1. INTRODUCTION TO PROBABILISTIC MODELS

DENIS D. MAUÁ
NUSP: 3730790

Abstract. This document describes how to specify and query probabilistic models specified in generalized propositional logic. Sentences are composed of logical constructs (negation, disjunction, conjunction, variable assignment), represented as expression trees. Probabilistic models are represented by tables of numbers representing the distribution over valuations. Queries are computed by enumeration of valuations, and truth-table implication. The connection between the standard, Kolmogorovian formulation of probabilistic models over algebras and the formulation over (generalized) propositional logic is formally presented. The operations are implemented in a accompanying source code, which is also described here.

1. Reasoning under Uncertainty

There have been many attempts at representing and handling uncertain knowledge in a systematic way. Early approaches were based on monotonic logics such as propositional and first-order logic. The logical approaches had the advantage of representing knowledge modularly, and being self-consistent. Specifying knowledge in those formalisms required a high level of detail, one that was often absent in applications. The difficulty of representing incomplete or uncertain knowledge led to the creation of several logic-based formalisms that contained some calculus over beliefs. Ultimately, it was advocated that any formalism that did not comply with probability theory was doomed to produce inconsistent and incoherent results. Bayesian networks appeared in the 90's as a way of reconciling the modular character of pure logical approaches with the solid foundations of probability theory. In this report, we review the basics of probability theory. We adopt a formal language that generalizes propositional logic with multivalued variables.

2. Generalized Propositional Logic

In order to formally represent knowledge, we need to define a language. The basic ingredients of our language are variables (e.g., X, Y, Z) which denote properties or attributes of the domain. Each variable X takes values in domain $\mathrm{dom}(X)$. We also have logic connectives ($\wedge$, $\vee$, $\neg$) and variable assignment $X = x$, $x \in \mathrm{dom}(X)$. Consider a finite set of variables $X_1, \ldots, X_n$. A language $\mathcal{L}(X_1, \ldots, X_n)$ is inductively defined as follows.

(1) For any variable X and value $x \in \Omega_X$, $(X = x)$ is a sentence.
(2) If ϕ is a sentence, then so is $\neg(\phi)$.
(3) If ϕ_1 and ϕ_2 are sentences, then so is $(\phi_1 \wedge \phi_2)$ and $(\phi_1 \vee \phi_2)$.
(4) Nothing else is a sentence.

Date: February 22, 2016.

We often use the standard precedence over connectives ($\neg < \wedge < \vee$) and drop some of the parentheses in the formulas.

The meaning of complex sentences is based on the concepts of valuations and satisfaction. A *valuation* is a function ν that maps each variable X to a value $x \in \mathrm{dom}(X)$. A valuation describes a context or state of affairs of the domain. The meaning of an arbitrary sentence is given by the *satisfaction* relation $\models$ over pairs of valuation and sentence. It is standard to write $\nu \models \phi$ to indicate that the pair (ν, phi) is in the relation. We then say that ν satisfies ϕ (ϕ is satisfied by ν). We also write $\nu \not\models \phi$ to indicate that (ν, ϕ) is *not* in the relation. In this case, we say that ν does not satisfy ϕ (ϕ is not satisfied by ν). The satisfaction relation is inductively defined by:

(1) $\nu \models X = x$ if and only if $\nu(X) = x$.
(2) $\nu \models \neg\phi$ if and only if $\nu \not\models \phi$.
(3) $\nu \models \phi \wedge \psi$ if and only if $\nu \models \phi$ and $\nu \models \psi$.
(4) $\nu \models \phi \vee \psi$ if and only if $\nu \models \phi$ or $\nu \models \psi$.

It follows that $\nu \models \phi \vee \neg\phi$ for any valuation ν and sentence ϕ. A sentence $\phi \vee \neg\phi$ is called a *tautology*. Similarly, $\nu \models \phi \wedge \neg\phi$ for any valuation and sentence. A sentence $\phi \wedge \neg\phi$ is called a *contradiction*. Two sentences ϕ and ψ are *mutually exclusive* if $\nu \models \neg(\phi \wedge \psi)$ for every valuation ν. A set of sentences is mutually exclusive if they are pairwise mutually exclusive. A set of sentences $\phi_1, \ldots, \phi_n$ is *exhaustive* if each valuation satisfies at least one model, that is, if $\nu \models \phi_1 \vee \cdots \vee \phi_n$ for each ν.

3. Probabilistic Models

A probabilistic model is a pair $(\mathcal{L}(X_1, \ldots, X_n), \mathbb{P})$ where $\mathcal{L}$ is a language of valid sentences as defined, and $\mathbb{P}$ is a function from $\mathcal{L}$ to $[0, 1]$ such that

$$\sum_{\nu} \mathbb{P}(\nu) = 1\,, \tag{1}$$

where $\mathbb{P}(\nu)$ denotes $\mathbb{P}(X_1 = \nu(X_1) \wedge \cdots \wedge X_n = \nu(X_n))$, and

$$\mathbb{P}(\phi) = \sum_{\nu : \nu \models \phi} \mathbb{P}(\nu) \tag{2}$$

for any sentence ϕ. The function $\mathbb{P}$ is called a probability function or probability measure on $\mathcal{L}$. The following results are true for sentences ϕ and ψ.

(1) $\mathbb{P}(\neg\phi) = 1 - \mathbb{P}(\phi)$.
(2) $\mathbb{P}(\phi \vee \neg\phi) = 1$ and $\mathbb{P}(\phi \wedge \neg\phi) = 0$.
(3) $\mathbb{P}(\phi \vee \psi) = \mathbb{P}(\phi) + \mathbb{P}(\psi) - \mathbb{P}(\phi \wedge \psi)$.
(4) $\mathbb{P}(\phi) = \sum_i \mathbb{P}(\phi \wedge \psi_i)$ for mutually exclusive and disjoint events $\phi_1, \ldots, \phi_n$.

The *conditional probability* of ϕ given ψ is

$$\mathbb{P}(\phi|\psi) = \frac{\mathbb{P}(\phi \wedge \psi)}{\mathbb{P}(\psi)}\,, \tag{3}$$

whenever $\mathbb{P}(\psi) > 0$. In particular, the unconditional probability $\mathbb{P}(\phi)$ equals the conditional probability $\mathbb{P}(\phi|\phi \vee \neg\phi)$.

A *field of sets* is a pair $(\Omega, \mathcal{F})$ where $\mathcal{F}$ is a non-empty collection of subsets of Ω closed under intersection, union and complement. The elements of $\mathcal{F}$ are called *events* and Ω the universe.

The standard (Kolmogorov) definition of probabilistic models is based on field of sets. A probabilistic model (or probability space) is a triple $(\Omega, \mathcal{F}, \mathbb{P})$, where Ω is a set of *atomic events* (also called atoms, outcomes or worlds), and $(\Omega, \mathcal{F})$ is a field of sets and $\mathbb{P}$ is a real-valued (probability) function on $\mathcal{F}$ such that

(K1) $\mathbb{P}(\alpha) \geq 0$;
(K2) $\mathbb{P}(\Omega) = 1$;
(K3) $\mathbb{P}(\alpha \cup \beta) = \mathbb{P}(\alpha) + \mathbb{P}(\beta)$ for disjoint events α and β.

The two definitions are equivalent. To see this, consider a universe Ω whose elements are the valuations of a language $\mathcal{L}(X_1, \ldots, X_n)$ and define $\mathcal{F}$ to be the sets

$$\alpha_\phi = \{\nu \in \Omega : \nu \models \phi\}\,, \tag{4}$$

where $\phi \in \mathcal{L}$. By identifying intersection with conjunction, union with disjunction and negation with complement, it is easy to verify that $\mathcal{F}$ is indeed a field of sets. For example, for any events α_ϕ and α_ψ we have that

$$\alpha_\phi \cup \alpha_\psi = \{\nu \in \Omega : \nu \models \phi\} \cup \{\nu \in \Omega : \nu \models \psi\} \tag{5}$$

$$= \{\nu \in \Omega : \nu \models \phi \text{ or } \nu \models \psi\} \tag{6}$$

$$= \alpha_{\phi \vee \psi}\,. \tag{7}$$

The converse is also true. Associate with every atomic event a binary variable. Then every event in $\mathcal{F}$ is a union of variable assignments.

3.1. **Tree-structured representations of formulas.** Sentences can be efficiently represented and manipulated in a computer as trees. An expression tree is a binary tree whose internal nodes are labeled with logical connectives $(\wedge, \vee, \neg)$ and whose leaves are labeled with variable assignments (e.g., $X = x$). Trees with a single node represent variable assignments. The evaluation of an expression tree takes a valuation and returns a true/false value. IT is inductively defined as follows. The evaluation of a one-node tree $X = x$ with respect to a valuation ν is true if $\nu(X) = x$ and false otherwise. Let T be a tree whose root node is annotated with $\wedge$ and have children T_1 and T_2. The evaluation of T with respect to a valuation ν is true if T_1 and T_2 evaluates to true (w.r.t. ν) and false otherwise. The evaluation of trees with other connectives as root is analogous. A tree represents a sentence if a valuation satisfies the sentence if and only if the tree evaluates to true under that valuation.

4. Solution to the Exercises

4.1. **Exercise 1.** The following facts are true:

(1) For any event α, it follows that $\mathbb{P}(\alpha) = 1 - \mathbb{P}(\alpha^{\mathsf{c}})$, where α^{c} is the complement of α.
(2) $\mathbb{P}(\emptyset) = 0$.
(3) If $\alpha \subseteq \beta$ are events then $\mathbb{P}(\alpha) \leq \mathbb{P}(\beta)$. This is called *monotonicity*.
(4) For any event α, we have that $0 \leq \mathbb{P}(\alpha) \leq 1$.
(5) For any events α and β (not necessarily disjoint), we have that $\mathbb{P}(\alpha \cup \beta) = \mathbb{P}(\alpha) + \mathbb{P}(\beta) - \mathbb{P}(\alpha \cap \beta)$.
(6) If $\mathbb{P}(\alpha) = 1$ for some event α then $\mathbb{P}(\beta) = \mathbb{P}(\alpha \cap \beta)$ for any event β.

Write proof.

State and solve remaining exercises.

5. The ProbLogic Library

The ProbLogic library implements the basic mechanisms for representing and querying probabilistic models specified in generalized propositional logic with a finite vocabulary.

5.1. **API.** The basic functions and data types regarding probabilistic models are delared in `problogic.h`. A variable in the library takes non-negative integer values (e.g., $0, 1, 2, 3$), and is represented by the struct:

```
typedef struct {
  char* name; /* user-friendly name */
  int size;   /* domain size */
} variable;
```

A probabilistic model is represented as:

```
typedef struct {
  int n;           /* the size of the vocabulary */
  int size;        /* size of the event space */
  variable **vars;/* list of (pointers) variables */
  double *p;       /* the distribution of valuations */
} model;
```

The function `createModel` instantiates a new model over an list of variables. For example, a probabilistic model of two loaded coins which land heads with probability 0.2 can be specified by:

```
variable **vars = malloc(2*sizeof(variable*));
vars[0] = malloc(sizeof(variable));
vars[0]->name = malloc(6*sizeof(char));
vars[0]->name = "coin1";
vars[0]->size = 2; /* heads=1, tails=0 */
vars[1] = malloc(sizeof(variable));
vars[1]->name = malloc(6*sizeof(char));
vars[1]->name = "coin2";
vars[1]->size = 2;
model *m = createModel(2, vars);
m->p[0] = 0.64; /* coin1=0, coin2=0 */
m->p[1] = 0.16; /* coin1=0, coin2=1 */
m->p[2] = 0.16; /* coin1=1, coin2=0 */
m->p[3] = 0.04; /* coin1=1, coin2=1 */
```

For models over Boolean variables (i.e., variables take two values), the function `createBooleanModel` is convenient:

```
  int i;
  double p[] = {.576, .144, .064, .016, .008, .032, .032,
      .128};
  model *alarm = createBooleanModel(3);
  for (i=0; i < alarm->size; i++) alarm->p[i] = p[i];
  display(alarm);  /* shows useful information about model */
```

A sentence is represented by an expression tree. The data type and functions for building sentences (formulas) are declared in `expression.h`. For example:

```
formula *var1 = addVariable(0);
formula *var2 = addVariable(1);
formula *var3 = addVariable(2);
formula *not  = addNot(var3);
formula *and  = addOperation(AND, var2, not);
formula *or   = addOperation(OR, var1, and);
```

The `query` function in `problogic.h` computes the probability of a sentence.

```
formula **f;
formula *marginals[] = {var1, var2, var3, NULL};
for (f = marginals; *f != NULL; f += 1) {
  printf("Prob{"); printFormula(*f); printf("} = %.4f\n",
      query(*f, alarm));
}
printf("Prob{"); printFormula(or); printf("} = %.4f\n", query
    (or, alarm));
```

5.2. **User interface.** The file *main.c* reads a model and a query from file (or stdin) and outputs the corresponding values. Its executable is the file *query*. The format of the input file is given by the following grammar in EBNF:

input	:=	"model", { variable }, distribution, { query };
variables	:=	"var", string, "/", integer, "."
distribution	:=	"dist", { number }, "."
query	:=	"query", expression, "."

String, integer and number are defined trivially. Expression is a propositional logic expression (with the standard operator precedence).

An example is given in the file *alarm.model* provided with the package.