. Analysis of incomplete multivariate data. London: Chapman Hall.

6 Schafer, Joseph L. and Maren K. Olsen. 1998. Multiple imputation for multivariate missing-data problems: A data analysts perspective. Multivariate Behavioral Research 33(4):545571.

7 Geiger, Dan, Thomas Verma, and Judea Pearl. "Identifying independence in Bayesian networks." Networks 20.5 (1990): 507-534.

8 Triantafillou, Sofia, and Ioannis Tsamardinos. "Score-based vs Constraint-based Causal Learning in the Presence of Confounders." CFA@ UAI. 2016.

9 Conrady, Stefan, and Lionel Jouffe. "Introduction to Bayesian Networks BayesiaLab." Bayesia SAS, USA (2013).

10 Mohammadi A. and E. C. Wit (2017-12-18). BDgraph: Bayesian Structure Learning in Graphical Models using Birth-Death MCMC.R package version 2.44.

11 Mohammadi A. and E. C. Wit (2017). BDgraph: An R Package for Bayesian Structure Learning in Graphical Models. arXiv preprint arXiv:1501.05108.

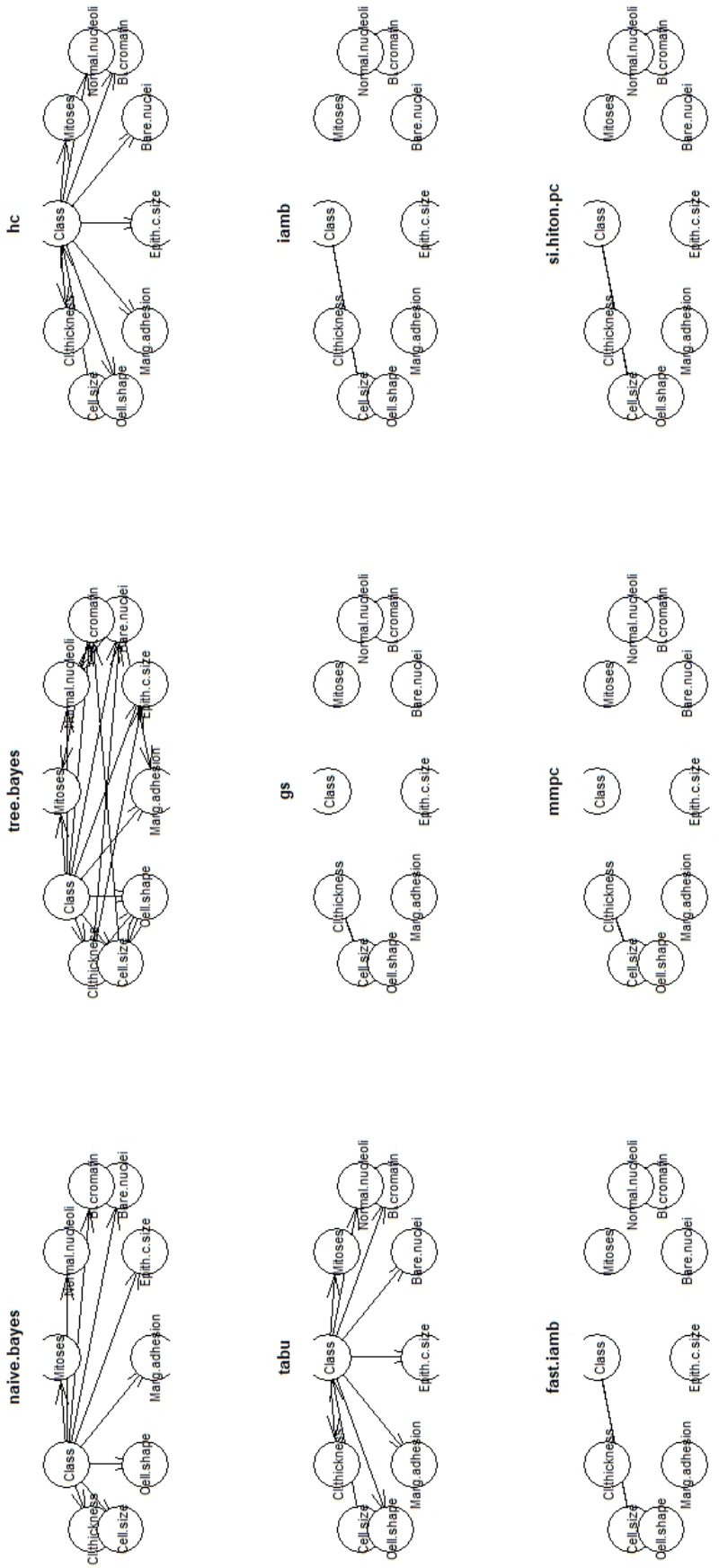

Figure 6: All the different structures learned on the 'omit' dataset

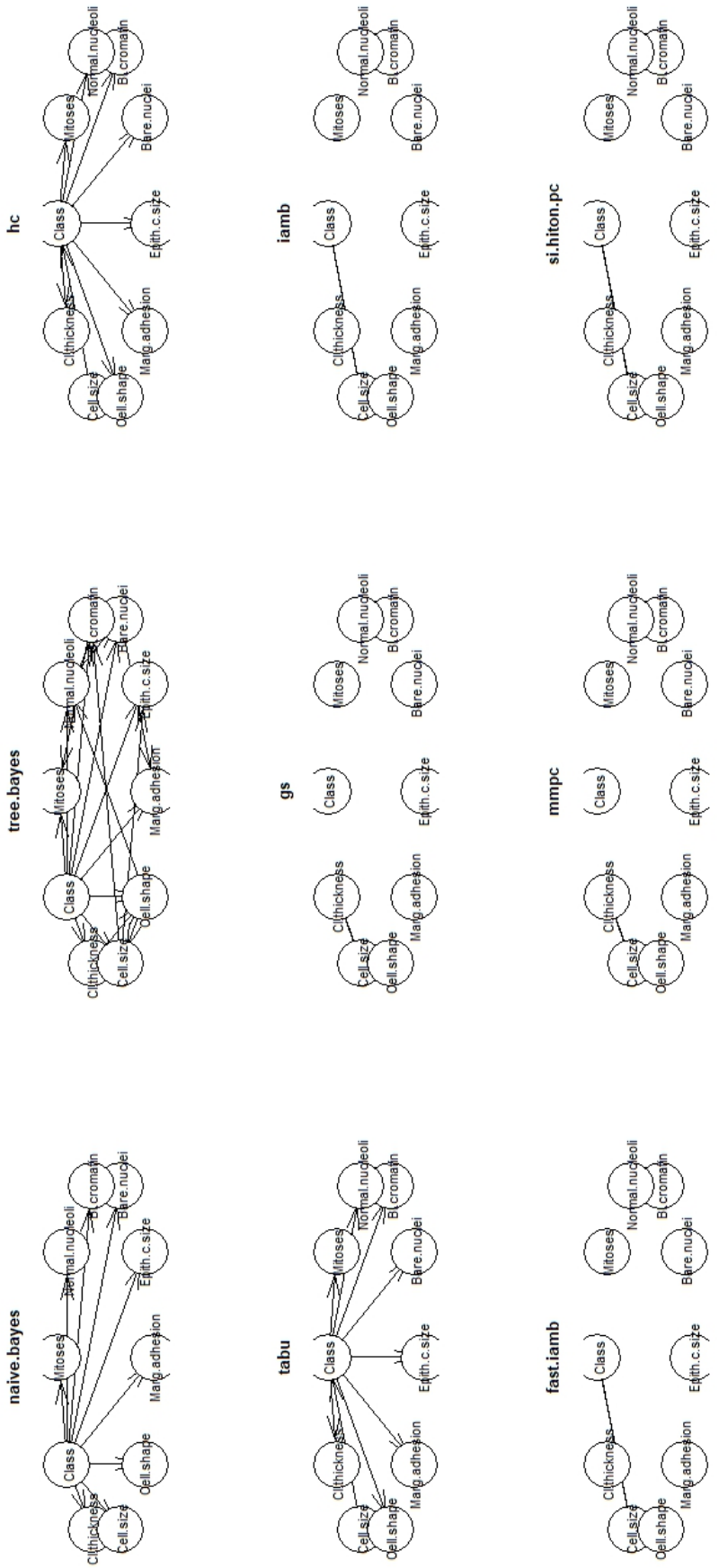

Figure 7: All the different structures learned on the 'imputed' dataset

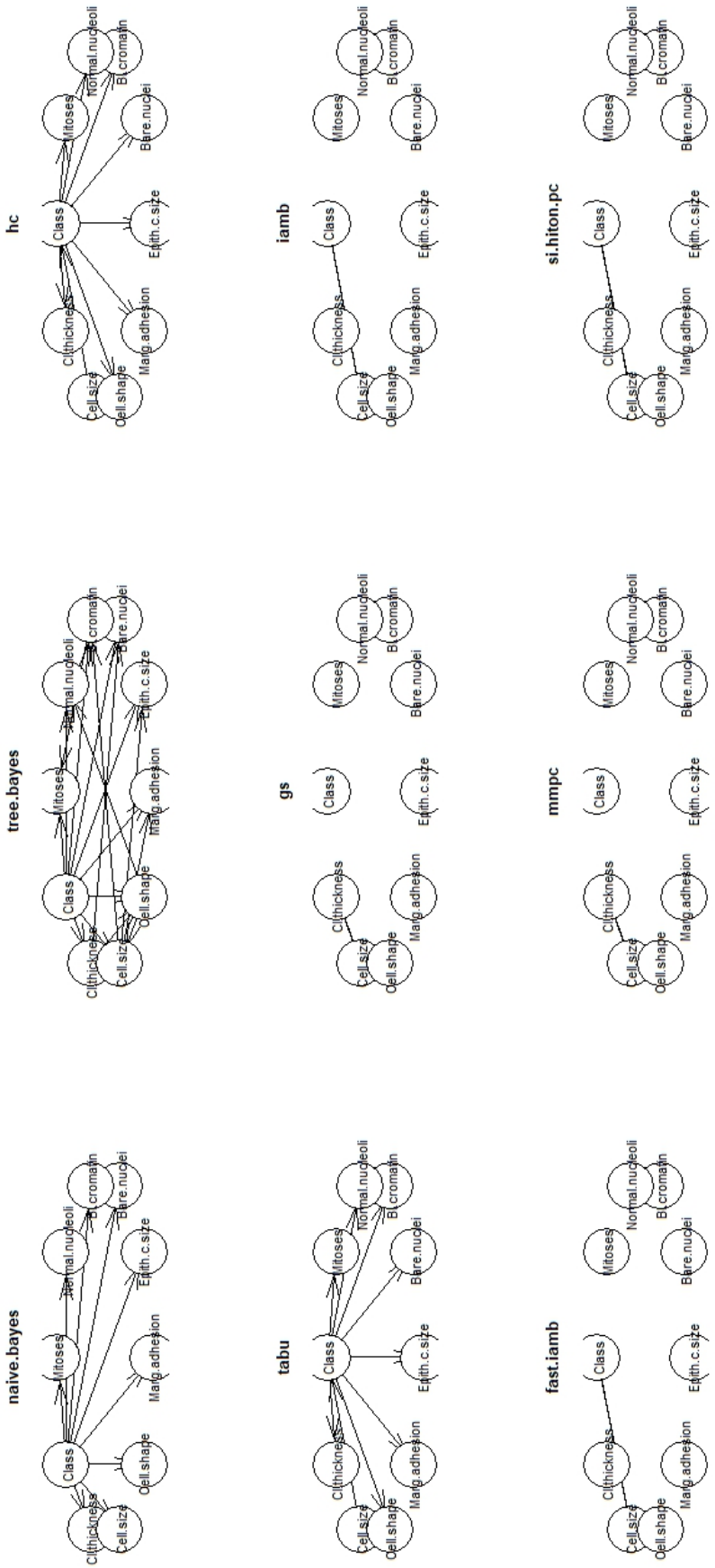

Figure 8: All the different structures learned on the 'imputed2' dataset