Politecnico di Torino

Universidad Politécnica de Madrid

IDBN-MINING: DATA MINING IN KNOWLEDGE BASES BY GRAPHICAL MODELS EVALUATION

MSc Data Science and Engineering (POLITO)

Departamento de Inteligencia Artificial (UPM)

Candidate:

Piero Ciffolillo

Supervisor:

Juan Antonio Fernández del Pozo de Salamanca

Co-Supervisor:

Mauro Gasparini

MADRID, SPAIN

Alla mia famiglia, il cui sostegno mi ha donato libertà e forza d'animo, e ai miei amici, che hanno arricchito e allegerito questi anni meravigliosi.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Special thanks go to Professor Juan Antonio Fernández del Pozo for his excellent availability, for suggesting and following me continuously in this work.

Heartfelt thanks to Professor Mauro Gasparini, for being interested in my path right away and guiding me with professionalism and gentleness.

Their advice was fundamental to develop the work: thank you for being extraordinarily helpful.

ABSTRACT

Probabilistic graphical models (PGMs) represent a branch of machine learning that supplies an intuitive framework for describing the interactions within systems with simple abstraction. Reasoning using PGMs allows answering inference queries with uncertainty backing the support of probability theory. Two classical types of PGMs are the well-known Bayesian Networks (Directed Acyclic Graphs) and Markov Random Fields (Undirected Graphs).

A helpful extension of Bayesian Networks for decision-making is known as Influence diagrams (IDs). IDs are graphical decision models used for reasoning under uncertainty: solving an ID means determining the optimal strategy for the decision-maker and the value of the decision when that optimal strategy is applied.

PGMs can be used to construct knowledge-based systems as guidance for domain experts and analysts, in fields as diverse as medicine, language processing, vision. IDs can be keys for decision support. When dealing with complex systems, the outcome of model inference may not be intuitive. The ability to critically interpret the results is crucial, so developing adequate methods to interpret and explain is currently an ambitious and challenging task.

Much research effort has been devoted to develop methods and algorithms for PGMs, and a vast literature exists. At first, this work attempts a brief exposition on the state-of-the-art techniques used for inference in BNs and for solving IDs, along with their criticalities.

Besides exploring the main methods, the purpose of this work is the implementation of a package in Python, providing some of the algorithms studied and described, with personal attempts to improvement. Some insights related to interdisciplinary work follow, together with application examples of the developed software. Finally, strategies for explaining and visualize results are sought using data mining approaches.

CHAPTER 1
MOTIVATION: WHY PROBABILISTIC GRAPHICAL MODELS

Probabilistic modeling it is a fascinating scientific field bridging two very different branches of mathematics: probability and graph theory. It also has intriguing connections to philosophy, particularly to the issue of causality.

At the same time, probabilistic modeling is widely used throughout machine learning and decision making. In this introductory chapter, the main concepts that define the structure and properties of Bayesian Networks are formally presented. These concepts are essential to understand techniques for inference and their extensions to optimization problems.

The subsequent discussion on inference-related procedures will be divided into two main categories: exact methods and approximate methods[1].

1.1 Review of Bayesian Networks (BNs)

A Bayesian network is a probabilistic graphical model that measures the conditional dependence structure of a set of random variables based on the Bayes theorem:

$$P(A \mid B) = \frac{P(A)P(B \mid A)}{P(B)}$$

where A, B are random variables and P represent their densities. J. Pearl [1998] stated that Bayesian Networks are graphical models that contain information about causal probability relationships between variables and are often used to aid in decision making. The causal probability relationships in a Bayesian network can be suggested by experts or updated using Bayes theorem when new data is collected. The inter-variable dependence structure is repre-

1. Interestingly, algorithms described in this work are heavily based on work done in the statistical physics community in the mid-20th century.

sented by nodes (which depict the variables) and directed arcs (which depict the conditional relationships) in the form of a directed acyclic graph (DAG[2]).

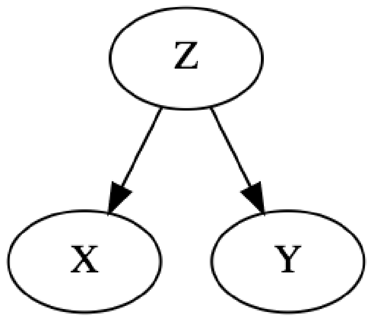


Figure 1.1: Simple Bayesian Network.

The notion of conditional independence is crucial. For example, node X is conditional independent from node Y given node Z in Figure 1.1 (not the same to say that they are independent in general). This is the same to state that:

$$P(X \mid Y, Z) = P(X \mid Z)$$

In order for a Bayesian Network to model a probability distribution, the following is true by construction:

Each variable is conditionally independent of all its non-descendants in the graph given the value of all its parents. This implies that, in general:

$$P(X_1 \dots X_n) = \prod_{i=1}^{n} P(X_i \mid Par(X_i)).$$

This work will mainly deal with DAG; however, it is appropriate to mention that a fundamental theorem concerning the conditions for which a probability distribution can be mapped on a graphical model has been developed by Hammersley and Clifford [1971]

2. It is a directed graph with no loop or self connection. A directed graph is a DAG if and only if it can be *topologically ordered*.

for undirect graphs (Markov random fields or MRF), and that in general there are useful conversion techniques from BNs to Markov models and vice versa.

In particular, to convert a BN it is *"moralized"*: the moralized counterpart of a DAG is formed by adding edges between all pairs of non-adjacent nodes that have a common child, and then making all edges in the graph undirected. This process is a key step for a particular inference method, the *junction tree* algorithm.

1.2 Hybrid Bayesian Networks

Bayesian Networks that handle both discrete and continuous nodes are said to be hybrid. There are two approaches to deal with continuous variables.

The first approach is to use parametric distributions within nodes that pertain to a continuous variable. This has two disadvantages (Halford [2020]). First, it is complex because there are different cases to handle: a discrete variable conditioned by a continuous one, a continuous variable conditioned by a discrete one, or combinations of the former with the latter. Secondly, such an approach requires having to pick a parametric distribution for each variable. There are methods to automate this choice for you, popular examples in the literature are Gaussian Networks (Lauritzen and Jensen [1999]) and Truncated Exponential Networks (Serafín Moral [2001]).

The second approach is to discretize continuous variables. Although this might seem naive, it is generally a good enough process and makes things simpler implementation-wise. There are many ways to go about discretizing a continuous attribute. For instance, you can apply a quantile-based discretization function. You could also round each number to its closest integer; in some cases, you might be able to use a manual rule. It is noticeable that also discretization techniques have drawbacks: if we need to store conditional probability tables with many predecessors and large domains, they have a huge usage of memory.

There are actually several different and much more sophisticated versions to the discretization approach. The method proposed by Kozlov and Koller [1997] performs the discretization by minimizing an upper bound of the Kullback-Leibler divergence between the true and the discretized density. The minimization is done dynamically on the join tree. This is, however, a computationally costly method that even requires a specific data structure. A more efficient approach is proposed by Neil and Marquez [2007]. Here, it is the individual posterior density that is discretized instead of dynamically discretizing the densities in the join tree. This approach to dynamic discretization can therefore be implemented on top of any discrete inference scheme.

To summarize, is preferable to give the user the flexibility to discretize the variables by herself. Most of the time, the best procedure depends on the problem at hand and cannot be automated adequately.

CHAPTER 2

EXACT INFERENCE IN BAYESIAN NETWORKS

Given a probabilistic model, typically we want to querying the marginal or conditional probabilities of certain events of interest. Concretely, there are two typical types of questions:

- *Marginal inference*: looks for the probability of a given variable in our model after we sum everything else out. For example:

$$p(x_1) = \sum_{x_2} \sum_{x_3} \cdots \sum_{x_n} p(x_1, x_2, \ldots, x_n).$$

- *Maximum a posteriori (MAP)* inference asks for the most likely assignment of variables. We may try to determine the most likely assignment given some evidence [3]

$$\arg \max_{x_1,\ldots,x_{n-1}} p(x_1, \ldots, x_{n-1}, x_n = 1).$$

When queries are involving evidence, the assignment are fixed for a subset of the variables.

Inference is a challenging task: for many probabilities of interest, it is NP-hard to answer these questions. Whether inference is tractable will depend on the structure of the graph that describes that probability: if a problem is intractable, is still possible to obtain useful answers via approximations (V. Kuleshov [2019]).

2.1 VE: The variable elimination method

2.1.1 Preliminaries: Brute force approach comparison

This chapter covers the first exact inference algorithm, *variable elimination* (Bertele and Brioschi [1972]). This technique is a special case of **dynamic programming**, a general

3. For evidence, we mean the information about the realization of certain variables.

approach to algorithmic design in which a larger problem is break into a sequence of smaller ones.

A chain example:

Consider the problem of marginal inference and suppose a chain BN, so a probability in the form[4]

$$p(x_1, \ldots, x_n) = p(x_1) \prod_{i=2}^{n} p(x_i \mid x_{i-1}).$$

Now compute the marginal probability $p(x_n)$. In a brute force approach, we sum the probability over all k^{n-1} assignments to $x_1, \ldots, x_{n-1}$:

$$p(x_n) = \sum_{x_1} \cdots \sum_{x_{n-1}} p(x_1, \ldots, x_n).$$

But one can do better by leveraging the factorization of our probability distribution, rewriting the sum in a way that pushes in certain variables into the product.

$$\begin{aligned} p(x_n) &= \sum_{x_1} \cdots \sum_{x_{n-1}} p(x_1) \prod_{i=2}^{n} p(x_i \mid x_{i-1}) \\ &= \sum_{x_{n-1}} p(x_n \mid x_{n-1}) \sum_{x_{n-2}} p(x_{n-1} \mid x_{n-2}) \cdots \sum_{x_1} p(x_2 \mid x_1) p(x_1) \end{aligned}$$

Now one can sum the inner terms first, starting from x_1 and ending with x_{n-1}. Concretely, one computes an intermediary **factor** $\tau(x_2) = \sum_{x_1} p(x_2 \mid x_1) p(x_1)$ by summing out x_1. This operation takes $O(k^2)$ time since one must sum over x_1 for each assignment to x_2. The resulting factor $\tau(x_2)$ can be interpreted as a table of k values, with one entry for each

4. We assume that x_i are discrete variables taking k possible values each.

assignment to x_2, as factor $p(x_1)$ can be also represented as a table. Then, is possible to rewrite the marginal probability using τ as

$$p(x_n) = \sum_{x_{n-1}} p(x_n \mid x_{n-1}) \sum_{x_{n-2}} p(x_{n-1} \mid x_{n-2}) \cdots \sum_{x_2} p(x_3 \mid x_2)\tau(x_2).$$

This has the same form as the initial expression, except that we are summing over one fewer variable. At each time, we are *eliminating* a variable, so this gives the algorithm its name. Another factor $\tau(x_3) = \sum_{x_2} p(x_3 \mid x_2)\tau(x_2)$ has to be computed, and repeat the process until we are only left with x_n. Since every step takes $O(k^2)$ time, and there are $O(n)$ steps, the inference process now takes $O(nk^2)$ time, that is definitively better than brute force $O(k^n)$ solution (S. Ermon [2019]).

2.1.2 Eliminate Variables

Now assume that there is a graphical model as a product of **factors**.

$$p(x_1, \ldots, x_n) = \prod_{c \in C} \phi_c(x_c).$$

In a BN, the factors correspond to conditional probability distributions (CPDs).

The VE algorithm repeatedly performs two factor **operations**: *product* and **marginalization**.

The factor product operation simply defines the product $\phi_3 := \phi_1 \times \phi_2$ as

$$\phi_3(x_c) = \phi_1(x_c^{(1)}) \times \phi_2(x_c^{(2)}).$$

The scope of ϕ_3 is defined as the union of the variables in the scopes of ϕ_1, ϕ_2; also $x_c^{(i)}$ denotes an assignment to the variables in the scope of ϕ_i defined by the restriction of x_c to that scope. For example, we define $\phi_3(a, b, c) := \phi_1(a, b) \times \phi_2(b, c)$.

The marginalization operation eliminates a set of variables from a factor. Given factor $\phi(X, Y)$ over two sets of variables X, Y, marginalizing Y produces a new factor

$$\tau(x) = \sum_y \phi(x, y),$$

where the sum is over all joint assignments to the set of variables Y.

τ refers to the marginalized factor. This factor does not necessarily correspond to a probability distribution, even if ϕ was a CPD.

The algorithm:

Essentially, we loop over the variables in a particular order O and eliminate them in that ordering.

Formally, for each variable X_i (ordered according to O),

1. Multiply all factors Φ_i containing X_i.
2. Marginalize out X_i to obtain a new factor τ
3. Replace the factors Φ_i with τ

2.1.3 Dealing with evidence

A common query type is in the form $P(X|E = e)$ where X and E are disjoint subsets of a probability distribution, and E is observed taking value e. We can compute this probability by computing first $P(Y, E = e)$ and then $P(E = e)$ with VE, since:

$$P(X \mid E = e) = \frac{P(X, E = e)}{P(E = e)}$$

2.1.4 Time complexity of Variable Elimination

The running time of VE depends heavily on the graph's structure.

Moreover, some orderings are more efficient than others. The running time of VE is $O(nk^{M+1})$, where M is the maximum size of any factor τ formed during the elimination process and n is the number of variables (V. Kuleshov [2019]).

Note that while n and k are parameters of the model, the value M is not: it depends on the variable elimination ordering. Thus, different orderings may dramatically alter the running time of the variable elimination algorithm.

Choosing VE order:

The VE algorithm requires an ordering over the variables according to which variables will be *eliminated*. In general, it is NP-hard to find the best ordering.

In practice, we rely on heuristics. The most common are:

- *Min-degree*: Next node with fewest neighbors.
- *Weighted-min-degree*: Next variable minimizing the product of the cardinalities of its neighbors.
- *Min-fill* Choose vertices to minimize the size of the factor that will be added to the graph.

2.2 Message passing algorithms

VE algorithm can answer marginal queries of the form $P(Y \mid E = e)$ for both directed and undirected networks.

This algorithm has a critical shortcoming: if we want to ask the model for another query, e.g., $P(Y_2 \mid E_2 = e_2)$, we need to restart the algorithm from scratch. This is very wasteful and computationally burdensome.

It turns out that this problem is also easily avoidable. When computing marginals, VE produces many intermediate factors τ as a side-product of the main computation; these factors turn out to be the same as the ones that we need to answer other marginal queries. By caching them after the first run of VE, we can quickly answer new marginal queries at essentially no additional cost, in $O(1)$ time. The idea was initially proposed by Pearl [1982], who formulated it as an exact inference algorithm on trees.

There are two variants: belief propagation (BP), and the complete Junction Tree (JT) method. BP applies to tree-structured graphs, while JT method applies to general networks.

2.2.1 Belief propagation

Consider to apply the VE algorithm on a tree to compute a marginal $p(x_i)$. We can easily find an optimal ordering for this problem by rooting the tree at x_i and iterating through the nodes starting from the leaves and goes up the tree such that a node is visited after all of its children.

At each step, we will eliminate x_j; this will involve computing the factor $\tau_k(x_k) = \sum_{x_j} \phi(x_k, x_j)\tau_j(x_j)$, where x_k is the parent of x_j in the tree. At a later step, x_k will be eliminated, and $\tau_k(x_k)$ will be passed up the tree to the parent x_l of x_k in order to be multiplied by the factor $\phi(x_l, x_k)$ before being marginalized out. The factor $\tau_j(x_j)$ can be thought of as a **message** that x_j sends to x_k that summarizes all of the information from the subtree rooted at x_j. So it is also called **message-passing**.

At the end of VE, x_i receives messages from all of its immediate children, marginalizes them out, and obtains the final marginal.

Now suppose that after computing $p(x_i)$, we want to compute $p(x_k)$ as well. We would again run VE with x_k as the root, waiting until x_k receives all messages from its children. It is noticeable the messages x_k received from x_j now will be the same as those received when

x_i was the root, and this is true since there is only a single path connecting two nodes in the tree. Thus, if we store the intermediary messages of the VE algorithm, we can compute other marginals as well.

The variant of the algorithm used for marginal inference is called *sum-product message passing*.

Sum-product:

The sum-product message passing algorithm is defined as follows: while there is a node x_i ready to transmit to x_j, send the message

$$m_{i \to j}(x_j) = \sum_{x_i} \phi(x_i)\phi(x_i, x_j) \prod_{\ell \in N(i) \setminus j} m_{\ell \to i}(x_i).$$

The notation $N(i) \setminus j$ refers to the set of nodes that are neighbors of i, excluding j. Again, observe that this message is precisely the factor τ that x_i would transmit to x_j during a round of variable elimination with the goal of computing $p(x_j)$.

Because of this observation, after computing all messages, is possible to answer any marginal query over x_i using the equation:

$$p(x_i) \propto \phi(x_i) \prod_{\ell \in N(i)} m_{\ell \to i}(x_i).$$

2.2.2 Junction tree algorithm

In general, the graph could not be a tree, especially when we are dealing with the moral graph of a DAG. However, we may try to transform the graph to its most tree-like form and then run message passing on this graph.

The junction tree algorithm partitions the graph into clusters of variables; internally, the variables within a cluster could be coupled; however, interactions among clusters will have a tree structure, so clusters neighbors will only directly influence a cluster in the tree. This leads to tractable global solutions if the local (cluster-level) problems can be solved exactly.

Another chain example:

Suppose that we are performing marginal inference on an MRF of the form

$$p(x_1, \ldots, x_n) = \frac{1}{Z} \prod_{c \in C} \phi_c(x_c),$$

It is crucial assume that the cliques [5] c have some path structure, meaning that we can find an ordering $x_c^{(1)}, \ldots, x_c^{(n)}$ with the property that if $x_i \in x_c^{(j)}$ and $x_i \in x_c^{(k)}$ for some variable x_i then $x_i \in x_c^{(\ell)}$ for all $x_c^{(\ell)}$ on the path between $x_c^{(j)}$ and $x_c^{(k)}$. This assumption as is called *running intersection property* (RIP).

We may again use a form of variable elimination to "push in" certain variables deeper into the product of cluster potentials.

This marginalization can also be interpreted as computing a message over the variables it shares with the neighbor: the RIP is important since it enables to do so.

Junction trees:

The core idea of the junction tree algorithm is to turn a graph into a tree of clusters that are amenable to the variable elimination algorithm like the above. Ultimately, we implement a form of message passing on the junction tree at a high level; this will be equivalent to variable elimination for the same reasons that BP was equivalent to VE.

Suppose we have an undirected graphical model G (if the model is directed, we consider its moralized graph). A junction tree $T = (C, E_T)$ over $G = (X_c, E_G)$ is a tree whose nodes

5. A clique is a subset of vertices of an undirected graph such that every two distinct vertices in the clique are adjacent.

$c \in C$ are associated with subsets $x_c \subseteq X_c$ of the graph vertices (i.e., sets of variables); the junction tree must satisfy the following properties:

- *Family preservation*: For each factor ϕ, there is a cluster c such that $\text{Scope}[\phi] \subseteq x_c$.
- *Running intersection*: For every pair of clusters $c^{(i)}, c^{(j)}$, every cluster on the path between $c^{(i)}, c^{(j)}$ contains $x_c^{(i)} \cap x_c^{(j)}$.

Even if we may always find a trivial junction tree with one node containing all the variables in the original graph, such trees are useless because they will not result in efficient marginalization algorithms (S. Ermon [2019]).

There are different ways for constructing good junction trees. On can do it by hand, when models will have a very regular structure, for which there will be an obvious solution. For instance in a grid clusters are associated with pairs of adjacent rows or columns. However, popular methods were developed by Shenoy and Shafer [1988] and bettered for the *Hugin* software by F. V. Jensen and Olesen [1990].

Optimal trees make the clusters as small and modular as possible; unfortunately, it is again NP-hard to find the optimal tree. For this very reason, I will return to this topic at the end of the Chapter on approximate methods.

CHAPTER 3

APPROXIMATE INFERENCE

When probabilistic models are quite complex, simple algorithms like variable elimination may be too slow. Many classes of models may not admit exact polynomial-time solutions at all, and for this reason, much research effort in machine learning is spent on developing algorithms that yield approximate solutions to the inference problem (V. Kuleshov [2019]).

There exist two fundamental families of approximate algorithms: sampling methods, which produce answers by repeatedly generating random numbers from distributions of interest, and variational methods[6], which formulate inference as an optimization problem.

3.1 Sampling methods

3.1.1 Preliminaries: Markov Chain Monte Carlo (MCMC)

Monte Carlo estimation:

Sampling from a distribution lets us perform many useful tasks, including marginal and MAP inference, as well as computing integrals of the form

$$\mathbb{E}_{x\sim P}[f(x)] = \int_x f(x)P(x)dx.$$

In many cases, this integral will be impossible to perform analytically; instead, we will approximate it using a large number of samples from P. Algorithms that construct solutions based on many samples (i.e. based on the law of large numbers) from a given distribution are known as Monte Carlo[7] (MC) methods.

6. Take their name from the calculus of variations, which deals with optimizing functions that take other functions as arguments.

7. The name refers to the Monte Carlo Casino in Monaco.

MC methods has a long history: as explained by Metropolis [1987], Enrico Fermi first experimented them while studying neutron diffusion in the 1930s, but he did not publish the work. Then, they were central to the simulations required for the Manhattan Project.

Monte Carlo integration is an important instantiation of the general MC principle. This technique approximates a target expectation with

$$\mathbb{E}_{x\sim P}[f(x)] \approx I_T = \frac{1}{T}\sum_{t=1}^{T} f(x^t),$$

where $x^1, \ldots, x^T$ are samples drawn according to p. It can be shown that the MC estimate I_T is an unbiased estimator for $\mathbb{E}_{x\sim p}[f(x)]$. Moreover $I_T \to \mathbb{E}_{x\sim p}[f(x)]$ as $T \to \infty$; in particular, the variance of I_T can be made arbitrarily small with a large enough number T of samples.

Review of stationary distribution for Markov Chains:

Given a discrete-time Markov Chain, on state-space S, and with transition matrix

$$T_{ij} = P(S_{\text{new}} = i \mid S_{\text{prev}} = j)$$

we know that if P_0 is the initial vector probabilities, after t steps we have $P_t = T^t P_0$.

The limit $\pi = \lim_{t\to\infty} p_t$, when it exist, is called a stationary distribution of the Markov chain. We will construct below Markov chain with a stationary distribution π that exists and is the same for all P_0; we will refer to such π as the stationary distribution of the chain.

A sufficient condition for a stationary distribution is called *detailed balance*:

$$\pi(x')T(x \mid x') = \pi(x)T(x' \mid x) \quad \text{for all } x$$

Sum both sides of the equation over x and simplify: π must form a stationary distribution. However, the reverse may not hold, and indeed it is possible to have MCMC without satisfying detailed balance (Durrett [2011]).

The high-level idea of MCMC will be to construct a Markov chain whose states will be joint assignments to the variables in the model and whose stationary distribution will equal the model probability P.

To construct such a chain, is important to understand when stationary distributions exist. That is true under two sufficient conditions:

- *Irreducibility*: You can get from any state x to any other state x' with probability > 0 in a finite number of steps.

- *Aperiodicity*: It is possible to return to any state at any time, i.e., there exists an n such that for all i and all $n' \geq n$, $P(s_{n'} = i \mid s_0 = i) > 0$

The first condition means no absorbing states, i.e., states from which we can never leave. Chains of this type are called *ergodics*, according Manning and Schütze [2008] definition.

3.1.2 Gibbs sampling and Metropolis-Hastings (MH)

The idea of MCMC algorithms is to construct a Markov chain over the assignments to a probability function P; the chain will have a stationary distribution equal to P itself; by running the chain a large number of times, we will thus sample from P.

At a high level, MCMC algorithms take as argument a transition operator T specifying a Markov chain whose stationary distribution is P, and an initial assignment x_0 to the variables of P. Then, perform the following steps:

1. Run the Markov chain from x_0 for B *burn-in* steps.

2. Run the Markov chain for N *sampling* steps and collect all the visited states.

Assuming B is sufficiently large, the latter collection of states will form samples from P. We may use them to produce Monte Carlo estimates of marginal probabilities. Finally, we may take the sample with the highest frequency and perform MAP inference.

Metropolis-Hastings:

Metropolis-Hastings (MH) origin is from an article by Metropolis [1953], introducing the algorithm for the case of symmetrical proposal distributions, and from Hastings [1970] that extended it to the more general case. It constructs a transition operator $T(x' \mid x)$ from two components:

- A transition kernel $Q(x' \mid x)$, also named proposal distribution, specified by the user
- An acceptance probability for moves proposed by Q, specified by the algorithm as

$$A(x' \mid x) = \min\left(1, \frac{P(x')Q(x \mid x')}{P(x)Q(x' \mid x)}\right).$$

At each step of the Markov chain, we choose a new point x' according to Q. Then, we either accept this proposed change (with probability α), or with probability $1-\alpha$ we remain at our current state.

The acceptance probability encourages to move towards more likely points in the distribution; when Q suggests that we move into a low-probability region, we follow that move only a certain fraction of the time.

In practice, the distribution Q is something simple, like a Gaussian centered at x. Given any Q the MH algorithm will ensure that P will be a stationary distribution of the resulting Markov Chain. More precisely, P will satisfy the detailed balance condition for the MH Markov chain.

To see that, observe that if $A(x' \mid x) < 1$, then $\frac{P(x)Q(x'|x)}{P(x')Q(x|x')} > 1$ and thus $A(x \mid x') = 1$. When $A(x' \mid x) < 1$, this lets to write:

$$A(x' \mid x) = \frac{P(x')Q(x \mid x')}{P(x)Q(x' \mid x)}$$

$$P(x')Q(x \mid x')A(x \mid x') = P(x)Q(x' \mid x)A(x' \mid x)$$

$$P(x')T(x \mid x') = P(x)T(x' \mid x),$$

which is actually the detailed balance condition. We used $T(x \mid x')$ to denote the full transition operator of MH (obtained by applying both Q and A). So, if the MH Markov chain is ergodic, its stationary distribution will be P.

Notice that P appears in the acceptance rate only as ratio, and that it is commonly available.

In a Bayesian setting, P is a posterior distribution that could be intractable with other methods. Since it is the objective of the estimation, it is called *target distribution.* In BNs inference, the targets will be the marginals.

Gibbs sampling:

The algorithm was described by Geman [1984], decades after the death of the physicist Josiah Willard Gibbs[8].

It is a widely-used special case of the MH method. Given an ordered set of variables $x_1, \ldots, x_n$ and a starting configuration $x^0 = (x_1^0, \ldots, x_n^0)$, consider the following procedure.

Repeat until convergence for $t = 1, 2, \ldots$:

- Set $x \leftarrow x^{t-1}$.
- For each variable x_i in the order we fixed:
    1. Sample $x_i' \sim P(x_i \mid x_{-i})$

8. The name is a reference to an analogy between the sampling algorithm and statistical physics.

2. Update $x \leftarrow (x_1, \dots, x'_i, \dots, x_n)$.

- Set $x^t \leftarrow x$

We use x_{-i} for all variables in x except x_i. It is often easy to performing each sampling step, since we only need to condition x_i on its Markov blanket[9], which is typically small. When updating x_i, its new value is immediately used for sampling other variables x_j.

Gibbs can be seen as a special case of MH with proposal $Q(x'_i, x_{-i} \mid x_i, x_{-i}) = P(x'_i \mid x_{-i})$. In this case, the acceptance probability simplifies to one.

Assuming the right transition operator, both Gibbs sampling and MH will eventually produce samples from their stationary distribution, which by construction is P.

It is easy to check when it is the case since, in practice, it is not difficult to ensure ergodicity.

MH within Gibbs:

It is widespread to use MH within Gibbs when concerning multivariate distribution, like joint probabilities in BNs. For pure Gibbs, full conditionals[10] are directly required for each variable to simulate. Instead, with a Metropolis step for a single variable in the main Gibbs loop is enough to set a proper proposal distribution, using the full conditional only to evaluate the acceptance rate.

Typically, the proposals are distributed $Normally$ or $Uniformly$, centered at the current point of the MCMC for that variable. One can choose different proposals for different variables or use the same parametric form for simplicity. The critical point is to set a *symmetric* proposal to reduce the computation in the ratio of the acceptance rate: if $Q(x' \mid x) = Q(x \mid x')$ the acceptance rate will depend only on the target distribution ratio.

9. In a BN, the Markov blanket of a node includes its parents, children and children's parents.

10. The resulting distribution obtained disregarding the factor not of interest from the joint

The only drawback of the MH step is that the algorithm will require an additional hyperparameter linked to the definition of the proposal Q (it can be the variance for a gaussian or the step-size for a uniform). For this issue, several algorithms can automatically tune the hyperparameters for the sampling procedure, adapting the proposal during the iterations to increase and accelerate performances. Those are called *Adaptive Metropolis methods*.

3.1.3 Hamilton-Monte Carlo (HMC)

This section describes a sampling method that takes advantage of an additional piece of information about a distribution: its shape. It is named Hamiltonian Monte Carlo, and uses an approximate Hamiltonian dynamics simulation based on numerical integration, which is then corrected by performing a Metropolis acceptance step.

The algorithm was originally proposed by Duane [1987] for calculations in lattice quantum chromodynamics.

It is noticeable that HMC can only be applied to continuous problems since it requires the differentiability of the target density, while the previous methods can be essentially used for any problems[11].

Physical intuition:

When doing Metropolis-Hastings sampling with a naive proposal distribution, we are essentially performing a random walk without considering any additional information we may have about the distribution we want to sample.

There is a possible improvement: if the density function we want to sample from is differentiable, we can access its local shape through its derivative. The derivative tells us, at each point x, how the value of the density $P(x)$ increases or decreases depending on how

11. Eventually ensuring the discretization of a proposal step in MH, for discrete targets

we change the x. That means we should use the derivative of $P(x)$ to propose states with high probabilities, precisely the critical idea of Hamiltonian Monte Carlo (HMC).

The intuition behind HMC is that we can interpret a random walker as a particle moving under the effect of forces attracting it to higher-probability zones.

The potential energy $V(x)$ looks like a mountainous landscape that attracts the particle to its bottom: the steeper the landscape, the stronger will be the force pulling it towards the bottom. The bottom of this landscape coincides with the region's most considerable probability of $P(x)$. If we can predict where the particle will move to given its position and velocity, we can use the result of that prediction as a proposal state for Metropolis-Hastings, such that the sampler will be more focused on the peak of the probability distribution.

This prediction problem is classical in physics as part of well-known theories and methods from *Hamiltonian* mechanics (Carstens [2020]).

Auxiliary momentum variable:

As usual, the goal of sampling is to draw from a density $P(x)$ for variable x. This is typically a Bayesian posterior $P(x|y)$ given data y. But HMC introduces auxiliary momentum variables ρ and draws from a joint density

$$P(\rho, x) = P(\rho|x)P(x).$$

In most applications of HMC, the auxiliary density is a multivariate normal that does not depend on the parameters x,

$$\rho \sim \mathsf{MultiNormal}(0, M).$$

M is the Euclidean metric (Stan-Manual). It can be seen as a transform of parameter space that makes sampling more efficient, see Betancourt [2017] for details.

The Hamiltonian:

The joint density $P(\rho, x)$ defines a Hamiltonian

$$\begin{aligned} H(\rho, x) &= -\log P(\rho, x) \\ &= -\log P(\rho|x) - \log P(x). \\ &= T(\rho|x) + V(x), \end{aligned}$$

where the term

$$T(\rho|x) = -\log P(\rho|x)$$

is the "kinetic energy" and the term

$$V(x) = -\log P(x)$$

is the *potential energy.* It is specified by the program through its definition of a log density.

Generating transitions:

Starting from the current value of the parameters x, a transition to a new state is generated in two stages before being subjected to a Metropolis accept step. First, a value for the momentum is drawn independently of the current parameter values (practically, performing an i.i.d. sampling from the multinormal).

Thus momentum does not persist across iterations. Next, the joint system (x, ρ) made up of the current parameter values x and new momentum ρ is evolved via Hamilton's equations,

$$\begin{aligned} \frac{dx}{dt} &= +\frac{\partial H}{\partial \rho} = +\frac{\partial T}{\partial \rho} \\ \frac{d\rho}{dt} &= -\frac{\partial H}{\partial x} = -\frac{\partial T}{\partial x} - \frac{\partial V}{\partial x}. \end{aligned}$$

With the momentum density being independent of the target density, i.e., $p(\rho|x) = p(\rho)$, the first term in the momentum time derivative, $\partial T/\partial x$ is zero, yielding the pair time derivatives

$$\begin{aligned} \frac{dx}{dt} &= +\frac{\partial T}{\partial \rho} \\ \frac{d\rho}{dt} &= -\frac{\partial V}{\partial x}. \end{aligned}$$

Leapfrog integrator:

The last part leaves a two-state differential equation to solve. Most HMC implementations use the leapfrog integrator, which is a numerical integration algorithm that's specifically adapted to provide stable results for Hamiltonian systems of equations. The leapfrog algorithm takes discrete steps of some small-time interval ϵ. It begins by drawing a new momentum term independently of the parameter values x or previous momentum value.

It then alternates half-step updates of the momentum and full-step updates of the position.

$$\begin{aligned} \rho &\leftarrow \rho - \frac{\epsilon}{2}\frac{\partial V}{\partial x} \\ x &\leftarrow x + \epsilon\, M^{-1}\,\rho \\ \rho &\leftarrow \rho - \frac{\epsilon}{2}\frac{\partial V}{\partial x}. \end{aligned}$$

By applying L leapfrog steps, a total of $L\,\epsilon$ time is simulated. The resulting state at the end of the simulation will be denoted (ρ^*, x^*). The leapfrog integrator's error is on the order of ϵ^3 per step and ϵ^2 globally, where ϵ is the time interval, also known as the *step size* (Stan-Manual).

MH step:

If the leapfrog integrator were perfect numerically, there would no need to do any more randomization per transition than generating a random momentum vector. Instead, what is done in practice to account for numerical errors during integration is to apply a MH acceptance step, where the probability of keeping the proposal (ρ^*, x^*) generated by transitioning from (ρ, x) is

$$\min(1,\ \exp(H(\rho, x) - H(\rho^*, x^*)))\,.$$

If the proposal is not accepted, the previous parameter value is returned for the next draw and used to initialize the next iteration, as usual.

Algorithm summary:

The Hamiltonian Monte Carlo algorithm starts at a specified initial set of variable x; this value is either user-specified or generated randomly. Then, for a given number of iterations, a new momentum vector is sampled, and the current value of the parameter x is updated using the leapfrog integrator with discretization time ϵ and number of steps L according to the Hamiltonian dynamics. Then a Metropolis acceptance step is applied, and a decision is made whether to update to the new state (x^*, ρ^*) or keep the existing state.

Tuning the parameter L is very critical. A well-known adaptive extension of the algorithm that controls L automatically is the NUTS (No U-Turn Sampler, Matthew D. Hoffman [2011]).

3.1.4 Time complexity of MCMC

A key parameter for MCMC is the number of burn-in steps B. Intuitively, this corresponds to the number of steps needed to converge to our limit (stationary) distribution. It is called the *mixing time* of the Markov chain. Unfortunately, this time may vary dramatically and may sometimes take very long times.

This problem does not occur only with particular complicated distributions. Sampling is a complex problem in general, and MCMC does not give us a free lunch. Nonetheless, for many real-world distributions, sampling will produce handy solutions.

Another, perhaps more significant problem is that we may not know when to end the burn-in period, even if it is theoretically not too long. Many heuristics exist to determine whether a Markov chain has mixed; however, typically, these heuristics involve plotting specific quantities and estimating them by eye; even the quantitative measures are not significantly more reliable than this approach (V. Kuleshov [2019]).

In summary, even though MCMC can sample from the proper distribution - that in turn we can use for solving any inference problem - doing so may sometimes require long times, and there is no easy way to judge the amount of computation that we need to spend to find a good solution.

3.2 Variational methods

As discussed, sampling-based methods have several significant shortcomings. Besides issues related to time complexity, in order to reach a good solution, MCMC methods require also to choose appropriate sampling techniques, as a good proposal in MH. That can be an art in itself: as mentioned, a vast literature for adaptive methods exists.

In this chapter, we will look at an alternative approach to approximate inference called the *variational* family of algorithms.

3.2.1 Preliminaries: Inference as optimization

The main idea of variational methods is to cast inference as an optimization problem.

Suppose we have an intractable probability distribution p. Variational techniques try to solve an optimization problem over a class of tractable distributions $\mathcal{Q}$ in order to find a $q \in \mathcal{Q}$ that is most similar to p. Then, query q (rather than p) in order to get an approximate solution.

The main differences between sampling and variational techniques are that:

- Variational approaches will rarely find the globally optimum.

- Nevertheless, we will always know if they have converged. In some cases, we will even have bounds on their accuracy.

- In practice, variational inference methods often scale better and are more amenable to techniques like stochastic gradient descent, parallelization over multiple processors, and acceleration using GPUs.

Although sampling methods were historically invented first (in the 1940s), variational techniques have been steadily gaining popularity and are currently more widely used (Attias [2000], S. Ermon [2019]).

The Kullback-Leibler divergence:

To formulate inference as an optimization problem, we need to choose an approximating family $\mathcal{Q}$ and an optimization objective $J(q)$. This objective needs to capture the similarity between q and p; the field of information theory provides us with a tool for this called the *Kullback-Leibler (KL) divergence*, from Kullback [1951].

Formally, the KL divergence between two distributions q and p with discrete support is defined as

$$KL(q\|p) = \sum_x q(x) \log \frac{q(x)}{p(x)}.$$

In information theory, this function is used to measure differences in information contained within two distributions. The KL divergence has the following properties that make it especially useful in our setting:

- $KL(q\|p) \geq 0$ for all q, p.

- $KL(q\|p) = 0$ if and only if $q = p$.

Note however that $KL(q\|p) \neq KL(p\|q)$, i.e., the KL divergence is not symmetric. For that reason, it is a divergence, but not a distance.

The variational lower bound:

To perform variational inference with a KL divergence, let's fix a form for p. We will assume that p is a general (discrete, for simplicity) undirected model of the form

$$p(x_1, \dots, x_n; \theta) = \frac{\tilde{p}(x_1, \dots, x_n; \theta)}{Z(\theta)} = \frac{1}{Z(\theta)} \prod_k \phi_k(x_k; \theta),$$

where the ϕ_k are the factors and $Z(\theta)$ is the normalization constant. This formulation captures virtually all the distributions in which we might want to perform approximate inference, such as marginal distributions of directed models $p(x \mid e) = p(x, e)/p(e)$ with evidence e.

Given this formulation, optimizing $KL(q\|p)$ instantly is not possible because of the potentially intractable normalization constant $Z(\theta)$. Even evaluating $KL(q\|p)$ is not possible, because we need to evaluate p.

Instead, we will work with the following objective, which has the same form as the KL divergence, but only involves the unnormalized probability $\tilde{p}(x) = \prod_k \phi_k(x_k; \theta)$:

$$J(q) = \sum_x q(x) \log \frac{q(x)}{\tilde{p}(x)}.$$

This function is not only tractable, but it also has the following important property:

$$\begin{aligned} J(q) &= \sum_x q(x) \log \frac{q(x)}{\tilde{p}(x)} \\ &= \sum_x q(x) \log \frac{q(x)}{p(x)} - \log Z(\theta) \\ &= KL(q\|p) - \log Z(\theta) \end{aligned}$$

Since $KL(q\|p) \geq 0$, we get by rearranging terms that

$$\log Z(\theta) = KL(q\|p) - J(q) \geq -J(q).$$

Thus, $-J(q)$ is a *lower bound* on the log partition function $\log Z(\theta)$. In many cases, $Z(\theta)$ has an interesting interpretation. For example, we may be trying to compute the marginal probability $p(x \mid D) = p(x, D)/p(D)$ of variables x given observed data D that plays the role of evidence. We assume that $p(x, D)$ is directed. In this case, minimizing $J(q)$ amounts to maximizing a lower bound on the log-likelihood $\log p(D)$ of the observed data.

Because of this property, $-J(q)$ is the variational lower bound or the evidence lower bound (ELBO); it is in the form

$$\log Z(\theta) \geq \mathbb{E}_{q(x)}[\log \tilde{p}(x) - \log q(x)].$$

Crucially, the difference between $\log Z(\theta)$ and $-J(q)$ is precisely $KL(q\|p)$. Thus, by maximizing the evidence-lower bound, we are minimizing $KL(q\|p)$ by *squeezing* it between $-J(q)$ and $\log Z(\theta)$.

3.2.2 Loopy Belief propagation

In the Exact Inference Chapter, we have seen the junction tree algorithm that has a running time that is potentially exponential in the size of the largest cluster, since we need to marginalize all the cluster's variables.

For example an Ising model with $N \times N$ grid and $N \sim O(1000)$ will yield a clique with 2^{100} entries. The conversion of the non-tree model to a junction tree would lead to a graph with extremely large tree-width, which is unaffordable (Donghan Yu [2019]).

So it will not be easy for many graphs to find a good junction tree; applying the algorithm will not be possible. Nevertheless, we may not need the exact solution that the junction tree algorithm provides; we may be satisfied with a quick approximate solution instead.

Frey and MacKay [1997] introduced a technique for performing inference on complex (non-tree structure) graphs: Loopy belief propagation (LBP). Unlike the junction tree algorithm, which attempted to find the exact solution efficiently, LBP is an approximate inference algorithm.

Properties:

This heuristic approach often works surprisingly well in practice. In general, however, it may not converge, and its analysis is still an area of active research. For example, it provably converges on trees and graphs with at most one cycle. In most cases, directly copying the idea of belief propagation from tree to non-tree graphical model leads to two outcomes (Weiss [2000]):

- LPB converges, and its beliefs, although they may not necessarily equal the true marginals, are regularly close.
- It fails to converge at all and oscillates between multiple answers of the beliefs b.

It can be possible to show it as a particular case of variational inference algorithms (Jonathan S. Yedidia and Weiss [2004]).

The algorithm:

So, the main idea of loopy belief propagation is to extend message-passing from tree to non-tree graphical models. Suppose that we are given an MRF with pairwise potentials. The main idea of LBP is to disregard loops in the graph and perform message passing in any case. Given an ordering on the edges, at each time t we iterate over a pair of adjacent variables x_i, x_j, in that order, and perform the update:

$$m_{i\to j}^{t+1}(x_j) = \sum_{x_i} \phi(x_i)\phi(x_i, x_j) \prod_{\ell \in N(i)\backslash j} m_{\ell\to i}^{t}(x_i).$$

We keep performing these updates for a fixed number of steps or until convergence, when the messages do not change. Messages initialization is typically uniform.

Then the marginal probability will be based on the following equation:

$$b(x_i) \propto \phi(x_i) \prod_{\ell \in N(i)} m_{\ell\to i}(x_i).$$

To be specific, the messages are updated and passed iteratively among nodes at the same time. Unlike the belief propagation algorithm, where we pass the messages from the leaves to the root, the message passing here is recurrent. Another difference is that the loopy belief propagation algorithm does not need to pass messages only after collecting all the messages from its neighbors.

A theory behind - Bethe approximation to Gibbs free energy:

Suppose we want to find the approximation Q for p. Based on the factorized probability of the joint distribution p, we can write the exclusive KL-divergence as follows:

$$\begin{aligned} KL(q\|p) &= \sum_x q(x) \log \frac{q(x)}{p(x)} \\ &= \sum_x q(x) \log q(x) - \sum_x q(x) \log p(x) \\ &= -H_q(x) - \mathbb{E}_{q(x)}[\log p(x)] \\ &= -H_q(x) - \sum_k \mathbb{E}_{q(x)}[\log \phi_k(x_k;\theta)] + \log Z(\theta) \end{aligned}$$

We call the first two terms $-H_q(x) - \sum_k \mathbb{E}_{q(x)}[\log \phi_k(x_k;\theta)]$ the (Gibbs) *Free Energy*. Now we consider an example of tree-structured distribution. Based on the chain rule and local Markov property of undirected graphical model, we can expand the joint probability to factorized probability. According to the Bayes rule, we can summarize the joint probability for any tree-structured distribution (for any factor tree in general) as $b(x) = \prod_k b_k(x_k) \prod_i b_i(x_i)^{(1-d_i)}$, where d_i where the first product term is over the doubleton factors, the second product term is over the singleton factors and represents the degree of singleton node x_i. The entropy H of this distribution is given by:

$$H_{tree} = \sum_k \sum_{x_k} b_k(x_k) \ln b_k(x_k) + \sum_i (d_i - 1) \sum_{d_i} b_i(x_i) \ln b_i(x_i),$$

Thus, the Gibbs *free Energy* for a tree-structured distribution Q can be written as:

$$\begin{aligned} F_{tree} &= -H_q(x) - \sum_k \mathbb{E}_{q(x)}[\log \phi_k(x_k; \theta)] \\ &= \sum_k \sum_{x_k} b_k(x_k) \ln \frac{b_k(x_k)}{\phi_k(x_k)} - \sum_i (d_i - 1) \sum_{d_i} b_i(x_i) \ln b_i(x_i) \end{aligned}$$

Consider a more general Markov network (Fig. (2b)). The factor graph here is not a tree, and the distribution cannot be "exactly" factorized as:

$$p(x) = \frac{\prod_k b_k(x_k)}{\prod_i b_i^{d_i - 1}(x_i)}$$

However, we can still "approximate" it to be the same. This approximation is known as the Bethe approximation and the corresponding approximated energy (known as the Bethe free energy) is given by $\hat{F}(p, q) = F_{Bethe}$, which has the formulation:

$$F_{Bethe} = \sum_k \sum_{x_k} b_k(x_k) \ln \frac{b_k(x_k)}{\phi_k(x_k)} - \sum_i (1 - d_i) \sum_{x_i} b_i(x_i) \ln b_i(x_i)$$

Note that this is equal to the exact Gibbs free energy when the factor graph is a tree, but for general graphs, H_{Bethe} is not the same as the H of a tree.

So actually, we can choose $-J(q) = -F_{Bethe}$ as the variational lower bound.

Constrained minimization of the Bethe free energy:

Then we want to solve the constrained minimization problem:

$$\min \quad F_{Bethe}(b(x_i), b(x_k))$$

$$\text{subject to} \quad \sum_{x_i} b_i(x_i) = 1, \sum_{x_k|x_i} b_k(x_k) = b_i(x_i)$$

We can write the Lagrange form as:

$$L = F_{Bethe} + \sum_i \gamma_i \left(\sum_{x_i} b_i(x_i) - 1 \right) + \sum_k \sum_{i \in N(k)} \sum_{x_i} \lambda_{ki}(x_i) \left(\sum_{x_k|x_i} b_k(x_k) - b_i(x_i) \right)$$

Then we can put the zero-gradient solutions, and the interesting finding is that, if we identify $\lambda_{ki}(x_i) = \log(m_{i \to k}(x_i)) = \log \prod_{b \in N(i) \neq k} m_{b \to i}(x_i)$, then we get exactly the BP equations:

$$b_i(x_i) \propto \phi_i(x_i) \prod_{k \in N(i)} m_{k \to i}(x_i)$$

$$b_k(x_k) \propto \phi_k(x_k) \prod_{i \in N(k)} \prod_{c \in N(i)|k} m_{c \to i}(x_i)$$

Therefore, the singleton and doubleton potentials[12] in the Bethe optimization problem can be interpreted as beliefs at each iteration of LBP. Morever, the analysis shows that belief propagation on factor graphs is equivalent to minimizing the Bethe energy function.

Formally, the LBP algorithm would converge to true values in case of tree-structured graphs and may only approximate the true values in case of general graph, as Jonathan S. Yedidia [2000] showed that the belief update rules correspond to the fixed points of the

12. The initial potential of the node itself, and the edge potential with its neighbour (Rajarshi Das [2014]).

Bethe free energy. Precisely, it has been demonstrated that stable fixed points of BP are local minima of the Bethe free energy (Heskes [2003]).

3.2.3 Other developments

In a variational inference setting, we choose an approximating family $\mathcal{Q}$. A widely used classes of distributions is simply the set of fully-factored $q(x) = q_1(x_1)q_2(x_2)\cdots q_n(x_n)$; here each $q_i(x_i)$ is categorical distribution over a one-dimensional discrete variable, which can be described as a one-dimensional table.

It is perhaps a popular choice when optimizing the ELBO:

$$\min_{q_1,\cdots,q_n} J(q).$$

Variational inference with this choice of $\mathcal{Q}$ is called VMP (Variational Message Passing, Winn and Bishop [2005]) or *mean-field* inference, since it is a message-passing version of the mean-field method (Peterson and Anderson [1987]).

Interestingly, in the recent years Bayesian message passing schemes as variational message passing and belief propagation – each of which is derived from a free energy functional that relies upon different approximations (mean-field and Bethe respectively) become very popular to describe neuronal processing[13][14] (Thomas Parr [2019]).

13. Both BP and VMP have had some success in reproducing aspects of cognitive function, as architecture of communication between populations of neurons.

14. A generic account of brain function that subsumes the Bayesian brain is active inference (Friston [2015]). This account derives from the imperative for living creatures to maximise Bayesian model evidence or, more simply, engage in self-evidencing (Hohwy [2016]). This is equivalent to minimising their variational free energy (Friston [2010]).

CHAPTER 4

INFLUENCE DIAGRAMS: EVALUATION METHODS

Influence diagrams (IDs) or Decision Graphs (Jensen [2001]) can be used by decision analysts to help structure decision. Formally, IDs are DAGs $G = \{N, A\}$, where N is a finite set of distinct nodes partitioned into three disjoint subsets: decision, chance and value nodes, and E is a set of arcs classified according to the nodes they get into.

Decision nodes ($D_1, \ldots, D_k$, represented as squares) are variables under the control of the decision maker, chance nodes ($C_1, \ldots, C_m$, represented as circles) are probabilistic events variables, and value nodes ($V_1, \ldots, V_m$, represented as diamonds) are profits and costs of actions.

Arcs into decision nodes (informational arcs) specify the information available at the time of the decision, arcs into chance nodes (conditional arcs) represent probabilistic dependence and the dependences of chance nodes C_i upon its parent (direct predecessors, represented by $Par(C_i)$ is characterized by CPD $P(C_i|Par(C_i))$, and arcs into value nodes indicate functional dependencies and the dependences of value node V_i upon their parents (represented by $Par(V_i)$) is characterized by utility table $u(Par(V_i))$ (Ruan [2013]).

4.1 Expected utility hypothesis

Let $D_1, \ldots, D_k$ be all the decision nodes in an ID, $dom(D_i)$ is the set of alternatives at D_i, $\vec{E}$ be the set of chance and decision nodes whose information available before taking action at D_i, V_i be a value node pointed from D_i, $CPar(V_i)$ be chance nodes that have arcs pointing to V_i, and $cpar(V)$ be an instance of $CPar(V_i)$, i.e. a configuration of states of nodes in $CPar(V_i)$, then a decision rule for decision node D_i is a mapping $\delta_i : \vec{E} \to D_i = d$, where $d \in dom(D_i)$ and a strategy is a list of decision rules $\Delta = (\delta_1, \cdots, \delta_k)$ consisting of one rule for each decision node. The expected utilities corresponding to a decision rule and a strategy are defined by:

$$EU(\delta_i) = EU(\delta_i \ : \ \vec{E} \to D_i = d) = \sum_{cpar(V)} P(cpar(V_i)/\vec{E}) \times u(cpar(V_i), D_i = d)$$

$$EU(\Delta) = \sum_{i=1}^{k} EU(\delta_i)$$

The maximum expected utility is obtained over all possible strategies, and solving IDs means selecting the strategy that maximizes its expected utility, that is,

$$MEU = \max_{\Delta} EU(\Delta),$$

$$\Delta^* = \arg\max_{\Delta} EU(\Delta)$$

Solving ID is one of the most critical problems in IDs. In reality, it is impossible to evaluate each possible strategy and compare their expected utilities because the number of strategies grows exponentially in the number of decisions to be taken. The complexity is much greater than BN inference since we need to solve a number[15] of BN inference problems (Ruan [2013]).

4.2 VE for IDs

Given a decision node, is possible to compute the expected utility on an action applying Variable Elimination (Koller and Friedman [2009]), since $EU(\delta_i)$ is actually a sum over factors - utility nodes can be treated just as factors whose values are not probabilities.

For a simple network with just a decision node associated, it is easy to compute the optimal policy Δ^* just extending the VE algorithm to the value node and finding in that

15. One BN problem for each setting of decision node parents and decision node value.

way the maximal expected utility.

To solve general and more complex IDs, much more effort could be required, and effective techniques are commonly on the use of dynamic programming principle to avoid enumerating all policies.

4.3 The Arc Inversion method

Shachter proposed a method based on arc reversal and node deletion in the paper Evaluating Influence Diagrams (1986). It belongs to a wider class of evaluation algorithms that preserves utilities during modifications of the graph. In fact, this approach removes nodes from an ID through some value-preserving reduction until only the value node remains.

At that point, it has determined all of the optimal strategies and computed the maximal expected utility. When a node is removed, it can be dropped from the current node-set, and all arcs incident to it can be drowned from the arc set (Ruan [2013]). Again, this method is essentially based on dynamic programming (for the nodes removal mechanism) and on the Bayes' rule (for the inversion of arcs).

- *Barren node removal*: A chance or decision node is called a barren node if it has no successors. A barren node may be simply removed from an ID. If it is a decision node, then an alternative would be optimal.

- *Chance node removal*: A chance node C that directly precedes the value node and nothing else in an ID may be removed by conditional expectation, and then the value node inherits all of the conditional predecessors from node C. The new utility of the value node can be computed by the formula:

$$u_n(V) \leftarrow \sum_{c \in dom(C)} EU(V/Par_o(V)) \times P(c/Par(C)) \tag{4}$$

Where $Par_o(V)$ is the set of parents of the value node before removing node C.

- *Decision node removal*: A decision node D that is a conditional predecessor of the value node and all other conditional predecessors of the value node are its informational predecessors may be removed by maximizing expected utility, conditioned on the values of its informational predecessors, i.e.

$$u_n(V) \leftarrow \max_{d \in dom(D)} EU(V/Par_o(V)) \tag{5}$$

  The maximizing alternatives should be recorded as the optimal strategy. The value node inherits no new conditional predecessors from this operation.

- *Arc reversal*: An arc (C_1, C_2) can be replaced by arc (C_2, C_1) if there is no other directed path from C_1 to C_2. Afterward, both nodes inherit each other's conditional predecessors, i.e. $Par_n(C_2) \leftarrow Par_o(C_1) \cup Par_o(C_2) \backslash \{C_1\}, P_n(c_2/Par_n(C_2))$ and $P_n(c_1/Par_n(C_1))$ can be computed by formula:

$$P_n(c_2/Par_n(C_2)) \leftarrow \sum_{c_1 \in dom(C_1)} P_o(c_2/Par_o(C_2)) \times P_o(c_1/Par_o(C_1))$$

$$P_n(c_1/Par_n(C_1)) \leftarrow P(c_1/c_2, Par_n(C_2)) = P(c_2/Par(C_2)) \times \frac{P(c_1/Par_o(C_1))}{P(c_2/Par_n(C_2))}$$

4.3.1 The algorithm

First of all, is important to check if the ID is regular (checking the associated DAG) and that exists at least a path containing all the decision nodes (*no forgetting* property).

Algorithm 1: Arc Inversion pseudo-code

```
Check for regular, oriented, "no forgetting" arcs ID;
Remove all barren nodes;
while $Par(V) \neq 0$ do
    if $\exists i \in CPar(V) : Par(C_i) = V$:
        remove chance node $i$;
    else if $\exists D_i \in Par(V) : Par(V)/D_i \in CPar(V)$:
        remove decision node $D_i$;
        remove all barren nodes;
    else:
        find $i \in CPar(V) : Par(C_i) \cup D = 0$ then:
            while $Par(C_i) \neq 0$ do
                find $j \in Par(C_i)$: no other directed $(i, j)$ paths;
                reverse arc $(i, j)$;
            end
            remove chance node $i$;
        end
    end if
end
```

4.4 Other approaches

An alternative method to solve Influence Diagrams could be that of a combinatorial optimization approach. Since we have to avoid the exploration of all the possible feasible policies, we can set up an heuristic that looks for an optimal solution with certain search methods: the key idea is to treat the expected utility as an objective function - so that we can use variable elimination for computing the *fitness* of a solution during iterations of the algorithm. This would be an approximate method for IDs.

4.4.1 Genetic Algorithms (GA)

Genetic algorithms (GA) are a class of iterative optimization methods that use the principles of evolutionary biology, proposed by Holland [1992]. Terminology is usually associated with optimization changes to the biological terms. Solutions in a problem are referred to as individuals, and the set of individuals is known as a population.

GA mimics the Darwinian theory of survival of fittest in nature: the fitness of an individual refers to the values of the objective functions. Each iteration in the GA is called a generation. For some generation n, the nth population is known as the offspring population while the $n-1$th population is the parent population.

GA begins with some initial population and each generation afterwards produces an offspring population using genetic operators. Genetic operators include selection, crossover, and mutation. They use the principle of natural selection to find the best solution, using the principle of diversity to avoid convergence to a local minima.

Selection:

Selection operators are used to select which offspring survive to the next generation. Tournament selection is a selection operator that randomly sorts the individuals into blocks and chooses the best individual from each block.

Crossover:

Crossover operators are used to mix two or more parents to produce similar, but slightly different offspring. Most crossover operators convert the individual into binary representation to perform the operations. One-point crossover crosses the binary digits at some crossover point of two parents to create two new individuals.

For example, consider the parents 7 and 18 which in binary are 00111 and 10010 repsectively. Let the crossover point be position 3; thus the offspring would be 00010 and 10111 or 2 and 23 respectively. Another type of commonly used crossover operation is two-point crossover which is similar to one-point crossover but with two crossover points.

Mutation:

Mutation operators are used to further preserve the diversity of a population to ensure non-convergence to an optimum. A type of mutation known as flip bit also performs operations on the binary representation of a number where each bit in the binary representation has some probability of being mutated.

The algorithm:

Algorithm 2: Pseudo-code for a simple genetic algorithm

```
$pop \leftarrow GenerateInitialPopulation(desiredPopulationSize)$;
$pop \leftarrow assignFitnessToEachMember(pop)$;
while $run - time < allowedTime$ do
    $newGeneration \leftarrow selectMembersOfPopForCrossover(pop)$;
    $newGeneration \leftarrow mutatePopulation(newGeneration)$;
    $pop \leftarrow assignFitnessToEachMember(pop)$;
    if $highestFitnessInPopulation(pop) >= fitnessRequired$ then
        return $fittestMemeberOfPopulation(pop)$
    end
end
return $fittestMemeberOfPopulation(pop)$
```

For IDs, this algorithm can be applied to maximize the expected utility, computed for each policy with VE. Individuals are taken as combinations of subsets of each decision rule: the composition of all the minimal-length arrays[16] useful to perform decisions is a possible solution for the algorithm. In the fitness computation phase, an unique transformation decompose individuals into decision factors to evaluate their associated utility.

16. The reduction of strategies is though to compress information: when we have a binary decision, for example, the single rule representation contains redundant values.

Criticalities:

This approach is very useful to solve complex IDs with several decision nodes. The implementation is for simplicity developed only for binary rule-nodes; an example is in the Python library chapter.

Although this optimization approach is generally very efficient, it is necessary to observe that the latter is not preferred in the community dealing with probabilistic graphical models.

The power of PGMs lies in the graphical representation of knowledge models; to reap benefits such as intuitiveness, analysis and updating. Algorithms such as message passing, or arc inversion, exploit and align with this philosophy. To withdraw from the graphing model to translate the problem into a combinatorial optimization problem could mean losing these advantages.

CHAPTER 5
A PYTHON PACKAGE FOR PGMS

5.1 Implementation details

The repository is available at `https://github.com/pierociffo/mypgm`. It contains the source code of the package and the examples. The work is named *mypgm* and it is organized in 5 main modules:

1. *base*, with fundamental classes:

   - Factor
   - CPD

   Tables can be visualized with *Pandas* (McKinney [2010]).

2. *pgms*, with graphical models classes:

   - BayesianNetwork
   - InfluenceDiagram

   Models regularity is checked with *topological sorting*; graphical manipulations are based on *networkx* (Aric A. Hagberg and Swart [2008]) and visualization tools are taken from *graphviz* (Gansner and North [2000]).

3. *exacted*, for exacted inference algorithms:

   - VariableElimination
   - ExpectedUtility

- ArcInversion

4. *approximated*, for approximated inference:

   - ProbabilisticSampling
   - LoopyBeliefPropagation
   - GeneticAlgorithm

5. *mining*, containing useful functions for conversion of the results in optimal decision tables, for analysis concerning Chapter 6.

Some computational issues were solved taking inspiration from works available in the repositories of Paul Rauber, in particular for the *Expected Utility* and the *Variable Elimination* classes, and Maxwell Forbes, for the *LPB* algorithm.

As introduced in the first Chapter, I faced the problem of hybrid BNs with the discretization approach. For this task, I use the class (quantile-based) *Discretized* from *pgmpy* (Ankan and Panda [2015]).

5.2 Examples

Defining Networks:

Let's see an example of Bayesian Net definition. This example is taken from *pgmpy*. Basically the network is created just passing to it the proper set of factors. The CPD objects are just particular instances of the $Factor$ class, properly normalized.

```python
# Clone the repository from github and import the project
import git
git.Git("./").clone("https://github.com/pierociffo/mypgm")
import mypgm
```

```python
# Import class and functions useful for the example
from mypgm.base import RandomVar, CPD
from mypgm.pgms import BayesianNetwork
P = RandomVar('Pollution', 2)
S = RandomVar('Smoker', 2)
C = RandomVar('Cancer', 2)
X = RandomVar('Xray', 2)
D = RandomVar('Dyspnoea', 2)
# CPDs
cpd_poll = CPD([P], values=[[0.9], [0.1]])
cpd_smoke = CPD([S], values=[[0.3], [0.7]])
cpd_cancer = CPD([C, S, P], values=[0.03, 0.05, 0.001, 0.02, 0.97, 0.95,
    0.999, 0.98])
cpd_xray = CPD([X, C], values=[0.9, 0.2, 0.1, 0.8])
cpd_dysp = CPD([D, C], values=[0.65, 0.3, 0.35, 0.7])
# define the net
bn2 = BayesianNetwork([cpd_poll, cpd_smoke, cpd_cancer, cpd_xray, cpd_dysp
    ])
# Visualize with graphviz
bn2.viz()
```

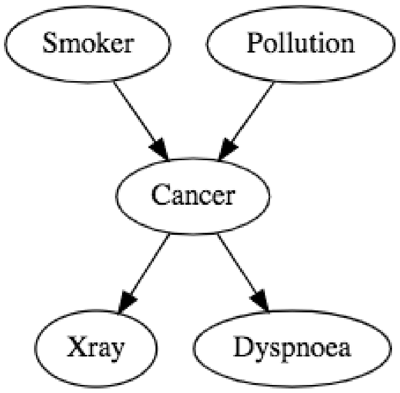


Figure 5.1: Example network.

Thanks to the *MarkovRF* class, is possible to easily convert Bayesian Nets to Markov

Random Fields.

5.2.1 BNs: Fraud detection example

This is an example of a simple fraud detection system for credit cards, based on the following information: when the card holder is traveling abroad, fraudulent transaction are more likely since tourists are prime targets for thieves.

We want to compute the probability of fraud if an abroad purchase is done, and the probability of fraud if an abroad purchase is done and the holder is travelling.

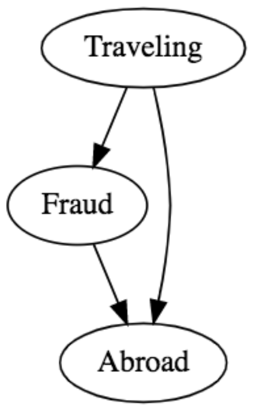
```python
from mypgm.exacted import VariableElimination
# Define the model structure: first, define the variables.
# Random variable for the indicator of traveling, with 2 possible values:
    0 if not traveling, 1 if traveling
T = RandomVar(name='Traveling', k=2)
# Indicator for the fraudolent transaction
F = RandomVar('Fraud', 2)
# Abroad purchase
A = RandomVar('Abroad', 2)
```

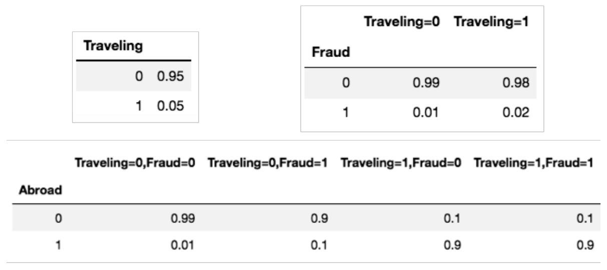
Precisely, 2% of transactions are fraudulent when the card holder is traveling, whereas only 1% of the transactions are fraudulent when he is not traveling. On average, 5% of all transactions happen while card holder is traveling. If a transaction is fraudulent, then the likelihood of a purchase abroad increases, unless the card holder happens to be traveling.

Specifically, when the card holder is not traveling, 10% of the fraudulent transactions are abroad purchases, whereas only 1% of the legitimate transactions are abroad purchases. On the other hand, when the card holder is traveling, 90% of the transactions are abroad purchases regardless of the legitimacy of the transactions.

```python
# Define the conditional distributions:
f_T = CPD(scope=[T], values=[0.95, 0.05])
```

```
# The first two probability values referred to cases when there is no
    fraud
f_F = CPD([F, T], [0.99, 0.98, 0.01, 0.02])
# Abroad purchase transaction probabilities conditioned on T and F
f_A = CPD([A, T, F], [0.99, 0.9, 0.1, 0.1, 0.01, 0.1, 0.9, 0.9])
# Visualizing table of the CPD can be useful
f_A.to_dataframe()
# The order of the variables in a CPD term is determining the conditional
    dependence relation:
# The first variable specified is conditional dependent on the others
# So, once defined the right CPDs, the structure of the Bayesian Network
    is set on
fraud_model = BayesianNetwork([f_T, f_F, f_A])
# Initializing the exact inference method
ve = VariableElimination(fraud_model)
# Computing the marginal of Fraud given Abroad=1.
print(ve.marginal(hypothesis=[F], evidence=[(A, 1)]))
[Output ]:
Fraud
(0,) -> 0.9665
(1,) -> 0.0334
# Computing the marginal of Fraud given Abroad=1 and Traveling=1.
print(ve.marginal([F], [(A, 1), (T, 1)]))
[Output ]:
Fraud
(0,) -> 0.9800
(1,) -> 0.0200
```





Traveling	
0	0.95
1	0.05

Fraud	Traveling=0	Traveling=1
0	0.99	0.98
1	0.01	0.02

Abroad	Traveling=0,Fraud=0	Traveling=0,Fraud=1	Traveling=1,Fraud=0	Traveling=1,Fraud=1
0	0.99	0.9	0.1	0.1
1	0.01	0.1	0.9	0.9

Figure 5.2: Fraud detection Network and CPT dataframes.

5.2.2 *BNs: the student's grade example*

Here is analyzed a Bayes network model of a student's grade g on an exam; in addition to g, are also modeled other aspects of the problem, such as the exam's difficulty d, the student's intelligence i, his SAT score s, and the quality l of a reference letter from the professor who taught the course. Each variable is binary, except for g, which takes 3 possible values.

```
D = RandomVar('Difficulty', 2)
I = RandomVar('Intelligence', 2)
G = RandomVar('Grade', 3)
L = RandomVar('Letter', 2)
S = RandomVar('SAT', 2)
# init the factors with propabilities relationship
cpd_d = CPD([D], values=[0.6, 0.4])
cpd_i = CPD([I], values=[0.7, 0.3])
cpd_g = CPD([G, I, D], values=[0.3, 0.05, 0.9,  0.5,
                               0.4, 0.25, 0.08, 0.3,
                               0.3, 0.7,  0.02, 0.2])
cpd_l = CPD([L, G], values=[0.1, 0.4, 0.99,
                            0.9, 0.6, 0.01])
cpd_s = CPD([S, I], values=[0.95, 0.2,
                            0.05, 0.8])
```

```
# pass CPDs to a DAG structure
bn3 = BayesianNetwork([cpd_d, cpd_i, cpd_g, cpd_l, cpd_s])
bn3.viz()
```

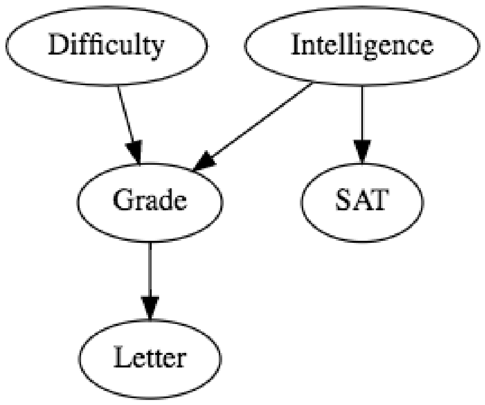


Figure 5.3: The student's grade Network.

Variable Elimination:

Let's start to make inference on the network with variable elimination. After passing the graph structure to the inference method, we can query (or looking for maximum a posterior assignment) whether variable present, also including evidence.

```
#initialize the method
ve = VariableElimination(bn3)
#marginal query on G
print(ve.marginal([G]))
[Output]:
Grade
(0,) -> 0.3620
(1,) -> 0.2884
(2,) -> 0.3496
#marginal query on G given an evidence on D and I
print(ve.marginal([G], [(D, 0), (I, 1)]))
```

```
[Output]:
Grade
(0,) -> 0.9
(1,) -> 0.08
(2,) -> 0.02
#map query on L, G, S
print(ve.map_query([(D, 0), (I, 1)]))
[Output]:
[(Letter, 1), (Grade, 0), (SAT, 1)]
#map query on G
print(ve.map_query([(D, 0), (I, 1), (L, 1), (S, 1)]))
[Output]:
[(Grade, 0)]
```

Probabilistic Sampling:

Now try to apply an approximate inference engine like probabilistic sampling to the same network. We can compare the results with the exact one just founded. The class *GibbsSampler* provide the implementation of a Metropolis-Hasting within Gibbs algorithm, if specified, otherwise we can perform a pure Gibbs sampling from the probability distribution of interest.

For the MH step, is possible to set the type of proposal distribution ($uniform$ by default) with the relative hyperparameter.

In this example I try to use a gaussian proposal (*delta* in that case is the standard deviation):

```
# set the MCMC parameters and query G with a simulation
gs = GibbsSampler(bn3, metropolis=True, proposal=gaussian_proposal, delta
   =0.8)
gs.sample(burn_in=1000, n=2500, plot=[G], print_posterior=[G])
[Output]:
```

```
100%|||||||||| 2500/2500 [00:03<00:00, 691.90it/s]
(0,): 0.3612
(1,): 0.2664
(2,): 0.3724
# incorporate evidence on D, I and query G
gs.reset()
gs.sample(burn_in=1000, n=1500, evidence=[(D, 0), (I, 1)], print_posterior
    =[G])
[Output]:
100%|||||||||| 1500/1500 [00:03<00:00, 691.90it/s]
(0,): 0.8900
(1,): 0.0740
(2,): 0.0360
```

As shown, results are not perfectly equals to those of variable elimination, but they are quite good. It is noticeable that results are sensible to changes in hyperparameters, and that different runs produces different results, as the method is based on stochastic simulations. One can also choose what posteriors to print as outcome and whether to plot the MCMC and the resulting histogram (when dealing with continue variables, also kernel density estimation is shown) for the variable queried.

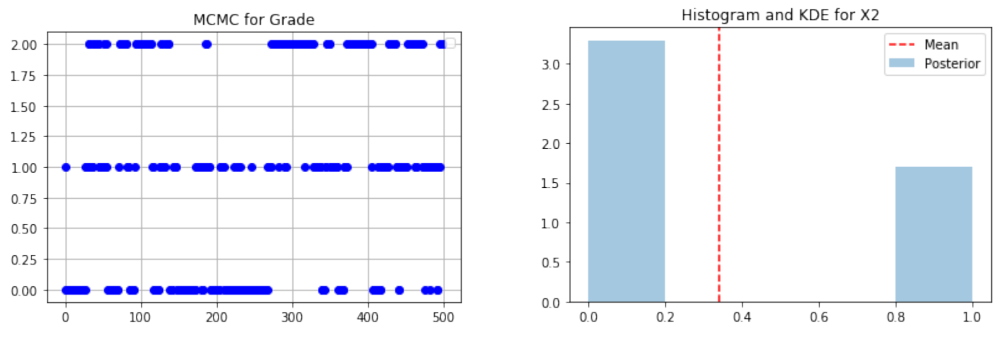


Figure 5.4: MH within Gibbs sampling results after querying *Grade*.

Loopy Belief Propagation:

Finally, we can perform message passing on the graph. As it does not contain loops, the algorithm will return exact solutions. We have to transform the network to an undirected model, and then a single execution of the *LBP* function will give as output all the marginals.

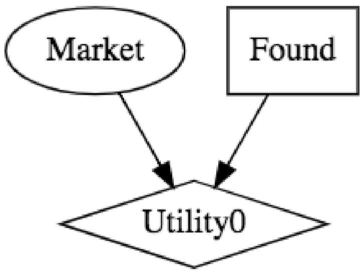
```
mrf3=bn3.to_markov_model()
iters, converged = mrf3.lbp(normalize=True)
print('LBP ran for %d iterations. Converged = %r' % (iters, converged))
# Print out the final marginals
mrf3.print_rv_marginals()
[Output]:
'LBP ran for 3 iterations. Converged = True
Marginals for RVs (normalized)':
Difficulty          Intelligence         Grade
   0    0.7            0    0.6            0    0.3620
   1    0.3            1    0.4            1    0.2884
                                           2    0.3496

Letter                                 SAT
   0    0.4976                      0    0.7250
   1    0.5023                      1    0.2750
```

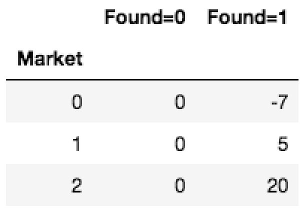
5.2.3 IDs: the market example

Regarding Influence Diagrams, the definition of decision nodes and utility functions is always related to factors. One has just to specify it with the argument "*mod*" (*chance* by default). In this setting, the decision maker have to choose between two opposite action (to fund/not to fund), when the its utility is determined by the outcome of a probabilistic event in the *market*. The following toy model is taken from Koller and Friedman [2009].

```
# random variable for the market: 3 possible situations
```

```
M = RandomVar('Market', 3)
# 2 possible action to take: to found or not to found the market
F = RandomVar('Found', 2)
# utility table of the actions given the values of the market
uMF = Factor([M, F], [0, -7, 0, 5, 0, 20],  mod='utility')
uMF.to_dataframe()
# probability table for the market
cM = CPD([M], [0.5, 0.3, 0.2])
# F defines the decision scheme
dF = Factor([F], mod='decision')
id_ = InfluenceDiagram([cM], [uMF], [dF])
id_.viz()
```





	Found=0	Found=1
Market		
0	0	-7
1	0	5
2	0	20

Figure 5.5: An influence diagram example and its utility function table.

Variable Elimination:

Consider the VE extension evaluation method for influence diagrams. In order to determine the optimal rule solution of the problem we have to pass the decision variable to determine to the function $optimal - decision - rule$, from the class $ExpectedUtility$. In this case, the aim is to find a strategy for F, that is associated to a binary action chance.

```
# Alternative decision rules for F
dF_1 = CPD([F], [1.0, 0])
dF_2 = CPD([F], [0, 1.0])  # Optimal
```

```
# initialize the decision graph to the method
eu = ExpectedUtility(id_)
print('Maximal expected utily for action:', eu.optimal_decision_rule([F]))
[Output]:
Found
(0,) -> 0.0
(1,) -> 1.0
print('Expected utily of the first action:', eu.expected_utility([dF_1]))
[Output]: 0.0
print('Expected utily of the second action:', eu.expected_utility([dF_2]))
[Output]: 2.0
```

The optimal solution consists in **to fund** ($F = 1$). For clarity, the value of the expected utility associated to the alternative actions are reported as well.

Arc Inversion:

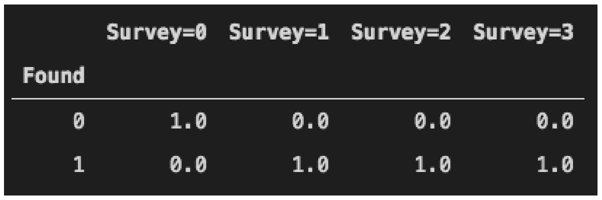
Here we use a slightly more complex case of the one introduce for the influence diagram representation example. In this extension, the decision maker can also take advantage from the results of a *survey*, possibly useful to decide whether to fund or not, thanks to the additional decision node *test*, describing the action of testing/not testing the *survey*. Also this example is from Koller and Friedman [2009].

```
M = RandomVar('Market', 3)
S = RandomVar('Survey', 4)  # S = 3 means no survey
T = RandomVar('Test', 2)
F = RandomVar('Found', 2)
# utility
u = Factor([M, F, T], [0, -1, -7, -8, 0, -1, 5, 4, 0, -1, 20, 19], mod='
    utility')
cM = CPD([M], [0.5, 0.3, 0.2])
# influence table for S
```

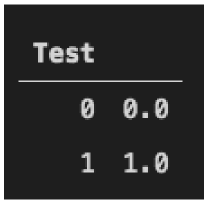
```
cST = CPD([S, M, T], [0.0, 0.6, 0.0, 0.3, 0.0, 0.1,
                        0.0, 0.3, 0.0, 0.4, 0.0, 0.4,
                        0.0, 0.1, 0.0, 0.3, 0.0, 0.5,
                        1.0, 0.0, 1.0, 0.0, 1.0, 0.0])
# decision node for F given S
dFS = Factor([F, S], mod='decision')
# decision factor for T
dT = Factor([T], mod='decision')
# set the diagram
id1 = InfluenceDiagram([cM, cST], [u], [dT, dFS])
id1.viz()
```

To evaluate the influence diagram is enough to pass the decision structure to an *ArcInversion* object and then *solve* it. The dedicated function will return 3 objects: the new solved network structure, the optimal policy found composed as a list of optimal action, the list of utilities gained from each action taken.

```
ai = ArcInversion(id1)
ai.solve()
```



```
      Survey=0  Survey=1  Survey=2  Survey=3
Found
0          1.0       0.0       0.0       0.0
1          0.0       1.0       1.0       1.0
```



```
Test
0  0.0
1  1.0
```

Figure 5.6: Output best decision of Arc Inversion on the market example.

In this case, the algorithm suggests the decision maker **to test** and **to fund** according to a specific policy. This is not a trivial solution, since the action of testing seen alone seems to reduce the utility. Otherwise, this is confirmed to be optimal also with other approaches.

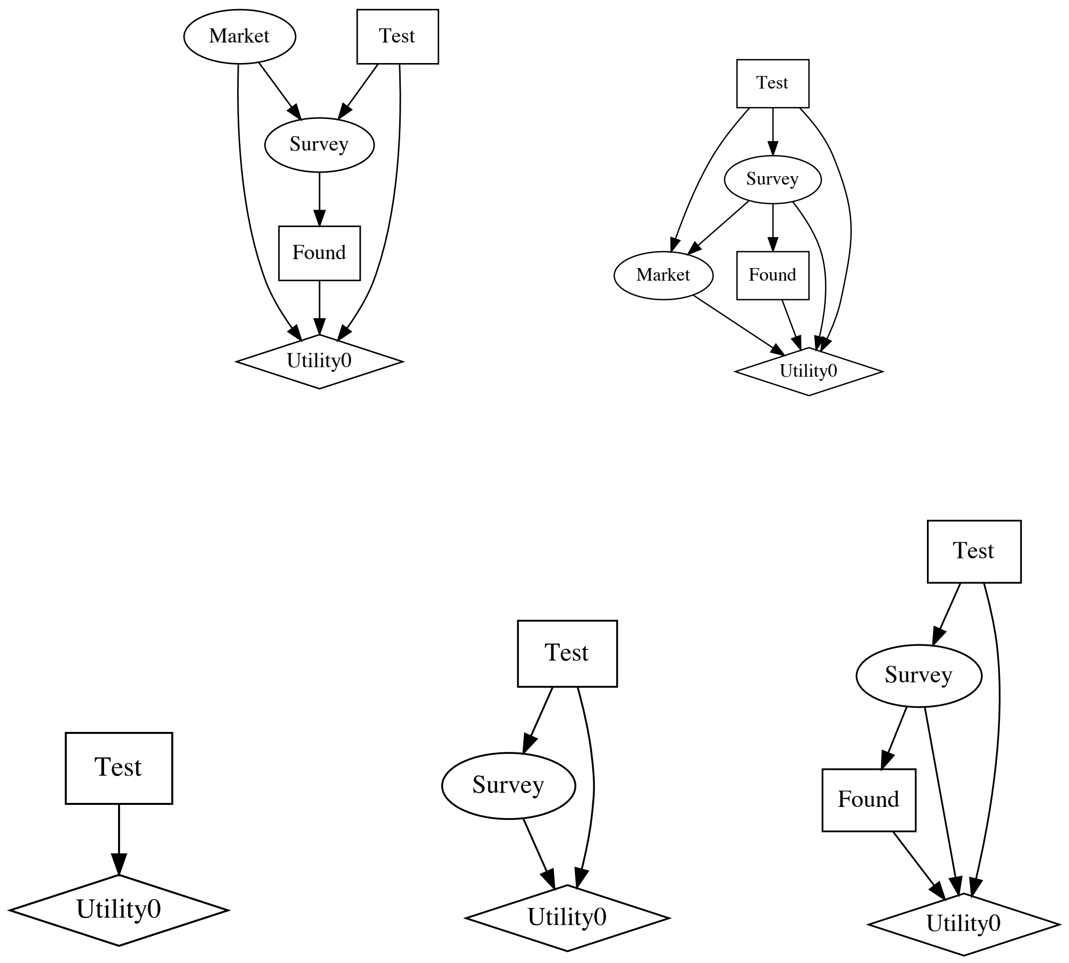


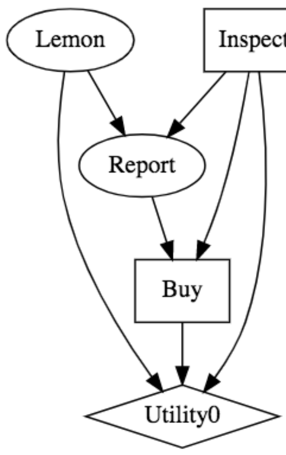
Figure 5.7: (Clockwise) Progressive reduction of the market influence diagram along the evaluation steps of the Arc Inversion.

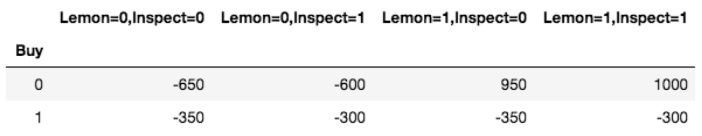
5.2.4 IDs: the car buyer example:

In this example from S. Matsumoto [2011], the decision maker wants to buy a used car, but there's a good chance it is a "lemon" (i.e., prone to breakdown). Before deciding to buy it, he can take it to a mechanic for inspection. S/he will give to him a report on the car, labelling it either "good" or "bad". A good report is positively correlated with the car being sound, while a bad report is positively correlated with the car being a lemon. The report costs 50 however. So you could risk it, and buy the car without the report. Owning a sound car is better than having no car, which is better than owning a lemon.

```
L = RandomVar('Lemon', 2) # y/n
R = RandomVar('Report', 3)  # good, bad, none
I = RandomVar('Inspect', 2) # get inspection or not
B = RandomVar('Buy', 2) # to buy/ no to buy a car

cL = CPD([L], [0.5, 0.5])
dI = Factor([I], mod='decision')
dB_RI = Factor([B, R, I], mod='decision')

cR_IL = CPD([R, I, L], [0.2, 0.9, 0, 0, 0.8, 0.1, 0, 0, 0, 0, 1, 1])
# utility
uBL = Factor([B, L, I], [-650, -600, 950, 1000, -350, -300, -350, -300])

car = InfluenceDiagram([cL, cR_IL], [uBL], [dI, dB_RI])
```

This time there is just an utility function, nevertheless it can be separated (the *Inspection* only lead to an additional price of 50 if *True*).





Buy	Lemon=0,Inspect=0	Lemon=0,Inspect=1	Lemon=1,Inspect=0	Lemon=1,Inspect=1
0	-650	-600	950	1000
1	-350	-300	-350	-300

Figure 5.8: The car influence diagram example and its utility function table.

Genetic algorithm:

To apply this combinatorial optimization method (belonging to the *Evolutionary algotithms*) is sufficient to initialize and set the hyperparameters.

```
h = heuristicID(car, [dI, dB_RI])
h.run_algorithm(max_epochs = 20, mut=0.9, population_size = 2, crossp=0.8)
[Output]:
'Max EU at iteration 0': 150.0
Starting simulation...
'Max EU at the end': 205.0
'End simulation': time spend = 0.1635451316833496
'Best solution': [1, 1, 0, 0, 1, 0, 0]
'Optimal strategy':
```

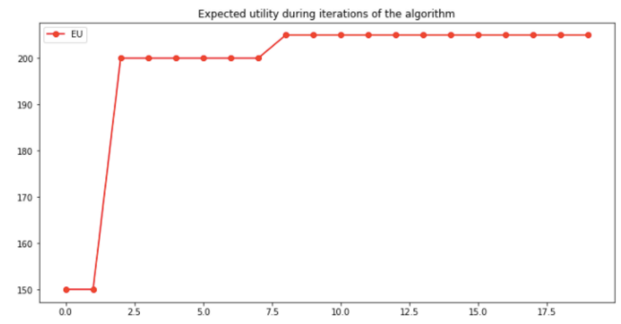
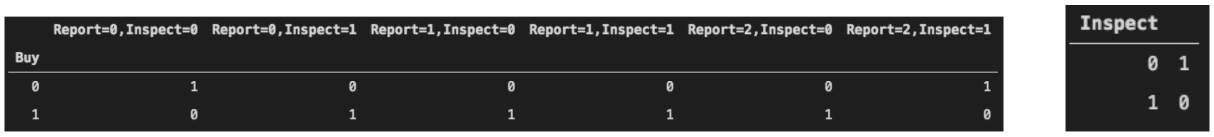


Figure 5.9: Expected utility computed during the GA steps.



```
     Report=0,Inspect=0  Report=0,Inspect=1  Report=1,Inspect=0  Report=1,Inspect=1  Report=2,Inspect=0  Report=2,Inspect=1
Buy
  0                   1                   0                   0                   0                   0                   1
  1                   0                   1                   1                   1                   1                   0
```

```
Inspect
     0  1
     1  0
```

Figure 5.10: Output best decision of Genetic Algorithm on the car example.

As one can see, the algorithm converges fast to the optimum, 20 epochs are more than sufficient. Surely this is because this example does not represent a complex problem from the combinatorial point of view.

CHAPTER 6

DATA MINING EVALUATION

In this chapter, I will perform data mining processing over the results of some of the graphical methods explored in the previous sections.

Especially when we are dealing with complex network structures, the output of an algorithm executed over the model is challenging to interpret. An optimal decision table may have millions of rows and typically more than twenty columns leading to enormous data sets for storage and analysis.

Solutions of an ID aim to provide the best decision-making recommendations. However, experts may find such recommendations hard to accept if they come without explaining the reasons the proposed decisions are optimal. Unexplained responses are not good enough for expert users since the model approximates the real world. They need to understand the underlying reasons or implicit rules (Bielza and Fernandez del Pozo [2011]).

Indeed, every decision algorithm is designed with a proper rationale, and results should be self-explanatory depending on the way it was constructed. Nevertheless, developing additional tools to search for features implicit in the result is an possibly valuable experiment to test the robustness of the proposed solution and provide ulterior sources of explanation. For this purpose, data mining seems suitable.

There are several data mining methods in the literature whose model results (and implicit rules) are simple to understand and to be visualized. So, this final part of the work aims to rely on some of these data mining techniques to explain the results of the decision algorithms previously applied on graphical models.

Possible policies for IDs can be viewed as tables of actions. Each action has several attributes so that it can be viewed as a vector. The attributes concerns both the realization of random variables involved and decisions linked to different nodes.

The label for each action can be represented by a binary state asserting the optimality, a feature established by the previous algorithm.

In this way, we can set up a complete dataset[17] to use as a training set for a learning algorithm: the resulting model could be helpful in our explanatory purposes. Each element of the dataset will have as its characteristics the realization of the events considered and the choices.

An example of conversion is reported with *A medical Decision Problem*.

Having split the dataset opportunely into train and test[18] is possible to analyze performance measures associated with those new methods and compare them. The processes that will follow have been developed using existing libraries, especially *scikit-learn*, a famous Python library for machine learning (Pedregosa et al. [2011]), *scikit-learn-extra*, for clusters analysis, and *mlxtend*, for association rules.

In the following sections, I will introduce two clinical examples with different levels of complexity. After a brief exposition of the data mining methods developed, there will be a conclusive section with the complete results of the analysis with eventual comparison.

6.1 Examples

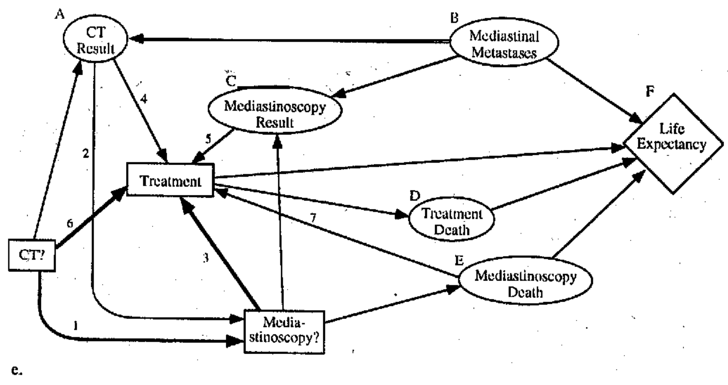
6.1.1 A medical decision problem

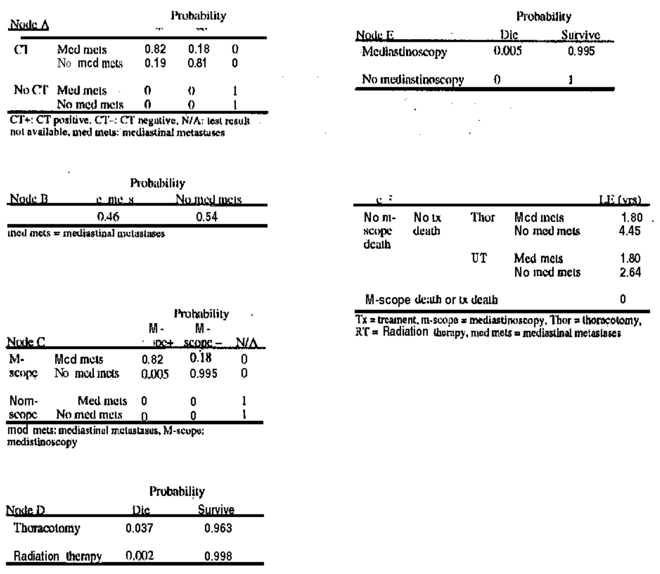
This example is taken from the paper "*Use of Influence Diagrams to Medical Decisions-Structure*, R. Nease, D. Owens, 2000".

The influence diagram model consists of 5 chance nodes, one value node, three decision nodes and 17 arcs. The utility fuction is associated to the life experience of the patient, that is strongly influenced by the decision of which treatment to take (thoracotomy or radiotherapy) and whether to make or not Computed tomography (CT) and mediastinoscopy.

17. Precisely, I used the *DataFrame* object from Pandas

18. *Scikit-klearn* can perform a split with *shuffle* and test proportion of 0.25 by default.





Node A		Probability		
		[illegible]	[illegible]	
CT	Med mets	0.82	0.18	0
	No med mets	0.19	0.81	0
No CT	Med mets	0	0	1
	No med mets	0	0	1

CT+: CT positive, CT–: CT negative, N/A: test result not available, med mets: mediastinal metastases

Node B	[illegible]	No med mets
	0.46	0.54

med mets = mediastinal metastases

Node C		M-[illegible]	M-scope –	N/A
M-scope	Med mets	0.82	0.18	0
	No med mets	0.005	0.995	0
Nom-scope	Med mets	0	0	1
	No med mets	0	0	1

mod mets: mediastinal metastases, M-scope: medistinoscopy

Node D	Die	Survive
Thoracotomy	0.037	0.963
Radiation therapy	0.002	0.998

Node E	Die	Survive
Mediastinoscopy	0.005	0.995
No medinstinoscopy	0	1

[illegible]				LE (yrs)
No m-scope death	No tx death	Thor	Med mets	1.80
			No med mets	4.45
		UT	Med mets	1.80
			No med mets	2.64
M-scope death or tx death				0

Tx = treatment, m-scope = mediastinoscopy, Thor = thoracotomy, RT = Radiation therapy, med mets = mediastinal metastases

Figure 6.1: A medical decision problem Influence Diagram with associated CPTs.

I will evaluate the solution computed by the Arc Inversion and the Genetic Algorithm. Once defined variables and CPDs as the figure is indicating, we do the following:

```python
dg = InfluenceDiagram([cA_FB, cC_BH, cD_G, cE_H, cB], [LifeExperience], [
    dF, dG, dH])
# This function convert the variables to a table of possible action.
t = action_table([A, B, C, D, E, F, G, H])
# run the decision algorithms
ai = ArcInversion(dg)
ai.solve()
# save results
optimal_trial = ai.decisions
```

```
# Use another example optimal strategy from heuristic solution
from mypgm.approximated import heuristicID
# it requires more epochs to find the exact solution since the ID is more
    complex
h = heuristicID(dg, [dF, dG, dH])
h.run_algorithm(max_epochs = 50, mut=0.9, population_size = 4, crossp=0.8)
optimal_trial_1 = h.solution
# This function is adding a label regarding the optimaliity of each action
     according to the supposed strategy.
train = add_label(t, optimal_trial)
# Now we have vectors for data mining evaluation
y = train.loc[:, ['Optimal']]
X = train.loc[:, train.columns != 'Optimal']
```

The optimal decision is overall a table of 8 columns and 144 rows. The Genetic Algorithm identifies 25 optimal actions (or rows) and 199 sub-optimal actions, while Arc Inversion labels 20 as optimal and 124 as sub-optimal.

6.1.2 *The Primary Gastric non-Hodgkin Lymphoma*

Now we consider a more complex example. The relative ID was introduced by P.J. Lucas in 1998 in the paper *"Computer-based Decision Support in the Management of Primary Gastric non-Hodgkin Lymphoma"*.

Primary gastric non-Hodgkin lymphoma, gastric NHL for short, is a relatively rare disorder, accounting for about 5% of gastric tumors. This disorder is caused by a chronic infection by the Helicobacter pylori bacterium. Treatment consists of a combination of antibiotics, chemotherapy, radiotherapy and surgery.

The influence diagram model consists of 17 chance nodes, one value node, three decision nodes and 42 arcs.

The first of the decision nodes, HELICOBACTER-TREATMENT, corresponds to the decision to prescribe antibiotics against H. pylori. The second decision concerns carrying out SURGERY. The possibilities are either curative surgery, involving the complete removal of the stomach and locoregional tumor mass; palliative surgery, i.e. partial removal of the stomach and tumor; or no surgery. The last decision, CT-RT-SCHEDULE, is concerned with the selection of chemotherapy, radiotherapy, chemotherapy followed by radiotherapy, or none.

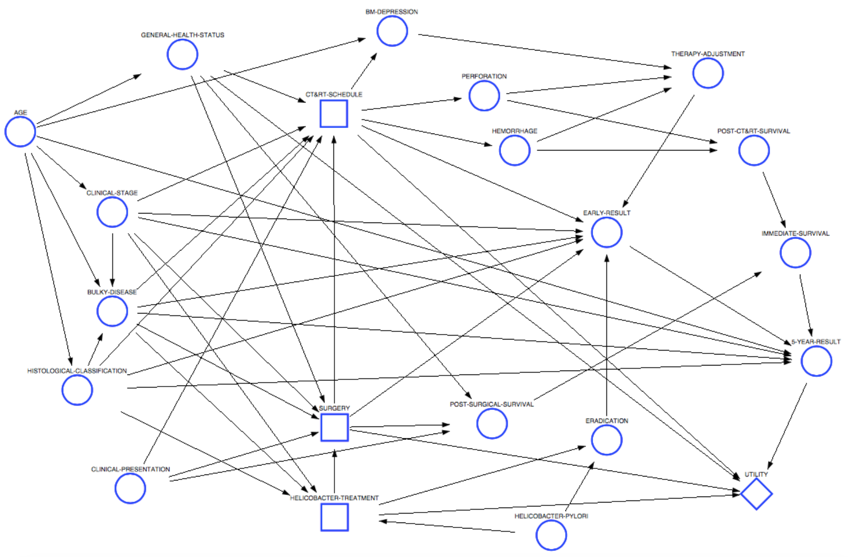


Figure 6.2: Influence diagram for the treatment of gastric NHL.

For this example the optimum is computed finding the maximum expected utility actions, given a particular shape of the utility function. The resulting decision table has 10 columns and 1029 rows, of which 120 are labeled as optimal and 900 as sub-optimal.

6.2 Performance measures

To analyze performances of data mining methods, I used ROC curves, confusion matrix and accuracy. The *Receiver Operating Characteristic* (ROC, Mandrekar [2010]) is a measure of a classifier's predictive quality that compares and visualizes the tradeoff between the model's *sensitivity* and *specificity*. When plotted, a ROC curve displays the true positive rate on the

Y axis and the false positive rate on the X axis on both a global average and per-class basis. The ideal point is therefore the top-left corner of the plot: false positives are zero and true positives are one.

This leads to another metric, *area under the curve* (AUC), which is a computation of the relationship between false positives and true positives. The higher the AUC, the better the model generally is. However, it is also important to inspect the steepness of the curve, as this describes the maximization of the true positive rate while minimizing the false positive rate.

By definition a *confusion matrix* C is such that $C_{i,j}$ is equal to the number of observations known to be in group i and predicted to be in group j.

Thus in binary classification, the count of true negatives is $C_{0,0}$, false negatives is $C_{1,0}$, true positives is $C_{1,1}$ and false positives is $C_{0,1}$.

Accuracy is the number of correct predictions divided by the total number of predictions.

6.3 Classfication trees

Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

The deeper the tree, the more complex the decision rules and the fitter the model.

Typical advantages of decision trees are:

- Simple to understand and to interpret. Trees can be visualised.
- Requires little data preparation compared to other techniques. The cost of using the tree for predicting data is logarithmic in the number of data points used to train it.
- Able to handle both numerical and categorical data.

- Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., an artificial neural network), results may be more difficult to interpret (Pedregosa et al. [2011]).

The main disadvantage of decision trees is that Decision-tree learners can create over-complex trees that do not generalise the data well, in other words they are easily affected by *overfitting*.

```python
from sklearn import tree
dt = tree.DecisionTreeClassifier()
dt.fit(X_train, y_train)
# predictions
y_train_hat = dt.predict(X_train)
y_test_hat = dt.predict(X_test)
# accuracy
print('Test accuracy: {}'.format(accuracy_score(y_train, y_train_hat)))
print('Test accuracy: {}'.format(accuracy_score(y_test, y_test_hat)))
# AUC score
y_test_score_tree = get_auc_scores(dt, X_train, X_test, y_train, y_test)
# confusion matrix
show_cm(y_test, y_test_hat)
```

In this way is possible to analyze the data mining model built over the two algorithmic results, thanks to performance measures and visualization tools (that for trees are simple).

As one can see from Fig. 6.3 and 6.4, the Arc Inversion solution is more simple and the data mining model performs better on it. Results summary is in Table 6.1.

6.4 Nearest Neighbors

The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these. The number of

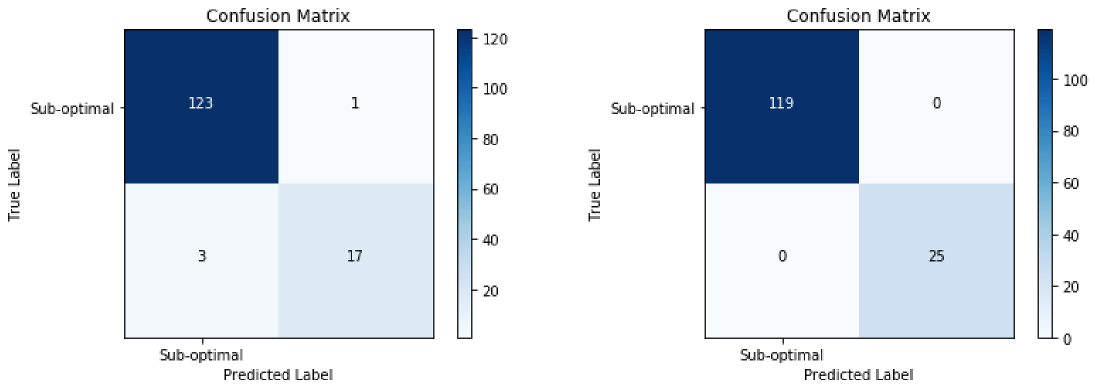


Figure 6.3: Confusion matrices of decision tree analysis applied to Genetic Algorithm (left) and Arc Inversion results (right) of the first clinical problem.

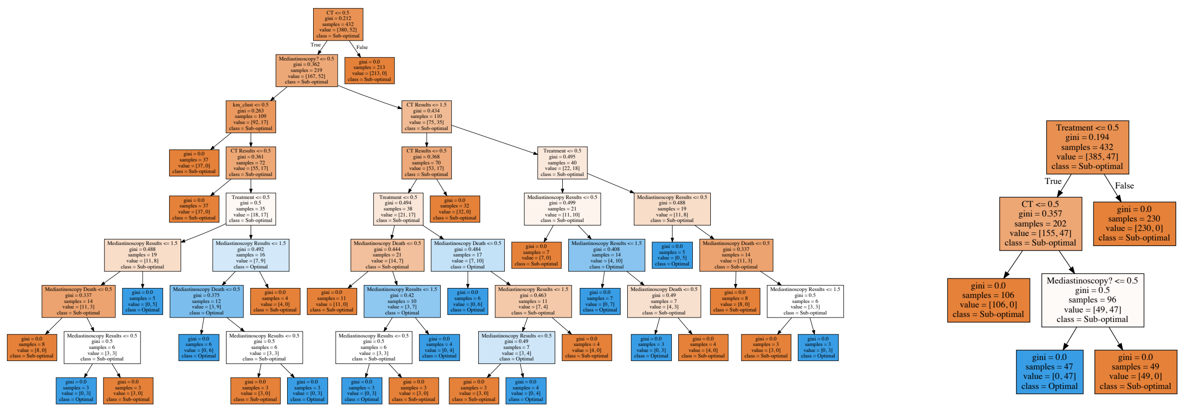


Figure 6.4: Decision trees applied to Genetic Algorithm (left) and Arc Inversion results (right) of the first clinical problem.

Decision table	Accuracy	Accuracy with Clusters	AUC Score	AUC with Clusters
Medical ID by GA	**0.9722**	0.9305	0.9209	0.9242
Medical ID by AI	1.0	1.0	1.0	1.0
Gastric NH by EU	**0.9951**	0.8182	0.9939	0.7699

Table 6.1: Summary of test performances of Decision Tree over results for the clinical examples.

samples can be a user-defined constant k. For my analysis, I set different values of k and the best turned out to be $k = 3$. The distance can be any metric measure: standard Euclidean distance is the most common choice. Neighbors-based methods are known as non-generalizing machine learning methods, since they simply “remember” all of its training data: it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point.

Despite its simplicity, nearest neighbors has been successful in a large number of classification and regression problems. Being a non-parametric method, it is often successful in classification situations where the decision boundary is very irregular. In our case we could not expect it to work very well, because optimal actions should not necessarily be close in the features space. However, as an experiment it exploits quite good results, shown here:

Decision table	Accuracy	Accuracy with Clusters	AUC Score	AUC with Clusters
Medical ID by GA	**0.8750**	**0.8750**	0.8213	0.8247
Medical ID by AI	0.9375	0.9236	**0.9727**	0.9648
Gastric NH by EU	**0.9931**	0.7191	**0.9993**	0.7523

Table 6.2: Summary of test performances of KNN over results for the clinical examples.

6.5 Bayes Classifier

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the *naive* assumption of conditional independence between every pair of features given the value of the class variable. Bayes' theorem states the following relationship, given class variable y and dependent feature vector x_1 through x_n, :

$$P(y \mid x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n \mid y)}{P(x_1, \dots, x_n)}$$

Using the naive conditional independence assumption that

$$P(x_i|y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i|y),$$

for all i, this relationship is simplified to

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \dots, x_n)}$$

Since $P(x_1, \dots, x_n)$ is constant given the input, we can use the following classification rule:

$$P(y \mid x_1, \dots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y),$$

and we can use Maximum A Posteriori (MAP) estimation to estimate $P(y)$ and $P(x_i \mid y)$.

The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $P(x_i \mid y)$. Common choices are *Gaussian* or *Multinomial*.

In spite of their over-simplified assumptions - strange in particular for our case, where x_i variables conditional dependence is determined by a network - naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering (Pedregosa et al. [2011]). Results are in Table 6.3.

Decision table	Accuracy	Accuracy with Clusters	AUC Score	AUC with Clusters
Medical ID by GA	0.6134	**0.6597**	**0.8439**	0.8423
Medical ID by AI	1.0	1.0	1.0	1.0
Gastric NH by EU	**0.9358**	0.9037	**0.9695**	0.9579

Table 6.3: Summary of test performances of Naive Bayes over results for the clinical examples.

6.6 SVM

Support-vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin, the lower the generalization error of the classifier.

Often happens that the sets to discriminate are not linearly separable in the original space. So, the original finite-dimensional space can be mapped into a much higher-dimensional space, for making the separation easier in that space. Mappings used by SVM schemes are designed to ensure that dot products of pairs of input data vectors may be computed easily in terms of the variables in the original space, by defining them in terms of a Kernel function $k(x, y)$ selected to suit the problem (Pedregosa et al. [2011]). This is called the "*kernel trick*".

The advantages of support vector machines are:

- Effective in high dimensional spaces, also in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided are *Linear* and *RBF*.

The main disadvantages of SVM include the criticality in choosing Kernel functions and regularization term[19] to avoid over-fitting when the number of features is much greater than

19. The regularization parameter is a degree of importance given to misclassifications, in the relaxed quadratic optimization problem.

the number of samples. For my analysis, I set an RBF kernel and a regularization parameter $C = 1.0$. Results are in Table 6.4.

Decision table	Accuracy	Accuracy with Clusters	AUC Score	AUC with Clusters
Medical ID by GA	**0.8611**	**0.8611**	0.8971	**0.9443**
Medical ID by AI	1.0	1.0	1.0	1.0
Gastric NH by EU	**0.9737**	0.8202	**0.9941**	0.7431

Table 6.4: Summary of test performances of SVM over results for the clinical examples.

6.7 Discriminant analysis: LDA

Linear discriminant analysis (LDA) is a discriminant approach that attempts to model differences among samples assigned to certain groups, that can be used as a classifier. The aim of the method is to maximize the ratio of the between-group variance and the within-group variance. When the value of this ratio is at its maximum, then the samples within each group have the smallest possible scatter and the groups are separated from one another the most. Once the LDA assumption of equal group covariances for a two-class discriminant problem is fulfilled, one tries to maximize the expression:

$$S = \frac{pC_b p^T}{pC_w p^T}$$

where C_b and C_w are between- and within-group covariance matrices, respectively, and p is the direction in multivariate data space that separates the two groups of samples the best[20]. Results are in Table 6.5.

20. It is insteresting to emphasize that p is the eigenvector obtained from the PCA decomposition of matrix $C_w^{-1}C_b$.

Decision table	Accuracy	Accuracy with Clusters	AUC Score	AUC with Clusters
Medical ID by GA	**0.8611**	**0.8611**	0.8491	0.8463
Medical ID by AI	1.0	1.0	1.0	1.0
Gastric NH by EU	**0.9416**	0.8814	**0.9718**	0.9449

Table 6.5: Summary of test performances of LDA over results for the clinical examples.

6.8 Association rules

For association rules mining over the results I developed two different algorithms: Apriori and F-P Growth. Apriori algorithm is given by R. Agrawal and R. Srikant in 1994 for finding frequent itemsets. Name of the algorithm is Apriori because it uses prior knowledge of frequent itemset properties. We apply an iterative approach or level-wise search where k-frequent itemsets are used to find k+1 itemsets.

To improve the efficiency of level-wise generation of frequent itemsets, an important property is used called Apriori property which helps by reducing the search space: *All subsets of a frequent itemset must be frequent.* If an itemset is infrequent, all its supersets will be infrequent. To analyze frequencies, I had to convert the training set made of possible solution to a categorical database.

Once found the frequent itemsets with one of the methods, is possible to search the boolean association rules. Here I report the first 4 association rules made with Apriori for the results of Arc Inversion:

```
                                        antecedents    consequents  \
0                                         (Optimal)           (CT)
1                                     (CT result 0)  (Sub-optimal)
2                                     (CT result 1)  (Sub-optimal)
3                                     (CT result 2)  (Sub-optimal)
4                           (Mediastinal Metastases)  (Sub-optimal)
```

Slightly different rules are mined with FP-Growth:

```
                                        antecedents    consequents  \
0                                               (No CT)  (Sub-optimal)
1                           (No Mediastinoscopy, No CT)  (Sub-optimal)
2     (No Mediastinoscopy, Mediastinal Metastases Re...  (Sub-optimal)
3     (Mediastinal Metastases, No Mediastinoscopy, N...  (Sub-optimal)
4     (Mediastinal Metastases, No Mediastinoscopy, M...  (Sub-optimal)
```

F-P Growth (frequent-pattern growth) algorithm is another popular technique in Market Basket Analysis (first introduced by Han). It produces similar results as Apriori algorithm but is computationally faster due to a mathematically different technique. It follows a two-step data preprocessing approach: first, it counts the number of occurrences of each item in the transactional dataset; then, it creates a search-tree structure using the transactions.

Unlike Apriori, F-P Growth sorts items within each transaction by it's frequency from largest to smallest before inserting it into a tree. This is where it has a substantial computational advantage over Apriori since it does the frequency sorting early on. Items which don't meet minimum frequency support requirements are discarded from the tree.

I made the same mining operation for the result given by Genetic Algorithm, and I report the first 4 rules with the FP-Growht method:

```
0                                         (Thoracotomy)  (Sub-optimal)
1        (Thoracotomy, Mediastinal Metastases Result 2)  (Sub-optimal)
2                                  (No Mediastinoscopy)  (Sub-optimal)
3                     (Thoracotomy, No Mediastinoscopy)  (Sub-optimal)
4     (Thoracotomy, No Mediastinoscopy, Mediastinal ...  (Sub-optimal)
```

6.9 Cluster analysis

Clustering is a type of unsupervised machine learning which aims to find homogeneous subgroups such that objects in the same group (*clusters*) are more similar to each other than the others.

Although an unsupervised machine learning technique, the clusters can be used as features in a supervised machine learning model.

K-Means is a clustering algorithm which divides observations into k clusters. Since we can dictate the amount of clusters, it can be easily used in classification where we divide data into clusters which can be equal to or more than the number of classes.

The idea is to use the clusters as additional features, looking for improvements over another classification model.

```python
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
clu = KMeans(n_clusters = 2, random_state=0, tol=0.00001, max_iter=3000)
clu.fit(X_train)
# predict
y_labels_train = clu.labels_
y_labels_test = clu.predict(X_test)
# score of the clusters
print(f'Silhoutte score:', silhouette_score(X_train, y_labels_train))
# add clusters as features
X_train['km_clust'] = y_labels_train
X_test['km_clust'] = y_labels_test
```

To check and to plot centers of the clusters (Fig. 6.5) can be useful for various reasons. Fist, one can check if according to the algorithm - with respect to a specific feature - optimal solutions are well separated from sub-optimal ones or not, looking at the centroids nearness. Furthermore, is possible to eventually identify additionally intermediate clusters. For the axis selection, one can be suggested by previous mining methods as association rules.

Finally, clusters visualization can be particularly useful in the case of discrete variables problems, where plotting points in the features spaces is generally less understandable.

I tried other 3 clustering techniques: K-Medoids, OPTICS (Ordering Points To Identify the Clustering Structure) and Agglomerative Clustering. In the summary tables, I also

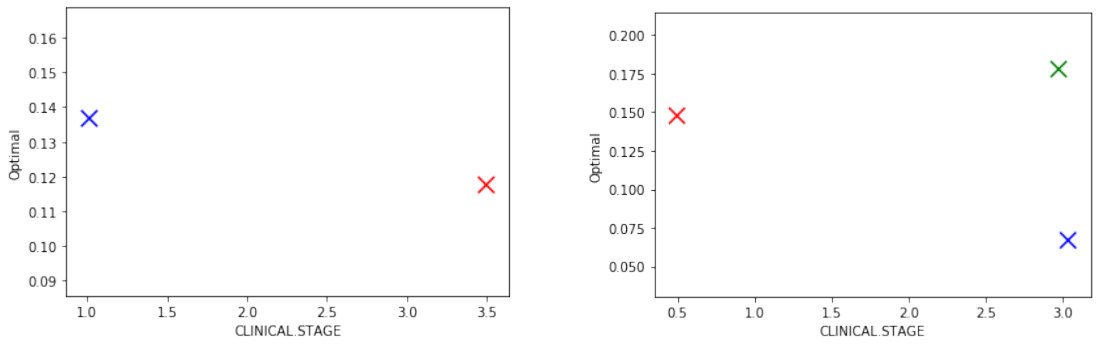


Figure 6.5: Plot of cluster centers found with K-Means with k=2 and k=3, along two features of the optimal Gastric NHL table.

reported hyperparamenters and random seeds to make replicable the results.

The K-Medoids problem is similar to K-Means. Both algorithms are partitional (breaking the dataset up into groups) and attempt to minimize the distance between points labeled to be in a cluster and a point designated as the center of that cluster, but K-Medoids chooses actual data points as centers (medoids or exemplars), and thereby allows for greater interpretability of the cluster centers than K-Means, where the center of a cluster is not necessarily one of the input data points (it is the mean between cluster's points). The classical greedy search to solve it is called Partitioning Around Medoids (PAM).

Hierarchical clustering is a general family of clustering algorithms that build nested clusters by merging or splitting them successively. This hierarchy of clusters is represented as a tree, an example is in Fig. 6.6. The root of the tree is the unique cluster that gathers all the samples, the leaves being the clusters with only one sample.

Agglomerative Clustering performs a hierarchical clustering using a bottom up approach: each observation starts in its own cluster, and clusters are successively merged together.

Finally, unlike previous method that based on distances, OPTICS uses the density of cases to assign cluster membership. Its basic idea is similar to another popular method called DBSCAN, but it addresses one of DBSCAN's major weaknesses: the problem of detecting meaningful clusters in data of varying density. To do so, the points of the database are

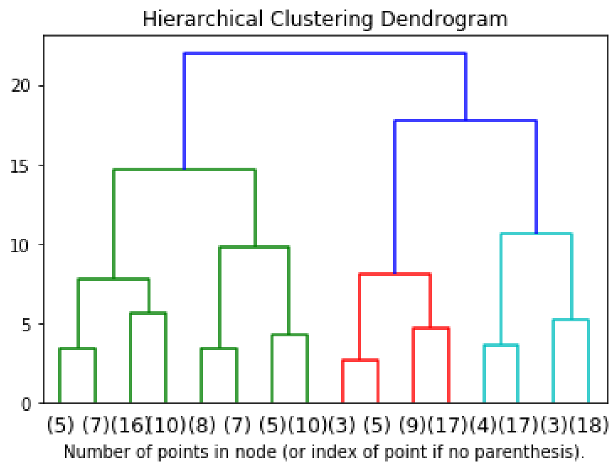


Figure 6.6: Dendogram obtained with Agglomerative Clustering performed on the heuristic solution of the first clinical ID.

ordered such that spatially closest points become neighbors in the ordering.

Evaluating the performance of a clustering algorithm is not as trivial as counting the number of errors or the precision and recall of a supervised classification algorithm. I used the *Silhouette* score, that is a measure of how similar an object is to its own cluster (*cohesion*) compared to other clusters (*separation*). The silhouette ranges from 1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. If most objects have a high value, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters.

Clustering method	N. Cluster	Silhouette Score	Hyperparameters	Random Seed
K-Means	2	**0.5601**	$tol = 0.001$	5
PAM	4	0.4308	$metric = manhattan$	0
OPTICS	18	0.4388	$min.\ samples = 3$	0
Agglomerative	5	0.2974	$metric = euclidean$	2

Table 6.6: Summary table of clustering analysis performances on the Genetic Algorithm results of the first clinical problem.

Clustering method	N. Cluster	Silhouette Score	Hyperparameters	Random Seed
K-Means	2	0.4701	$tol = 0.0001$	5
PAM	4	0.4905	$metric = euclidean$	0
OPTICS	18	**0.4328**	$min.\ samples = 3$	0
Agglomerative	5	0.2974	$metric = euclidean$	2

Table 6.7: Summary table of clustering analysis performances on the Arc Inversion results of the first clinical problem.

Clustering method	N. Cluster	Silhouette Score	Hyperparameters	Random Seed
K-Means	2	0.2508	$tol = 0.0001$	5
PAM	4	0.2489	$metric = manhattan$	0
OPTICS	5	0.0237	$min.\ samples = 5$	0
Agglomerative	5	**0.2722**	$metric = l1$	2

Table 6.8: Summary table of clustering analysis performances on the solution of the gastric NHL problem.

6.10 Results and comments

In this section, the final results are reported and discussed. A first observation is that the solution of the first clinical example founded with Arc Inversion is straightforward to mine concerning the one of Genetic Algorithm. As Tables 6.4 and 6.5 report, accuracy and AUC scores are very high for Arc Inversion results: except for KNN, they are maximal. Firstly, it is noticeable that the example is not highly complex, so it is easy for powerful data mining methods to perform well. In addition, the logic behind the Arc Inversion is different with respect to that of Genetic Algorithm, and this can explain the different result in the analysis: GA works in practice as a combinatorial method, so its results could be less interpretable. GA results are performed remarkably well by Decision Tree, followed by LDA and Naive Bayes in terms of AUC.

The additional clustering analysis in the first example does not improve the results, but they remain fairly good. This is clear looking at the ROC curves in Fig. 6.7. Reasons are mainly due to the difficulty in identify clusters for those problems: *silhouttes* scores are quite low; moreover, different clustering methods are the better performing on different instances, as reported in Tables 6.6, 6.7 and 6.8.

In particular, regarding the gastric NHL optimal treatment, the clustering analysis does not perform very well. The increasing complexity of the network makes it hard to divide possible actions into reasonable clusters: this negatively affects performance. Comparable results are in the summary tables and Fig. 6.8. Interestingly, the only method whose performances are not significantly reduced by the cluster is the Naive Bayes.

I also report the first 4 association rules mined with FP-Growth on the optimal treatment for the gastric NHL example, in Fig. 6.9.

The first rules could not be very useful to explain results, since they are based on frequent itemsets and the more frequent could be trivial ones. But analyzing the table deeper could be interesting.

The optimal gastric NHL treatment shows a deep associated decision tree as showed in Fig. 6.10, and this is coherent with the structure of the problem. However, confusion matrices of many methods give good measures, as reported in Fig. 6.11. Best performances are obtained with Decision tree, quickly followed by KNN and then by SVC.

6.11 Conclusions

It noticeable that, in general, more than one optimal table could exist for a given problem. This is the first clinical example, where two different algorithms produce two different optimal solutions that are both associated with the same maximal expected utility. However, each algorithm finds only one optimal solution; therefore, in the data mining process, all the others possibilities are labeled as sub-optimal, regardless of their actual optimality or sub-optimality. On the one hand, this could affect the reliability of the results classifier. On the other, this follows our purposes since we want to mine the reasons behind algorithm outcomes instead of finding all the optimal solutions.

In conclusion, overall results show that it is possible to explain outcomes of an unsupervised method with a supervised method. The idea to construct a new model over the result of a complete model could be successful in terms of analysis, finding criticalities, and understanding essential variables for the decisions suggested by the first approach.

Classification performances alone could not be instrumental. Though, when the mining procedure are not *black-boxes* we can access several visualization and explanation tool, which may not be available with the original model, or would be mainly different.

In our case, since the example problems treated involve only discrete variables, the extrapolated rules easier to illustrate concerned decision trees and association rules essentially. However, in general, when dealing with continuous variables, SVM or LDA's decision boundaries could be very useful too.

When it is needed to critically analyze the recommended result of a machine learning process, visualizing it from different points of view can be crucial, especially if accessing a more intuitive version of the explanation.

6.12 Future lines of research

Nowadays, to construct decision-support systems (DSSs) supervising experts through concise knowledge extraction and satisfactory answers when approaching complex queries is a challenging task.

Mine is a proposal of an analysis method to demonstrate how data mining is a powerful tool capable of lending itself to that particular issue.

However, this work did not deal with optimizing the performance of the decision model on the basis of such analysis. A natural following could be to perform sensitivity analysis over sets of parameters of the decision model. For instance, constructing a dataset by merging the data mining evaluation of models sampled over the parameter sets of the decision algorithm can show the relevance of the parameters conditioned to the optimal evaluated policies.

In this direction, an intriguing topic to investigate would concern the ability of DSSs to identify criticalities in their responses when the applied evaluation methods show poor or contrasting performances.

Besides, further research could concern the automatic selection of the best set of hyper-parameters of the mining technique, looking for optimization of the research techniques for explaining results, probably necessary when the decision tables are incredibly complex.

Finally, it could be interesting to search for suitable visualization methods of decision boundaries when dealing with discrete or mixed variables.

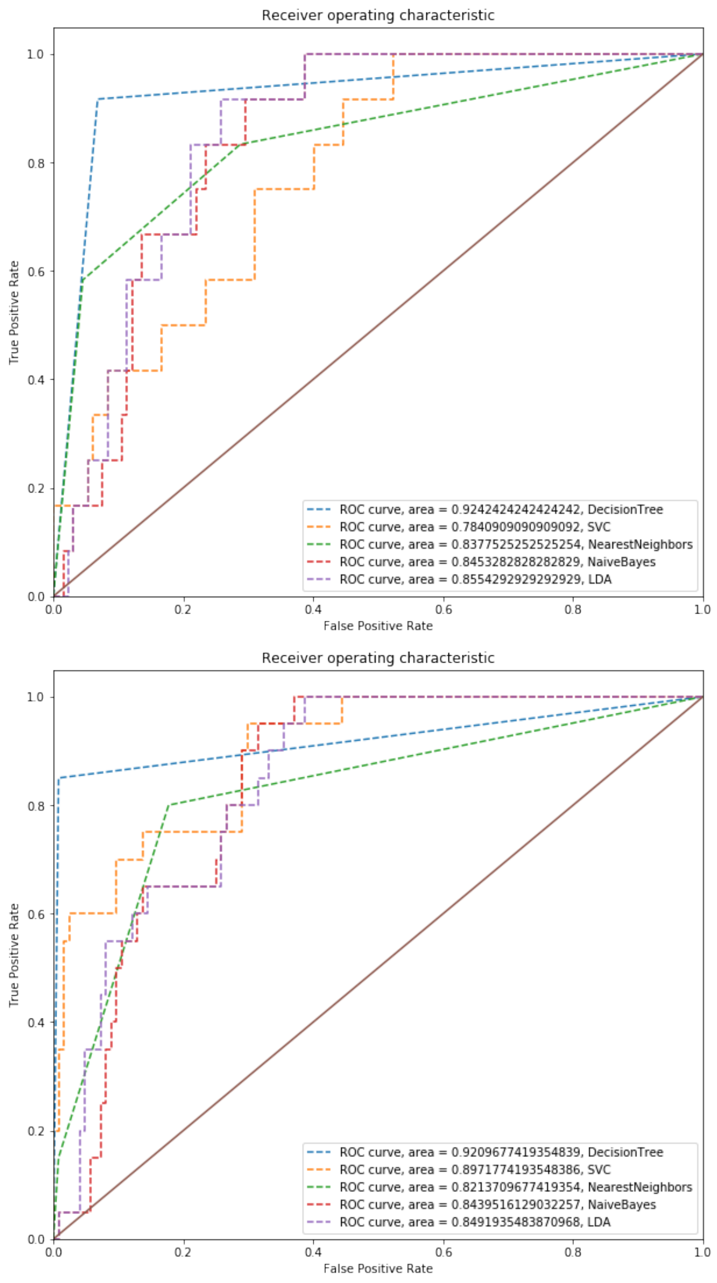


Figure 6.7: ROC curves of different methods applied to the result of Genetic Algorithm on the first clinical problem, with (up) and without (down) the additional clustering analysis.

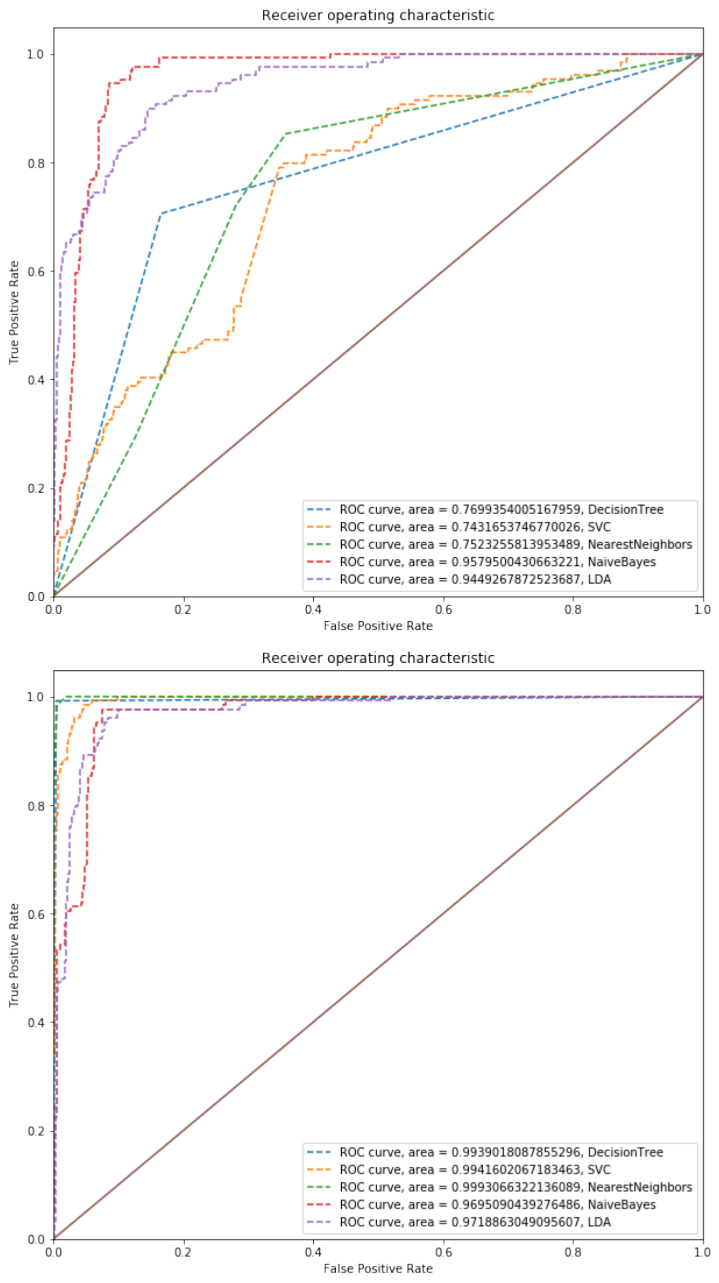
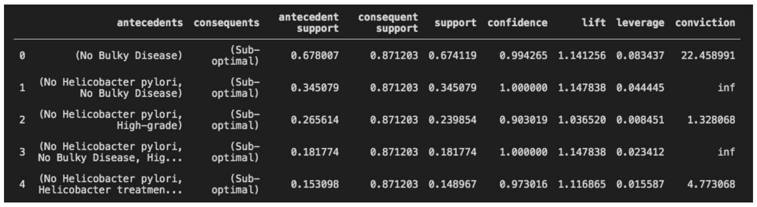


Figure 6.8: ROC curves of different methods applied to the solutions of the gastric NHL ID, with (up) and without (down) the additional clustering analysis.



```
                  antecedents  consequents  antecedent  consequent  support  confidence      lift  leverage  conviction
                                               support     support
0          (No Bulky Disease)        (Sub-    0.678007    0.871203 0.674119    0.994265  1.141256  0.083437   22.458991
                                  optimal)
1  (No Helicobacter pylori,          (Sub-    0.345079    0.871203 0.345079    1.000000  1.147838  0.044445         inf
        No Bulky Disease)         optimal)
2  (No Helicobacter pylori,          (Sub-    0.265614    0.871203 0.239854    0.903019  1.036520  0.008451    1.328068
              High-grade)         optimal)
3  (No Helicobacter pylori,          (Sub-    0.181774    0.871203 0.181774    1.000000  1.147838  0.023412         inf
  No Bulky Disease, Hig...        optimal)
4  (No Helicobacter pylori,          (Sub-    0.153098    0.871203 0.148967    0.973016  1.116865  0.015587    4.773068
  Helicobacter treatmen...        optimal)
```

Figure 6.9: Dataframe head of gastric NHL treatment association rules.

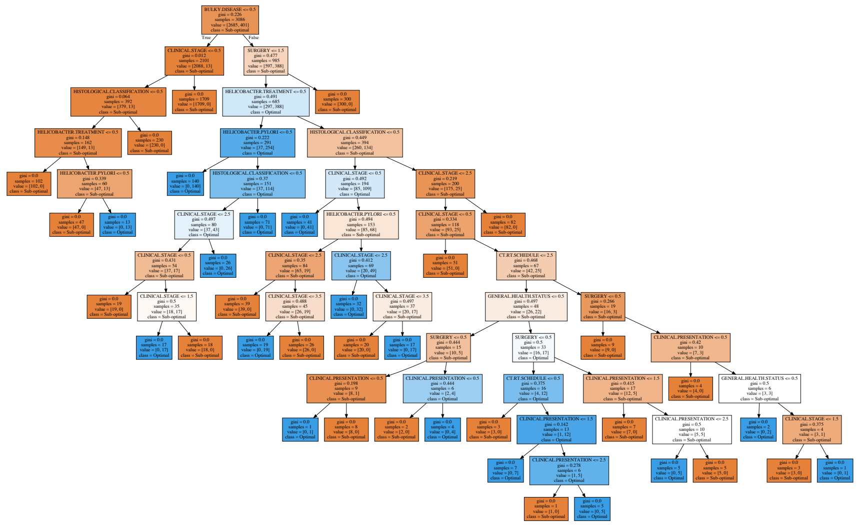


Figure 6.10: Decision tree on gastric NHL treatment.

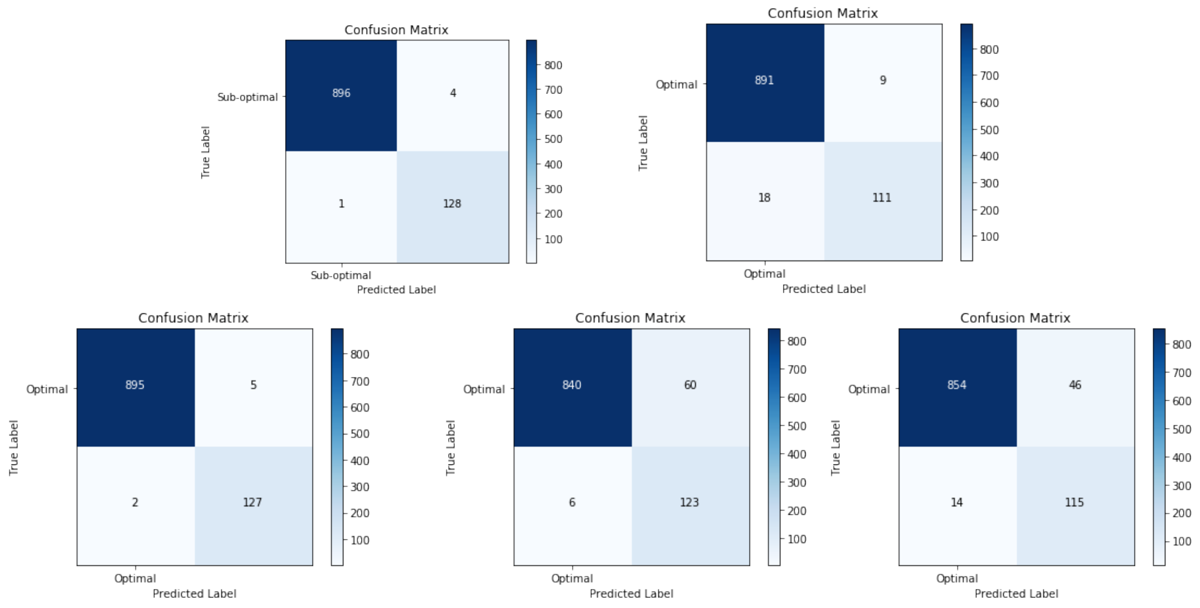


Figure 6.11: In clockwise order: Decision Tree, SVC, LDA, Naive Bayes, KNN confusion matrix of the optimal gastric NHL treatment.

REFERENCES

Ankur Ankan and Abinash Panda. pgmpy. 2015. `https://github.com/pgmpy`.

Daniel A. Schult Aric A. Hagberg and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. *Proceedings of the 7th Python in Science Conference (SciPy2008)*, 2008.

Hagai Attias. A variational bayesian framework for graphical models. *Neural Information Processing Systems*, 2000.

U. Bertele and F. Brioschi. *Nonserial Dynamic Programming.* 1972.

M. Betancourt. A conceptual introduction to hamiltonian monte carlo. 2017.

C. Bielza and J.A. Fernandez del Pozo. Dealing with complex queries in decision-support systems. *Data and Knowledge Engineering*, 2011.

S. Carstens. Introduction to markov chain monte carlo (mcmc) sampling, part 3: Hamiltonian monte carlo. 2020. `https://www.tweag.io/blog/2020-08-06-mcmc-intro3/`.

Xiaofei Shi Aniketh Janardhan Reddy Donghan Yu, Songwei Ge. Lecture 12: Theory of variational inference: Marginal polytope, inner and outer approximation. 2019. `https://sailinglab.github.io/pgm-spring-2019/notes/lecture-12/`.

Anthony D.; Pendleton Brian J.; Roweth Duncan Duane, Simon; Kennedy. Hybrid monte carlo. *Physics Letters B. 195*, 1987.

R. Durrett. *Essentials of Stochastic Processes.* 2011.

S. L. Lauritzen F. V. Jensen and K. G. Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 1990.

Maxwell Forbes. A tiny python library for factor graphs. `https://github.com/mbforbes/py-factorgraph`.

Brendan J. Frey and David MacKay. A revolution: Belief propagation in graphs with cycles. *Advances in Neural Information Processing Systems 10 (NIPS 1997)*, 1997.

Daunizeau J. Kilner J. Kiebel Friston, K. J. Taction and behavior: a free-energy formulation. *Biological Cybernetics 102*, 2010.

K. et al. Friston. Active inference and epistemic value. *Cognitive Neuroscience 6*, 2015.

Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *SOFTWARE - PRACTICE AND EXPERIENCE*, 30(11):1203–1233, 2000.

D. Geman, S.; Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *Transactions on Pattern Analysis and Machine Intelligence*, 1984.

Max Halford. Bayesian networks in python. 2020. `https://github.com/MaxHalford/hedgehog`.

Hammersley and Clifford. Markov fields on finite graphs and lattices. 1971. `www.statslab.cam.ac.uk/~grg/books/hammfest/hamm-cliff.pdf`.

W.K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 1970.

Tom Heskes. Stable fixed points of loopy belief propagation are minima of the bethe free energy. 2003.

J. Hohwy. The self-evidencing brain. 2016.

John H. Holland. Equation of state calculations by fast computing machines. *Scientific American Vol. 267*, 1992.

Finn V. Jensen. *Bayesian Networks and Decision Graphs*. 2001.

William T. Freeman Jonathan S. Yedidia and Yair Weiss. Constructing free energy approximations and generalized belief propagation algorithms. 2004.

Yair Weiss Jonathan S. Yedidia, William T. Freeman. Generalized belief propagation. *Proceedings of the 13th International Conference on Neural Information Processing System*, 2000.

Mudassir Khan. 2017. KMeans Clustering for Classification.

D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. 2009.

Kozlov and Koller. Nonuniform dynamic discretization in hybrid networks. *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, 1997.

R.A Kullback, S.; Leibler. On information and sufficiency. *Neural Information Processing Systems*, 1951.

S.L. Lauritzen and F. Jensen. Stable local computation with conditional gaussian distributions. *Statistics and Computing*, 1999.

S. Liu, J. McGree, Z. Ge, and Y. Xie. *Computational and Statistical Methods for Analysing Big Data with Applications*. 2016.

P.J. Lucas, H. Boot, and B.G. Taal. Computer-based decision support in the management of primary gastric non-hodgkin lymphoma. 1998.

J.N. Mandrekar. Receiver operating characteristic curve in diagnostic test assessment. *SOFTWARE - PRACTICE AND EXPERIENCE*, 2010.

Raghavan Manning and Schütze. *Introduction to Information Retrieval*. 2008.

Andrew Gelman Matthew D. Hoffman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. 2011.

Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

A.W.; Rosenbluth M.N.; Teller A.H.; Teller E. Metropolis, N.; Rosenbluth. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 1953.

N. Metropolis. The beginning of the monte carlo method. *Los Alamos Science (1987 Special Issue dedicated to Stanislaw Ulam*, 1987.

Tailor Neil and Marquez. Inference in bayesian networks using dynamic discretisation. *Statistics and Computing*, 2007.

Judea Pearl. Reverend bayes on inference engines: A distributed hierarchical approach. *Proceedings of the Second National Conference on Artificial Intelligence*, 1982.

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems*. 1998.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Carsten Peterson and James R. Anderson. A mean field theory learning algorithm for neural networks. 1987.

Dishan Gupta Rajarshi Das, Zhengzhong Liu. Notes on variational inference: Loopy belief propagation. 2014. `https://www.cs.cmu.edu/~epxing/Class/10708-14/scribe_notes/scribe_note_lecture13.pdf`.

Paulo Rauber. Probabilistic graphical models in python. `https://github.com/paulorauber/pgml`.

Ming Yang; Lihua Zhou; Huifeng Ruan. The methods for solving influence diagrams: A review. *International Conference on Information Technology and Applications*, 2013.

et al. S. Ermon. Cs228 notes. 2019. `https://github.com/ermongroup/cs228-notes`.

Rommel N. Carvalho S. Matsumoto. Unbbayes : a java framework for probabilistic models in ai. 2011.

A. Salmerón. A review of inference algorithms for hybrid bayesian networks. *Journal of Artificial Intelligence Research*, 2018.

Antonio Salmerón Serafín Moral, Rafael Rumí. Mixtures of truncated exponentials in hybrid bayesian networks. *Proceedings of the 6th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, 2001.

Ross D. Shachter. Evaluating influence diagrams. *Operations Research Vol. 34, No. 6*, 1986.

Shenoy and Shafer. An axiomatic framework for bayesian and belief function propagation. 1988.

Stan-Manual. Hamilton-monte-carlo. `https://mc-stan.org/docs/2_26/reference-manual/hamiltonian-monte-carlo.html`.

Stefan J. Kiebel Karl J. Friston Thomas Parr, Dimitrije Markovic. Neuronal message passing using mean-field, bethe, and marginal approximations. 2019.

S. Ermon V. Kuleshov. Notes on probabilistic graphical models. 2019. `https://ermongroup.github.io/cs228-notes/`.

Yair Weiss. Correctness of local probability propagation in graphical models with loops. *Neural Computation*, 2000.

John Winn and Christopher M. Bishop. Variational message passing. *Journal of Machine Learning Research 6*, 2005.