Applied Concepts of Probabilistic Programming

Olga Ivanova

Business informatics

E-Mail: ollyenn@gmail.com

ABSTRACT

Probabilistic programming is one of the auspicious, fast-growing fields of IT research, which is applicable in the context of machine learning. Probabilistic programming looks into the possibilities of mapping theoretical concepts of probability theory onto suitable practical programming techniques to handle uncertainty in data. This paper provides an overview of the applied concepts of probabilistic programming and main groups of probabilistic programming tools, as well as outlines the theoretical context and open issues of probabilistic programming.

KEYWORDS

Probabilistic system, reasoning pattern, evidence, inference, probability query, MAP.

INTRODUCTION

Probabilistic approach in programming has gained considerable attention of academic community over the last decade. The area of the research keeps growing, as the approach in question explores new ways and techniques of massive data processing and decision making. As B. Cronin remarks, "probabilistic programming languages are in the spotlight".[Cron] M. Hicks describes probabilistic programming as "an exciting, and growing, area of research", with "people in both AI/ML and PL working together and making strides".[Hicks]

There exist a number of reasons accounting for this rise of interest. However, the main of them comes down to the fact that constantly growing vast amount of data demands new techniques of automation, prediction, analysis and modelling. This ongoing search of new ways to make IT systems more intelligent and sophisticated has boosted the development of the whole domain of machine learning.

Handling uncertainty is one of numerous challenges machine learning is facing, as IT systems initially have been very restricted by means of processing these uncertainties. Applied concepts of probabilistic programming could provide machine learning with suitable tools for dealing with uncertainty of data, thus, enabling ML to combine the available knowledge of the subject or situation with mathematical probability rules.

There exist a number of definitions of what a probabilistic program or probabilistic programming in general is. Although none of them is universally applicable or standardized, most of them tend to share substantial similarity.

Probabilistic programs are defined by A. D. Gordon as "usual functional or imperative programs with two added constructs: (1) the ability to draw values at random from distributions, and (2) the ability to condition values of variables in a program via observations".[GordHenzNo] F. Wood asserts that "probabilistic programs are written with parts not fixed in advance that instead take values generated at runtime by random sampling procedures".[WoodMeMan] In a similar way N. D. Goodman remarks that probabilistic programming languages "in their simplest form ... extend a well-specified deterministic programming language with primitive constructs for random choice".[Good]

Given uncertain or incomplete knowledge the agent (whether human or not) is required in most cases to make an analysis of reliant data and make an assumption of it thus, restoring to some extent the missing information. As D. Koller states, "Most tasks require a person or an automated system to reason: to take the available information and reach conclusions, both about what might be true in the world and about how to act".[KollFried] According to A. Pfeffer, probabilistic programming is a way to create systems, supporting decision-making in the face of uncertainty. He also points out that probabilistic approach combines the knowledge of a situation with the laws of probability to determine those unobserved factors that are critical to the decision.[Pfeff3] Performing such tasks as preliminary analysis of bulk data as well as subsequent decision-making in case of incomplete knowledge entails an extensive use of applied probabilistic tools and techniques. Transferring the latter from the field of mathematical reasoning into the field of applied informatics implies describing probability distributions and providing ways of drawing probabilistic inference. This can be performed in the "traditional" imperative programming with the help of complex cumbersome control flows but as A. D. Gordon noticed, the purpose of probabilistic programming is to make probabilistic modelling accessible to a programmer without expert knowledge of probability theory, i.e. without revealing the details of inference implementation.[GordHenzNo]

BASIC COMPONENTS OF A PROBABILISTIC REASONING SYSTEM. KEY TERMS AND DEFINITIONS.

A probabilistic reasoning system presupposes a coherent interaction of its components. Despite the fact that approaches to separate out the components of such a system differ in the level of abstraction, the underlying principles and ideas bear a certain resemblance to each other.

According to A. Pfeffer, components of a probabilistic reasoning system generally include a probabilistic model, general and evidential knowledge, query and inference engine.

Probabilistic model is an integral part of any probabilistic reasoning system encompassing the most relevant information about specifics of the particular domain in a suitable form for formal processing. A. Pffefer describes a probabilistic model as "an encoding of general knowledge about a domain in quantitative, probabilistic terms". He also stresses that "each model has an element of inherent randomness".[Pfeff3] D. Koller also points out that a probabilistic model "encodes our knowledge of how the system works in a computer-readable form".[KollFried] In terms of programming a probabilistic model consists of variables, dependencies between these variables, numerical parameters as values of these variables and the so-called functional forms of dependencies (e.g. the possibility of modelling as the result of a coin toss with a certain weight).[Pfeff2]

B. Cronin emphasizes the importance of "clean separation between modelling and inference", as it "can vastly reduce the time and effort associated with implementing new models and understanding data". According to his comparison, probabilistic languages can "free the developer from the complexities of high-performance probabilistic inference" the way "high-level programming languages transformed developer productivity by abstracting away the details of the processor and memory architecture".[Cron] In other words, loose coupling between the model and the inference engine enables the system to process different models and thus, serve as a generic tool.

A. Pfeffer differentiates between general knowledge embracing "what you know to hold true of your domain in general terms, without considering the details of a particular situation" and evidence as "specific information about a particular situation".[Pfeff3]

Query is elucidated as a property of some particular situation which is looked for. In this interpretation probabilistic inference is defined as "the process of using the model to answer queries based on the evidence".[Pfeff3]

It is also of importance that the relations of all components of any probabilistic reasoning system strictly comply with mathematical laws of probability. General, and evidential information is treated in "quantitative, probabilistic terms".[Pfeff2]

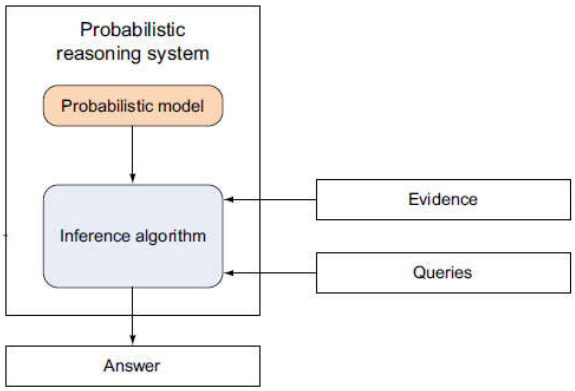

Figure 1: Probabilistic Reasoning System [Pfeff2]

A similar approach has been given by D. Koller and N. Friedman, who singled out three major components or rather layers in any complex reasoning system, namely representation, inference and learning. In their perception, declarative representation is a reasonable encoding of the world model, used to answer the questions of interest. Inference is viewed as "answering queries using the distribution as our model of the world" in particular that process is carried out by "computing the posterior probability of some variables given evidence on others".[KollFried]

As far as the stage of learning is concerned, D. Koller and N. Friedman assume that models can be constructed either with the help of a human expert or automatically, "by learning from data a model that provides a good approximation to our past experience". Furthermore, data-driven approach to model construction with human experts setting only guidelines for an automatic supply of details, was acknowledged to be more effective compared to purely human-constructed.[KollFried]

A. Pfeffer remarks that as far as learning is concerned, there are two main things to be done. The most obvious one is to "learn from the past to improve your general knowledge" and as a result "to better predict a specific future situation". However, it is also possible to go further learning from the past, that is "to improve the model itself", especially if "a lot of past experiences to draw on" is available. In this case the goal of a learning algorithm is to produce a new model, not to answer queries. The learning algorithm begins with the original model and updates it based on the experience to produce the new model. The new model can be used then to answer queries in the future in a "better-informed" way. [Pfeff2]

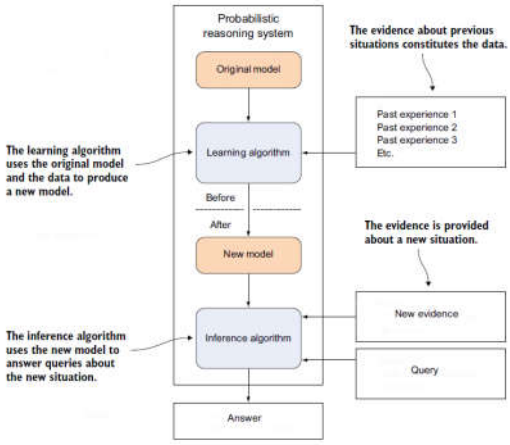

Figure 2: Probabilistic Reasoning System with a Learning Component [Pfeff2]

N. D. Goodman and J. B. Tenenbaum developed an approach to the study of probabilistic systems on the basis of the concept "generative models", which represent knowledge about the causal structure of the world in a simplified form. A generative model in this theory is used to describe some process of the reality, which produces observable data. Probabilistic generative models are defined as models of processes, "which unfold with some amount of randomness" and can be used to inquire about these processes with the help of probabilistic inference. The main idea here is to deal with a process with uncertainty as with computation, which involves random choices. The simulation of such processes is inevitably connected with the degree of belief, as the expected outcomes are formalized as probability distribution. [GoodTenen]

OVERVIEW OF EXISTING PROBABILISTIC SYSTEMS AND TOOLS

The growing interest in the probabilistic programming approach within the academic community has resulted in the emergence of new languages and frameworks designed and implemented to perform tasks specific to the domain of probabilistic programming. The wiki-list of existing probabilistic programming systems contains more than 20 entries.[WikiPP]

Despite numerous differences concerning the paradigm and implementation, probabilistic languages and libraries have much in common. The most important similarity between these languages relates to their common purpose, namely to "allow programmers to freely mix deterministic and stochastic elements" and "to specify a stochastic process using syntax that resembles modern programming languages".[WinStuhGood]

N. D. Goodman observes that probabilistic languages "provide compositional means for describing complex probability distributions", "provide generic inference engines: tools for performing efficient probabilistic inference over an arbitrary program" and points out that "in their simplest form, probabilistic programming languages extend a well-specified deterministic programming language with primitive constructs for random choice".[Good]

B. Cronin defines a probabilistic programming language as "a high-level language that makes it easy for a developer to define probability models and then "solve" these models automatically" and remarks that these kind of languages "incorporate random events as primitives and their runtime environment handles inference".[Cron]

D. Poole mentions that "most of the work in probabilistic programming languages has been in the context of specific languages". He tries to abstract from thorough consideration of specific probabilistic programming languages and to focus on the design of them, singling out three additional features, which are inherent in all probabilistic programming languages:

- conditioning as the ability to make observations about some variables in the simulation and to compute the posterior probability of arbitrary propositions with respect to these observations.
- inference
- learning as the ability to learn probabilities from data.[Poole]

As far as the implementation paradigm is concerned, most authors divide the existing probabilistic programming languages and systems into several major groups.

According to A. Pfeffer, some languages belong to the logic-based group (PRISM, BLOG, Markov Logic), others constitute the groups based on principles of functional (IBAL, Church) and imperative (FACTORIE, Picture) programming paradigms. Object-oriented approach is also stated to have several advantages in the domain of probabilistic programming and Figaro is mentioned as an example of an object-oriented probabilistic language.[Pfeff1]

A. D. Gordon, T. A. Henzinger, A. V. Nori also single out three paradigms in the diversity of probabilistic languages: imperative, functional, and logical. PROB, Infer.NET are mentioned as examples of imperative languages, functional paradigm is the base for BUGS, IBAL and Church, whereas probabilistic logic languages include BLOG, Alchemy, and Tuffy.[GordHenzNo]

It should also be pointed out, that most authors use the notions "probabilistic programming language" and "probabilistic programming system" interchangeably. To be precise, only a small number of the existing probabilistic programming systems are Turing complete programming languages such as Venture. Most of them present an extension (e.g. Church extending Scheme with probabilistic semantics, ProbLog extending Prolog) or a framework (e.g. Infer.NET for C#, PFP for Haskell) of an existent general purpose language.

Despite the apparent variety of the existing probabilistic programming systems (both languages and frameworks), the experimental character of the majority of them might present a certain difficulty when used in a real-life project for applied rather than academic purposes. In this case, pure probabilistic languages are

placed in an unfavourable position compared to the frameworks and libraries extending general purpose languages, because of the restricted number of their users as well as lack of community knowledge and support.

INTERPRETATION OF PROBABILITY

The notion of probability belongs to the fundamental concepts of probabilistic programming. However, the interpretation of probability is not always unambiguous. Generally speaking, there exist two common interpretations of probability, namely frequentist and Bayesian (or subjective).

D. Koller writes, "The frequentist interpretation views probabilities as frequencies of events. More precisely, the probability of an event is the fraction of times the event occurs if we repeat the experiment indefinitely". [KollFried] K. Murphy observes that in frequentist interpretation "probabilities represent long run frequencies of events". [Murphy] This interpretation suits for describing events that can be repeated a number of times, like flip of a coin, random choice of a card, etc. However, it can become problematic to describe the probability of an occurrence of a one-time time event in future (e.g. the probability of precipitation the next day, stock exchange course tomorrow, etc.). D. Koller remarks, "several attempts have been made to define the probability for such an event by finding a reference class of similar events for which frequencies are well defined; however, none of them has proved entirely satisfactory". [KollFried] That is when the subjective (Bayesian) interpretation comes into play. D. Koller describes probabilities as "subjective degrees of belief" within this interpretation, observing that "the statement $P(\alpha) = 0.3$ represents a subjective statement about one's own degree of belief that the event α will come about". [KollFried] Similarly, K. Murphy notes, "in this [Bayesian] view, probability is used to quantify our uncertainty about something; hence it is fundamentally related to information rather than repeated trials", adding that a major advantage of Bayesian approach is that it provides means to "model our uncertainty about events that do not have long term frequencies". [Murphy]

Although the subjective interpretation enables the quantification of uncertainty of events that happen zero or one time but can hardly happen repeatedly, the approach still possesses certain flaws. D. Koller criticizes it for being unable to determine "what exactly it means to hold a particular degree of belief". The source of the problem in her view is that "we need to explain how subjective degrees of beliefs (something that is internal to each one of us) are reflected in our actions". She suggests employing indirect ways of attributing degrees of beliefs (e.g. by a betting game) where it is possible. Nevertheless it is also pointed out that "both interpretations lead to the same mathematical rules" and as a result of that "the technical definitions hold for both interpretations". [KollFried]

K. Murphy also notes that "the basic rules of probability theory are the same, no matter which interpretation is adopted" and chooses the Bayesian interpretation for his research. [Murphy]

S. J. Russell and P. Norvig single out three main interpretations of probabilities, namely frequentist, objectivist and subjectivist and describe them as follows. According to the frequentist position "the numbers can come only from experiments. The objectivist view is that probabilities are real aspects of the universe - propensities of objects to behave in certain ways - rather than being just descriptions of an observer's degree of belief. In this view, frequentist measurements are attempts to observe the real probability value. The subjectivist view describes probabilities as a way of characterizing an agent's beliefs, rather than having any external physical significance. [RussNor]

S. J. Russell and P. Norvig note, "in the end, even a strict frequentist position involves subjective analysis, so the difference probably has little practical importance". It is also pointed out that the total refusal of subjective methods will inevitably result in the reference class problem. The reference class problem arises when everything is known about an object. That makes the object unique and, as a result, devoid of any reference class, needed to collect experimental data. That was characterised as "a vexing problem in the philosophy of science". [RussNor]

Summing up, it is obvious that the pure frequentist approach of defining probabilities is not sufficient in a number of cases, where an event cannot take place multiple times. Moreover, a certain degree of subjectivity is unavoidable at the stage of singling out relevant properties of an object to be able to assign it to a reference class. No matter which interpretation of probabilities is chosen for a particular case, the most important thing is compliance with the rules of probability theory.

REASONING PATTERNS

Probabilistic reasoning systems are characterized by a high degree of flexibility. The latter is essential, as the system should enable to query about different aspects/properties of a particular probabilistically modelled situation given evidence about other aspects or properties. Approaches to differentiation between types of inference vary in literature.

According to A. Pfeffer, there exist three kinds of reasoning that probabilistic systems can do:

1. Predict future events. The evidence will typically consist of information about the current situation.
2. Infer the cause of events. The evidence here is the same as before, together with an additional fact that the event of interest has happened.
3. Learn from past events to better predict future events. The evidence includes all evidence from last time (making a note that it was from last time), as well as the new information about the current situation. In

answering the query, the inference algorithm first infers properties of the situation that led to the present. It then uses these updated properties to make a prediction.

The third type of reasoning is characterized by A. Pfeffer as "a kind of machine learning".[Pfeff3]

D. Koller introduces more formal terminology in this context. According to her view, queries with predicted "downstream" effects of various factors are instances of causal reasoning or prediction, whereas queries, where one reasons from effects to causes, are instances of evidential reasoning or explanation. It must be pointed out that D. Koller's interpretation of the first two types of reasoning is equivalent to that of A. Pfeffer. The third pattern of reasoning examined by D. Koller is intercausal reasoning, "where different causes of the same effect can interact". The subtype explaining away is treated as an instance of intercausal reasoning. However, D. Koller remarks that "explaining away is not the only form of intercausal reasoning" and that "the influence can go in any direction".[KollFried]

S. J. Russell and P. Norvig differentiate between four distinct types of inference, namely:

- diagnostic inferences: from effects to causes
- causal inferences: from causes to effects
- intercausal inferences: between causes of a common effect (also known as explaining away)
- mixed inferences: combining two or more of the above.[RussNor]

N. D. Goodman and J. B. Tenenbaum, however, adopt a considerably different approach to differentiation types of reasoning. Causal relations are considered to be the basic type, encoding the knowledge of the dependencies in the real world within causal models. They are described as "local, modular, and directed". It is further elaborated that a causal structure is local in the sense that many related events are not related directly, but rather are connected only through causal chains of several steps, a series of intermediate and more local dependencies. Causal relations are also described as modular in the sense that any two arbitrary events in the world are most likely to be causally unrelated, or independent. Causal relations are always directed, as causal influence flows only one way along a causal relation.[GoodTenen]

Causal dependence is opposed to statistical dependence or correlation. According to N. D. Goodman and J. B. Tenenbaum, two events may be statistically dependent even if there is no causal chain running between them, as long as they have a common cause (direct or indirect).

e.g. Cough and fever are not causally dependent but they are statistically dependent, because they both depend on cold.[GoodTenen]

However, events that are considered to be statistically dependent a priori may become independent when conditioned on some other observation; this is called screening off, or context-specific independence. Also, events that are statistically independent initially may become dependent when conditioned on other observations; this is known as explaining away.[GoodTenen]

Screening off, as stated by N. D. Goodman and J. B. Tenenbaum, implies that if the statistical dependence between two events A and B is only indirect, mediated strictly by one or more other events C, then observing C should render A and B statistically independent. This can occur if events A and B are connected by one or more causal chains, and all such chains run through the set of events C, or if C comprises one or more common causes of A and B.

In case of explaining away if two events A and B are statistically independent, but they are both causes of one or more other events C, then conditioning on C can render A and B statistically dependent.[GoodTenen]

The main types of probabilistic inference can be illustrated with the example of a student's work as follows:

- Causal reasoning implies that a hard working student is more likely to understand the material, which in turn makes them more likely to be successful with their homework grade.
- According to the evidential reasoning, flowing in the opposite direction, observing a high mark of the student's homework provides evidence that the student understood the material, which in turn increases the probability that the student works hard.
- The case of mixed reasoning (composed of the causal and evidential types) presupposes that if a student earned a good exam grade, that provides evidence, that they understood the material, which in turn makes it more likely that they also received a high homework grade. However, it must be pointed out that the nodes "Exam Grade" and "Homework Grade" are conditionally independent given the node "Understands Material". In other words, if it is already known that the student understands the material, then the fact of the student's receiving a good exam grade does not deliver any new information about the homework grade.
- In case of intercausal reasoning, also called explaining away, if the value of the node "Understands Material" (as a common effect) is unknown, then values of the nodes "Smart" and "Hard working" are independent. However, if it is known that "Understands Material" is true, then the fact of "Smart" being true reduces the probability that "Hard working" is true.[CS181-Lec]

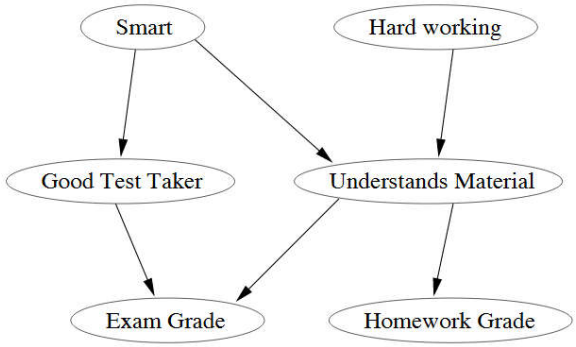

Figure 3: Student's work [CS181-Lec]

To summarize, causal, evidential, mixed and intercausal patterns of reasoning seem to be essential means of constructing relationships within a probabilistic reasoning system. Although the details of different classifications of reasoning patterns (such as naming and nesting of elements) tend to vary, the pragmatic logic behind them shows a certain degree of similarity.

TYPES OF QUERIES (PROBABILITY QUERIES AND MAPS)

According to D. Koller and N. Friedman, two main types of queries can be singled out in the probabilistic context, namely probability queries and MAP (maximum a posterior) or MPE (Most Probable Explanation) queries.[KollFried] K. Karkera's classification adheres to the same types.[Kark]

D. Koller characterizes probability queries as "perhaps the most common query type", which is comprised of two types:

- the evidence: a subset E of random variables in the model, and an instantiation e to these variables;
- the query variables: a subset Y of random variables in the network.[KollFried]

According to D. Koller and N. Friedman, the task consists in the computation of $P(Y \mid E = e)$, "that is, the posterior probability distribution over the values y of Y, conditioned on the fact that $E = e$. This expression can also be viewed as the marginal over Y, in the distribution we obtain by conditioning on e".[KollFried]

However, there exist situations, when the most probable result is of interest. That is where MAP queries, also known as MPE queries come in play.

A. Pfeffer describes MPE query as the one to be employed, when it is needed "to know the world that is the most probable explanation of the data", noting that "sometimes, rather than knowing a probability distribution over outcomes, you want to know which outcomes are the most likely". He underscores that "the goal of probabilistic inference in this case can be to find out the most likely state of the system", because "identifying the most likely state tells you the most likely cause of the problems you're seeing". So, according to A. Pfeffer, MPE query is "the query that tells you the most likely state of variables in the model".[Pfeff2]

According to L. E. Sucar, "the MPE or abduction problem consists in determining the most probable values for a subset of variables (explanation subset) in a BN given some evidence". It is also underlined that "the MPE is not the same as the union of the most probable value for each individual variable in the explanation subset".[Suc]

D. Koller considers MAP queries to fulfil "a second important type of task" consisting in "finding a high-probability joint assignment to some subset of variables". So, according to D. Koller, MPE query's aim is "to find the MAP assignment - the most likely assignment to all of the (non-evidence) variables" or if defined more formally:

if we let $W = X - E$, our task is to find the most likely assignment to the variables in W given the evidence $E = e$:

$MAP(W \mid e) = \text{argmax}_w P(w, e)$, where, in general, $\text{argmax}_x f(x)$ represents the value of x for which f(x) is maximal.[KollFried]

Addressing the difference between probability queries and MAP queries, D. Koller states, "in a MAP query, we are finding the most likely joint assignment to W. To find the most likely assignment to a single variable A, we could simply compute $P(A \mid e)$ and then pick the most likely value. However, the assignment where each variable individually picks its most likely value can be quite different from the most likely joint assignment to all variables simultaneously".[KollFried]

Likewise, K. Karkera defines MAP as "the highest probability joint assignment to some subsets of variables", emphasizes that "the MAP assignment cannot be obtained by simply taking the maximum probability value in the marginal distribution for each random variable" [Kark] and illustrates it with the following example.

e.g. There are two non-independent random variables X and Y, where Y is dependent on

X. The MAP assignment for the random variable X is X_1 since it has a higher value.

Table 1: Probability Distribution over X [Kark]

X_0	X_1
0.4	0.6

Table 2: Probability Distribution P(Y | X) [Kark]

P(Y | X)	Y_0	Y_1
X_0	0.1	0.9
X_1	0.5	0.5

However, the MAP assignment to random variables (X, Y) in the joint distribution is (X_0, Y_1), and the MAP assignment to X (X_1) is not a part of the MAP of the joint assignment.

Table 3: The Joint Distribution over X and Y [Kark]

Assignment	Value
X_0, Y_0	0.04
X_0, Y_1	0.36
X_1, Y_0	0.3
X_1, Y_1	0.3

The marginal MAP query can be regarded as a more general query type, which consists of "elements of both a conditional probability query and a MAP query". [KollFried]
Thus, with a subset of variables Y of the query and with the task to find the most likely assignment to the variables in Y given the evidence $E = e$ and $Z = X - Y - E$:
$MAP_{mar}(Y \mid e) = argmax_Y \sum_Z P(Y, Z \mid e)$, marginal MAP contains "both summations and maximizations".[KollFried]

PROBABILISTIC GRAPHICAL MODELS

Probabilistic programming needs a formal representation of real life situations to perform reasoning under uncertainty. Introduction of variables denoting the quantified knowledge of the situation, its agents and objects is an essential step to enable this kind of reasoning. As D. Koller remarks "domains can be characterized in terms of a set of random variables, where the value of each variable defines an important property of the world", emphasizing that "the set of possible variables and their values is an important design decision, and it depends strongly on the questions we may wish to answer about the domain".[KollFried] However, the introduction of variables formally representing elements of a particular situation is not sufficient for building a viable model of this situation. It is also the interaction of the elements, their mutual influence that needs to be reflected in the model. In other words, there should be means of encoding dependencies. As A. Pfeffer states, "dependencies capture relationships between variables" and he singles out two general kinds of them, namely "directed dependencies, which express asymmetric relationships, and undirected dependencies, which turn into symmetric relationships", pointing out that "probabilistic models essentially boil down to a collection of directed and undirected dependencies".[Pfeff2] The two main frameworks that are used for this kind of dependency-encoding are Bayesian networks and Markov networks, expressing directed and undirected dependencies respectively.
A. Pfeffer treats the Bayesian network as "a representation of a probabilistic model consisting of three components:
1. A set of variables with their corresponding domains. The domain of a variable specifies which values are possible for that variable.
2. A directed acyclic graph in which each variable is a node.
3. For each variable, a conditional probability distribution (CPD) over the variable given its parents.

A CPD specifies a probability distribution over the child variable given the values of its parents. A CPD considers every possible assignment of values to the parents, when the value of a parent can be any value in its domain. For each such assignment, it defines a probability distribution over the child. When a variable has no parents, the CPD just specifies a single probability distribution over the variable".[Pfeff2]
e.g. The following simple model contains five random variables with corresponding CPDs: the student's intelligence, the course difficulty, the grade, the student's SAT score, and the quality of the recommendation letter.

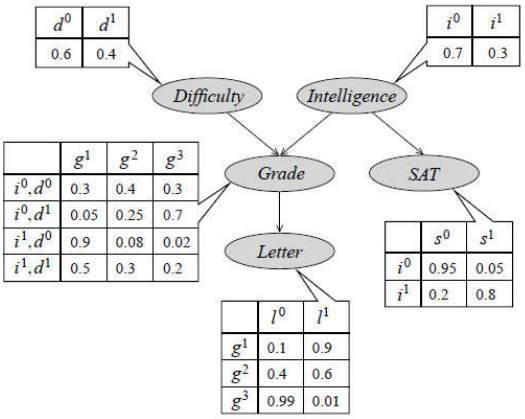

Figure 4: Student Bayesian network [KollFried]

A Markov network is defined by A. Pfeffer as a representation of a probabilistic model consisting of three things:
1. A set of variables. Each variable has a domain, which is the set of possible values of the variable.
2. An undirected graph in which the nodes are variables. The edges between nodes are undirected. This graph is allowed to have cycles.
3. A set of potentials, providing the numerical parameters of the model.[Pfeff2]
As opposed to Bayesian networks, where each variable is characterised by a CPD, variables in Markov networks do not have their own numerical parameters. The interaction between variables can be represented and quantified with the help of a function called a potential. As stated by A. Pfeffer, "When there's a symmetric dependency, some joint states of the variables that are dependent on each other are more likely than others, all else being equal. The potential specifies a weight for each such joint state. Joint states with high weights are more likely than joint states with low weight, all else being equal. The relative probability of the two joint states is equal to the ratio between their weights, again all else being equal".[Pfeff2] He defines a potential as "simply a function from the values of variables to real numbers", stressing the fact that only positive real numbers or zero are allowed as the values of a potential". Describing the interaction of potential functions with the graph structure, A. Pfeffer singles out two main rules, namely:

1. A potential function can only mention variables that are connected in the graph.
2. If two variables are connected in the graph, they must be mentioned together by some potential function.[Pfeff2]

e.g. The following model describes students teamwork in pairs. The following pairs can work well together: Alice and Bob; Bob and Charles; Charles and Debbie; and Debbie and Alice. Each interaction is described with a factor. For instance, $\varphi_1(\mathrm{A}, \mathrm{B})$ means that Alice and Bob tend to agree with each other.

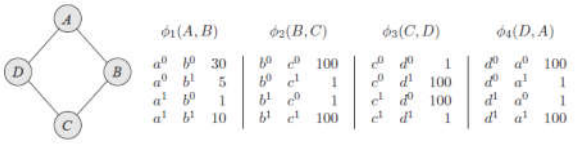

Figure 5: Students Teamwork Model [KollFried]

INFERENCE: MAIN GROUPS OF INFERENCE ALGORITHMS

The variety of algorithms dealing with the task of drawing inference can be examined in different perspectives. Thus, for example, A. Pfeffer puts stress on differentiating between factored and sampling algorithms, which he defines as follows:

- Factored algorithms - group of algorithms that operate on data structures called factors that capture the probabilistic model being reasoned about (e.g. Variable Elimination (VE) algorithm and Belief Propagation (BP) algorithm).
- Sampling algorithms are algorithms creating examples of possible worlds from the probability distribution and using those examples to answer queries (MCMC algorithms).[Pfeff2]

K. Karkera makes use of juxtaposition of exact inference (e.g. Variable Elimination, Tree Algorithms) and approximate inference methods (MCMC group), parallelly addressing the problem of complexity of inference tasks with the words, "even approximate inference is NP-hard". He notes, that inference might seem to be "a hopeless task, but that is only in the worst case" and that generally exact inference can successfully serve "to solve certain classes of real-world problems (such as Bayesian networks that have a small number of discrete random variables)", whereas approximate inference is required "for larger problems".[Kark]

Other scholars, such as D. Koller, S. J. Russel and P. Norvig also hold to the classification of inference algorithms in two major groups, namely exact inference algorithms (with VE and clustering algorithms as classical examples) and approximate inference algorithms, including a family of sampling methods.

It must be pointed out that the choice of a suitable inference algorithm depends on the structure of the model. For example, A. Pfeffer remarks that since Variable Elimination "is an exact algorithm that perfectly computes the probabilities", one might "think that it's not suitable for real-world applications with complex models", which is not the case. Variable Elimination is frequently employed, as long as the model has the right structure. In particular, it is of importance whether variables can be eliminated "without adding too many edges to the VE graph, leaving the size of the largest clique [set of nodes in a graph, which are all connected with each other] in the VE graph small and the complexity low". Hidden Markov models with a possible application of speech recognition and also parse trees in natural language processing are among structures, which allow running inference with VE.[Pfeff2]

According to A. Pfeffer, an approximate algorithm Belief Propagation has fewer limitations of use compared to VE and for a "model with discrete variables, BP is a good candidate technique to use". Possible applications of BP include Markov networks for image analysis and loopy Bayesian networks for medical diagnostics. The higher applicability of BP is a result of the fact that BP "operates using the moral graph (the initial VE graph), without adding edges". However, since adding these edges is necessary for correct inference, not adding them will result in errors. Nevertheless, inference can be approximately correct with a certain error margin even when these edges aren't added. [Pfeff2]

D. Koller also emphasizes the importance of choosing the right algorithm, as she addresses the problem of inference complexity, noting that "exponential blowup of the inference task is (almost certainly) unavoidable in the worst case: the problem of inference in graphical models is NP-hard, and therefore it probably requires exponential time in the worst case. Even worse, approximate inference is also NP-hard".[KollFried] Nevertheless, she also stresses, "the story does not end with this negative result. In general, we care not about the worst case, but about the cases that we encounter in practice" and that "many real-world applications can be tackled very effectively using exact or approximate inference algorithms for graphical models".[KollFried]

PROBABILISTIC OBJECT-ORIENTED KNOWLEDGE REPRESENTATION

Object-oriented programming paradigm has become an inalienable part of the modern landscape of software development. Hence, it is appropriate to consider most general concepts of exercising probabilistic programming in the context of OOP.

A. Pfeffer accentuates the following two advantages of OOP, namely

- Providing structure to complex programs. Objects are coherent units that capture a set of data and behaviors. An object provides a uniform interface to these data and behaviors, and the internals of the object are encapsulated from the rest of the program. This allows the programmer to modify the internals of the

object in a modular way, without affecting the rest of the program.
- Enabling reuse of code. First, the same class code, with all its internal structure, can be reused for all instances of the class. Second, inheritance makes it possible to reuse common aspects of different classes. [Pfeff2]

As A. Pfeffer perceptively states, OOP paradigm might be "even more appropriate for probabilistic models", since probabilistic programming deals with building models of the real world, which can be naturally described "in terms of objects".

As far as elements of object-oriented modelling are concerned, A. Pfeffer mentions "classes that describe general data and behaviours" and "instances of those classes that contain specific data and instantiations of those behaviors", characterizing both as follows:

- A probabilistic class model defines a general process for generating the values of random variables.
- An instance is a specific instantiation of this general class model that describes a process to generate the values of random variables that pertain to this specific instance. [Pfeff2]

e.g. The following code snippet written in Figaro, Scala defines a class CellPhone with 3 attributes. Atomic elements of Figaro Flip and Select are used for initializing of these attributes. myCellPhone is a particular instance of the class named CellPhone.

```
class CellPhone {
        val isOn = Flip(0.90)
        val withinCoverageArea = Flip(0.93)
        val connectionQuality = Select(0.6 -> "Medium", 0.3 -> "Well", 0.1-> "Poor")
}
val myCellPhone = new CellPhone
```

A set of probabilistic class definitions provide general definitions of random processes that can be reused many times for different instances the same way any OOP class model can be reused for many different instances in the "conventional" programming.

Thus, according to A.Pfeffer a probabilistic program following the concepts of OOP includes three phases:

- definition of the class models,
- creation of instances of those classes,
- reasoning with the instances

e.g. In the following code snippet, written in Figaro, Scala an instance of the importance sampling algorithm is created with parameters of 1000 samples and targetInformation property to be predicted. The algorithm is started explicitly and after all necessary actions are completed, the resources taken by algorithm are freed and cleaning up is performed.

```
val algorithm = Importance(1000, model.targetInformation)
algorithm.start()
// other actions
algorithm.kill()
```

A. Pfeffer also puts emphasis on the purposes of class probability models in a relational probability model:

1. To describe the structure of the model, including the classes in the model, their attributes, and relationships between classes.
2. To define the probabilistic dependencies, functional forms, and numerical parameters that govern the probabilistic model.

When defining the dependencies it should be kept in mind that "an attribute of an instance can depend on other attributes of that instance or on attributes of related instances". [Pfeff2]

In the paper "Object-Oriented Bayesian Networks", D. Koller and A. Pfeffer have made an attempt to describe an object-oriented Bayesian network (OOBN) language, which pursues the purpose of characterizing complex subject domains in terms of "interrelated objects". A Bayesian network fragment is used "to describe the probabilistic relations between the attributes of an object", while "these attributes can themselves be objects, providing a natural framework for encoding part of hierarchies". [KollPfeff]

An object is considered by D. Koller and A. Pfeffer to be the basic element of OOBN, with a standard random variable being the most basic object. However, more complex objects are also possible in this context. Typically, an object possesses a set of attributes, "each of which is an object" itself. As D. Koller and A. Pfeffer write: "One way of viewing an object is as a collection of properties that are associated with some entity in our domain. An object will sometimes correspond to a physical entity in the world being modelled, but it may also represent an abstract entity, or a relationship between different entities". The overall assignment of all values to the corresponding attributes of a particular object is treated by D. Koller and A. Pfeffer as the value of this object. Thus, a probabilistic model is defined over the assignments of values to an object. It is also emphasized that the "probabilistic model must take into account the influences of the environment on the object". [KollPfeff]

In regard to the role of the class structure typical of OOP, D. Koller and A. Pfeffer observe, "Classes are used to provide a reusable probabilistic model which can be applied to multiple similar objects. Classes also support inheritance of model fragments from a class to a subclass, allowing the common aspects of related classes to be defined only once". Noting that "complex models often involve many similar objects (or attributes of objects), whose stochastic functions are essentially identical" D. Koller and A. Pfeffer emphasize the necessity of defining generic object-oriented Bayesian network fragment, which "can be used multiple times in defining many similar objects" and could be associated with multiple objects. [KollPfeff]

D. Koller and A. Pfeffer put a particular accent on the main advantage OOP "to naturally represent objects that are composed of lower level objects", as well as "the ability to explicitly represent classes of objects, crucial for the incorporation of inheritance into the language", stressing that these properties are of crucial importance "for large-scale knowledge representation".

Nevertheless, D. Koller and A. Pfeffer also address particular potential weaknesses of the approach. Thus, for instance, it is pointed out that although object-oriented Bayesian networks allow "to utilize the same class hierarchy to define models of a variety of different structures, once a model is described in the language, its structure is fixed. In particular, the language does not allow us to express uncertainty about the identity and number of objects in the model and about the relationships between them". The inability to "express global constraints on a set of objects" is also mentioned as a related restriction. The lack of "the expressive power to deal suitably with situations that evolve over time" is considered by D. Koller and A. Pfeffer to be another major restriction. For example, they describe objects as "static", as "once an object is defined, its properties are determined once and for all (although we may still be uncertain about them)", whereas it would be preferable "to be able to apply OOBNs to domains involving multiple interacting entities whose state changes over time". [KollPfeff]

Still, the undertaken approach permits "a knowledge engineer to organize a model in a natural and coherent manner", combining three types of knowledge: "relevance relationships and conditional probabilities" and "organizational structure".[KollPfeff]

To sum up, object-oriented paradigm places at the disposal of probabilistic programming suitable means of creating and appropriate structuring models for complex domains of human knowledge, as well as means of reasoning over these models. Representation of domain information in terms of classes, objects and attributes is an essential prerequisite for reusable code production. However, the question about the most proper ways of representation uncertainty and situation changeability over time in the context of this approach still remains open.

BRIEF OVERVIEW OF FIGARO

Figaro is one of the most mature and promising probabilistic programming tools. It is a Scala open-source library, which provides rich functionality for probabilistic reasoning and is applicable in industrial IT projects due to its interoperability with Java.

Figaro possesses means of construction and procession of different kinds of models, which include:

- directed and undirected models,
- models in which conditions and constraints are expressed by arbitrary Scala functions,
- models involving interrelated objects,
- open universe models in which we don't know what or how many objects exist,
- models involving discrete and continuous elements,
- models in which the elements are rich data structures such as trees,
- models with structured decisions,
- models with unknown parameters.[PfeffRutHowCon]

Moreover, Figaro also allows a possibility of extending its built-in means and creating customary model elements and new data structures.

A number of reasoning algorithms are available in Figaro, as for example:

- exact inference using variable elimination,
- belief propagation,
- lazy factored inference for infinite models,
- importance sampling,
- Metropolis-Hastings,
- most probable explanation (MPE),
- particle filtering,
- Gibbs sampling,
- parameter learning using expectation maximization.[PfeffRutHowCon]

New reasoning algorithms can also be created in Figaro in addition to the already available built-in algorithms.

Figaro operates with elements, which are instances of an Element class, with a parameter for a value type. An atomic element is the simplest element, which is defined as "one that does not depend on any other elements".[Pfeff2] For example:

- Flip(0.8) is an Element[Boolean], expressing the probabilistic model that produces true with probability 0.8 and false with probability 0.2.
- Select(0.3 -> 0, 0.4 -> 1, 0.6 -> 2) is an Element[Int] that represents the probabilistic model that produces 0 with probability 0.3, 1 with probability 0.4, and 2 with probability 0.6. The element "Select" can operate between elements of any type.

More complex elements, which can be created from the combination of simpler ones, are called compound elements. For example, following conditional element is a compound:

If(Flip(0.6), Constant("a"), Select(0.3 -> "b", 0.7 -> "c")),

where Constant("a") is chosen with the probability 0.6, Select(0.3 -> "b", 0.7 -> "c") is chosen with the probability of 0.4. In other words, value "a" is produced with probability 0.6 * 1 = 0.6, value "b" has the probability of 0.4 * 0.3 = 0.12, while "c" equals 0.4 * 0.7 = 0.28.

Another important feature of Figaro is the ability to work with continuous elements (library.atomic.continuous package), such as Normal, Exponential, Gamma, Beta, and Dirichlet. "Continuous" is defined as meaning that "the values lie in a continuum with no separation, such as the real numbers".[Pfeff2]

One of the central concepts in Figaro is the concept of Universe, which is a special type of an element collection with services for inference algorithms, such as memory management and dependency analysis. As a result, inference algorithms usually operate on a universe. If the universe is not specified explicitly, an element is placed in a default universe. [Pfeff2]

Explicit processing of Universe elements is mostly needed "to create dynamic probabilistic programs that describe a domain that changes over time". For this case, the model is specified in two steps:

- initial universe definition,
- a function from a universe representing the distribution at one time point to a universe representing the distribution at the next time point. [PfeffRutHowCon]

All in all, Figaro is one of the most viable probabilistic tools nowadays, because it provides an extensive flexibility defining data models and running inference algorithms on them. Figaro's compatibility with Java environment allows to integrate its functionality in a wide range of IT projects.

CONCLUSION

Probabilistic programming is one of the most rapidly growing areas of IT research nowadays arousing interest in academic circles (including research groups in MIT and Oxford), acknowledged IT leaders as Microsoft, well-established industrial customers and IT community all over the world.

The increasing interest of the international IT community to this relatively new direction can be accounted by practical applicability of probabilistic programming concepts in the context of machine learning.

Probabilistic programming explores possibilities of mapping theoretical concepts of probability theory onto suitable practical programming techniques to reason under uncertainty.

Probabilistic programmes operate with variables holding the quantified knowledge about the constituent elements of the modelled situation. There exist two general types of dependencies representing relations of variables in probabilistic programmes: directed and undirected. Bayesian networks are, as a rule, used to express directed dependencies, whereas Markov networks represent undirected dependencies.

An active interest of the academic community to the probabilistic programming encouraged appearance of various tools designed to perform tasks of probabilistic inference. These tools include both frameworks of already existent general purpose programming languages and "purely" probabilistic programming systems (many of them not Turing complete). A large number of current probabilistic programming tools are implemented on the basis of functional programming paradigm. However, OOP paradigm is considered to be promising in the context of probabilistic programming, as it suggests natural mechanisms of modelling the reality in terms of objects and enables reuse of code.

With respect to inference algorithms, there can be differentiated two major groups, i.e. exact (e.g. Variable Elimination algorithm) and approximate inference algorithms (e.g. sampling family). Inference complexity makes it especially important to choose the right algorithm for each particular situation.

Although probabilistic programming has managed to arouse the interest of the international IT community and to achieve positive results in a number of research projects worldwide, there are still things to be done for probabilistic programming to prove itself as a generally accepted standard. In particular, it's needed to work out a unified basis for different approaches within probabilistic programming and develop "best practices" of it. Second, probabilistic programming has to be explored and tested in large-scale industrial IT projects, outside purely academic environment.

LITERATURE

Cronin B. "What is probabilistic programming?" Retrieved 20.01.2017 from http://radar.oreilly.com/2013/04/probabilistic-programming.html

CS181 Lectures 20 - 21 - "Bayesian Networks". Retrieved 20.01.2017 from http://isites.harvard.edu

Goodman N. D. "The Principles and Practice of Probabilistic Programming". In Principles of Programming languages (POPL), 2013

Goodman N. D., Tenenbaum J. B. (electronic). "Probabilistic Models of Cognition". Retrieved 20.01.2017 from http://probmods.org.

Gordon A. D., Henzinger T. A., Nori A. V. "Probabilistic Programming". Proc. of the International Conference on Software Engineering (ICSE, FOSE track), 2014

Hicks M. "What is probabilistic programming?" Retrieved 20.01.2017 from http://www.pl-enthusiast.net/2014/09/08/probabilistic-programming

Karkera K. R. "Building Probabilistic Graphical Models with Python". Packt Publishing, 2014.

Koller D., Friedman N. "Probabilistic Graphical Models. Principles and Techniques". MIT Press, 2009

Pfeffer A., Ruttenberg B., Howard M., O'Connor A. "Figaro Tutorial". Charles River Analytics. Retrieved 20.01.2017 from https://www.cra.com/sites/default/files/pdf/Figaro_Tutorial.pdf

Pfeffer A. "Figaro: An object-oriented probabilistic programming language". Charles River Analytics Technical Report, 2009

Pfeffer A. "Practical Probabilistic Programming". MEAP Edition, Manning Publications, 2015

Pfeffer A. "What Probabilistic Programming is and How to Use it". Manning Publications. Free Content. 2014

Poole D. "Probabilistic Programming Languages: Independent Choices and Deterministic Systems". In R. Dechter, H. Geffner, and J. Y. Halpern, editors, Heuristics, Probability and Causality: A Tribute to Judea Pearl. College Publications, 2010

Russell S. J., Norvig P. "Artificial Intelligence. A Modern Approach". Third Edition. Prentice Hall, 2010

Sucar L. E. "Probabilistic Graphical Models. Principles and Applications". Springer, 2015

Wiki. "Probabilistic Programming". Retrieved 20.01.2017 from http://probabilistic-programming.org/wiki/Home

Wingate D., Stuhlmüller A., Goodman N. D. "Lightweight Implementations of Probabilistic Programming Languages Via Transformational Compilation". Proc. of the 14th Artificial Intelligence and Statistics, 2011

Wood F., van de Meent J. W., Mansinghka V. "A New Approach to Probabilistic Programming Inference". AISTATS, 2014

CONTACT

Olga Ivanova, software developer,
ollyenn@gmail.com