Assignment 1 - Bayesian Networks

Davide Pedranz, Mat. number 189295
davide.pedranz@studenti.unitn.it

***Abstract*—Bayesian networks are powerful graphical probabilistic models widely used if various fields. In this assignment, we compare some of the available algorithms to learn the structure of a Bayesian network from a dataset of samples.**

I. Introduction

Bayesian networks are graphical statistical models that represent the joint probability distribution and conditional dependencies of a set of random variables via a directed acyclic graph (DAG). They help to visualize the structure of the model in an intuitive way and allow all kinds of probabilistic inference on it.

A typical application of the Bayesian modelling is for diagnosis: given some clinical tests (observations), we want to estimate the probability of having some diseases (causes). In general, Bayesian networks can be used for any kind probabilistic inference. Common tasks are computing conditional independence between groups of random variables and computing the probability of a certain configuration given a set of observations.

Bayesian networks are usually build with the help of a domain expert, who knows some of the causality and independence relationships between the random variables, as well as some rough conditional probabilities of a random variables given its possible causes (the parents in the DAG).

Given a network, there are algorithms to learn the model parameters from training data, or, if all parameters are given, tune them to better fit the data. There are also algorithms to learn the structure of the network. In this assignment, we focus on learning the structure of the network using different algorithms and comparing their performances on a given learning problem. In particular, we use the `HUGIN Lite`[1] software to learn the structure of the network using two different algorithms:

- NPC
- Greedy search-and-score

Moreover, we compare the results with a Naive Bayes structure.

II. Learning problem

We want to understand the relationships of 6 given genes: `ATP2B4`, `NAP1L1`, `MDK`, `PCCB`, `MDS1`, `AML`. Each gene can be active or inactive and can influence the status of the others. We are interested in predicting the probability of gene `AML` to be active, given the status of the others.

First, we will learn a Bayesian network using one of the algorithms described in Section I. Then, we will use the build in expectation–maximization (EM) algorithm to learn the model's parameters. Finally, we will evaluate the network's performances on predicting gene `AML`.

We are given dataset that contains 72 complete examples of the 6 genes. We will use 80% of them for the training and the remaining 20% for the test. Given the small number of examples, we will use the k-fold cross validation technique (explained in Section III) to get more reliable results.

III. k-fold cross validation

A typical approach to evaluate the performances of a supervised learning algorithm is to test its prediction of on some labeled test data. To do that, the examples are divided in two parts: a training set, used to train the learner, and a validation test, used to evaluate its performances. The learner never sees the test set before, so the results tells how good the learner will generalize on future data.

A common setting is to use $70 - 80\%$ of the data for the training and the remaining $20 - 30\%$ for the test. This works fine as long as the available dataset is big enough, but the learner has problems if the dataset is small, since this further reduces the number of available examples. In fact, the learner tends not to generalize properly if trained with insufficient examples.

k-fold cross validation is a technique that tries to reduce this problem. The dataset is randomly partitioned in k equal sized sets. Instead of just one, k different learners are trained, each time taking as training set the union of all partitions except the k$^{\text{th}}$ one. The k$^{\text{th}}$ partition is used to test the performances of the k$^{\text{th}}$ learner. To obtain the final prediction, the average of the predictions of the k learners is taken.

In this case, we choose $k = 5$, so that we always train the learners on the 80% of the dataset. The generation of the k subsets is implemented using the `scikit-learn`[2] library. The performances are evaluated on each of the 5 trained learners and averaged together for each of the 3 algorithms.

IV. NPC

The NPC algorithm tries to construct the network exploiting the conditional independence in the samples. It perform some statistical test for conditional independence for each pair of random variables. An undirected edge is added between each pair of variables which are not statistically independent, then the conflicts (eg. cycles) are resolved using some heuristic or the user input. Note that the algorithm is not able to decide the direction of causal relationships, since this is proven to be impossible.

[1]Available at: http://www.hugin.com/index.php/hugin-lite/

[2]http://scikit-learn.org/

Table I
NPC NETWORK'S PERFORMANCES

k	accuracy	recall	precision	f1
1	0.80	1.00	0.40	0.57
2	0.80	1.00	0.63	0.77
3	0.79	1.00	0.63	0.77
4	0.64	0.86	0.60	0.71
5	0.57	0.83	0.50	0.63
avg	0.72	0.92	0.56	0.70

Table II
GREEDY SEARCH-AND-SCORE NETWORK'S PERFORMANCES

k	accuracy	recall	precision	f1
1	0.80	1.00	0.40	0.57
2	0.73	1.00	0.56	0.71
3	0.86	1.00	0.71	0.83
4	0.64	0.86	0.60	0.71
5	0.57	0.83	0.50	0.63
avg	0.72	0.92	0.56	0.70

The algorithm is executed with the default settings. During the parameters learning with the EM algorithm, the experience is set to once “active” and once “inactive” for all genes, even if they are never observed active or never observed inactive. This forces the algorithm to never assign zeros probabilities to random variables, which would create problem to the inference process. Table I summarises the performances of the 5 networks learned using NPC.

V. GREEDY SEARCH-AND-SCORE

The greedy search-and-score algorithm performs a search through the space of possible networks. It evaluates the goodness of each network using a score function and returns the “best” one found. Network candidates are constructed by adding an arc, removing an arc or reversing an arc to the currently best scoring graph.

As for NPC, we use EM to learn the parameters of the learned network, setting all experiences to 1 to avoid null probabilities. Table II summarises the performances of the 5 networks learned using greedy search-and-score.

VI. FIXED NAIVE BAYES STRUCTURE

Naive Bayes networks assume a simplified structure for the dependencies between random variables.

Let's denote with Y the random variables we are interested in and with y_i its possible values. We call $x = \langle a_1, ..., a_m \rangle$ the conjunction of the other random variables in the network. Then, the decision rule used in normal Bayesian networks is:

$$y^* = \operatorname*{argmax}_{y_i \in Y} P(y_i|x) = \operatorname*{argmax}_{y_i \in Y} P(a_1, ..., a_m|y_i)P(y_i)$$

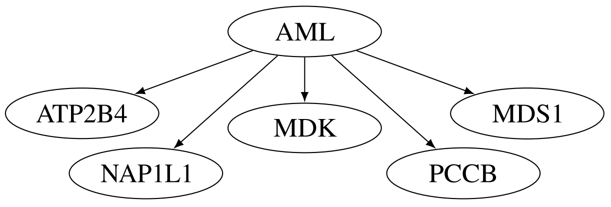
The first term $P(a_1, ..., a_m|y_i)$ is quite complex and require the knowledge of the joint probability distribution of all random variables in the network. Naive Bayes classifiers make the simplifying assumption that all random variables a_i are independent on each other given the class. Formally:

$$P(a_1, ..., a_m|y_i) = \prod_{j=1}^{m} P(a_j|y_i)$$

The decision rule of a Naive Bayes classifier can thus be simplified to:

$$y^* = \operatorname*{argmax}_{y_i \in Y} \prod_{j=1}^{m} P(a_j|y_i) \tag{1}$$

To make Equation (1) easy to compute, the network for a Naive Bayes classifier is made of the random variable to predict as the root node and all other random variables as leaves, connected only to the root. Since we are interested in predicting `AML`, the network can be represented as:



The network's parameters are trained using the EM algorithm, using the same trick of putting some expirience on all possible configurations as described in section Section IV for NPC. Table III summarises the performances of the Naive Bayes network.

Table III
NAIVE BAYES NETWORK'S PERFORMANCES

k	accuracy	recall	precision	f1
1	0.93	1.00	0.67	0.80
2	0.80	1.00	0.63	0.77
3	0.86	1.00	0.71	0.83
4	0.79	0.86	0.75	0.80
5	0.71	0.83	0.63	0.71
avg	0.82	0.92	0.68	0.78

VII. RESULTS

NPC and greedy search-and-score algorithm showed very similar performances, with an average accuracy of 72%. In both cases, recall is much higher than precision. This means that the network tends to classify negative examples as positive. On the other hand, it manages to correctly identify almost all positive examples.

The Naive Bayes network performed significantly better than both the networks learned with NPC and greedy search-and-score algorithms, reaching an average accuracy of 82%. Also, the network results to be simpler than the learned one.

There are two main possible explanation for this results. The first possibility is that most of the genes are independent of each other with regards to `AML`. In this setting, the independence assumption of the Naive Bayes approach is a good approximation of the reality, so the Naive network generalizes well on real data. The second possibility is that the training dataset was too small to learn a good structure. The learned structures models the noise of the sample data and do not generalize properly.

VIII. Conclusions

On this learning problem, the fixed Naive Bayes structure performed significantly better than the networks learned with both NPC and Greedy search-and-score. Naive Bayes classifiers should considered if the training set is small and thus does not contain enough information to learn a good structure. The network structure of a Naive Bayes classifier is simpler than the structure learned from the data and may perform better in some cases. If the random variables are known to be independent on each other given the class, then the Naive Bayes approach should be preferred.