Capita Selecta AI (Probabilistic Programming) 2017-2018

Inference for SRL

The goal of the Probabilistic Programming assignment is to build your own state-of-the-art inference and learning engine for probabilistic graphical models such as Bayesian networks and statistical relational models.

- You can work alone or in teams of two students. Do not share results between teams.
- Send a yourlastnames.zip file containing (1) a report describing your approach, design choices and results in a pdf and (2) executable code with a readme file to wannes.meert@cs.kuleuven.be by March 30, 2018.
- Grades for this part are fully determined by the submission (no presentation).
- Questions can be asked in the Toledo discussion forum.

1 Probabilistic Inference Using Weighted Model Counting (4/10)

One of the most performant techniques to compute the marginal or conditional probability of a query given a probabilistic graphical model (PGM) is to reduce the problem to weighted model counting (WMC). This entails that a PGM such as a Bayesian network is represented as a propositional knowledge base in conjunctive normal form (CNF) with weights associated to the propositional variables.

1.1 PGM to CNF (1/4)

Read about this approach in Chavira and Darwiche [1] (sections 1-3, optionally also 4) and familiarize yourself with ENC1 (the encoding of Chavira and Darwiche) and ENC2 (the encoding of Sang, Beam and Kautz).

In this task you are asked to encode the Cancer Bayesian network on `http://www.bnlearn.com/bnrepository/discrete-small.html#cancer` as a CNF. You can do this manually or write an automatic convertor.

Task 1.1.1 Write the encoding for the Bayesian network as a CNF and associated weights using ENC1[1].

Task 1.1.2 Write the encoding for the Bayesian network as a CNF and associated weights using ENC2.

1.2 SRL to CNF (1/4)

ProbLog, a Statistical Relational Learning formalism, is a generalization of PGM that allows one to express complex relations. Similar to PGMs, probabilistic inference for ProbLog can be reduced to a Weighted Model Counting task. Read about this approach in Fierens et al. [3] (sections 1-6.1, you can ignore cyclic rules). The concept of annotated disjuctions (or multivalued) variables is not explained in that paper but you can use the same ideas as used in ENC1 or ENC2 for multivalued variables.

In this task you will encode the (in)famous Monty Hall question on `https://dtai.cs.kuleuven.be/problog/tutorial/various/01_montyhall.html` as a CNF. You can use the ProbLog program to get a grounding (`problog ground -h`). The conversion can be performed manually or by writing a script.

Task 1.2.1 Write the encoding for the ProbLog program as CNF and associated weights.

1.3 Weighted Model Counting (1/4)

WMC can be performed by applying a search algorithm on the CNF or by compiling the CNF into a structure on which WMC can be performed in polynomial time with respect to the size of the structure. An advantage of the standardization to CNF is that multiple model counters can be applied, each with their own advantages and disadvantages. An exhaustive overview of model counters is avialable on `http://beyondnp.org/pages/solvers/model-counters-exact/`.

[1] In the report you can use logic formulae, in the zip-file you should include the CNF in dimacs format that you use for Task 1.3.1.

Task 1.3.1 Select at least two exact weighted model counters and apply them to the three CNFs of the previous tasks. Compute and report the WMC. Can you interpret them as probabilities?

Task 1.3.2 Explain briefly the main theoretical differences between the three weighted model counters.

Task 1.3.3 Create an overview of the computational requirements (e.g. runtime, memory).

1.4 Knowledge Compilation (1/4)

In this task we will focus on compilation of the CNF to an SDD structure. SDDs make use of the concept *vtree* to figure out the order in which variables are dealt with. The choice of this vtree has a large impact on the size of the circuit and thus the cost of performing inference. You can make use of the miniC2D package[2] [5, section 1] or the SDD package[3] [2, section 1] to translates a CNF into an SDD and perform weighted model counting.

Task 1.4.1 For each of the CNFs you build, decribe the vtree that, in your experiments, gives the most compact circuit. You can use the heuristics available in miniC2D (flags `-m/-t/...`) or SDD (flags `-m/-t/-r/...`) to generate different vtrees or build your own (flag `-v`).

Task 1.4.2 Is there a pattern in what makes a good vtree (e.g. is minfill always better than balanced or right-linear).

2 Build an Inference Engine (4/10)

In this part you are tasked with implementing your own pipeline to perform inference on a ProbLog program. The challenge is to outperform the ProbLog implementation available on the website[4]. Your pipeline should implement the following steps (you are allowed to deviate as long as you provide a motivation):

1. If the input is a Bayesian network, translate it to a ProbLog program. You can use the conversion scripts that are available as part of the ProbLog distribution or write an encoder to CNF yourself and skip step 2.
2. Ground the ProbLog program. You are allowed to use the command `problog ground` that is available as part of ProbLog.
3. Transform the ground ProbLog program to a CNF or propositional formula. You can use any encoding or variation you wish. You can assume that the ground ProbLog has only conjunctions of literals in the body (no complex parsing required). You can use helper libraries such as SymPy.
4. Feed the CNF to the miniC2D or SDD toolbox. You can use any set of options or vtree.
5. Compute the (conditional) probability of a given query.
6. Report (a) the probability, (b) total runtime and runtime of the separate parts, (c) number of variables and lines in the CNF, (d) statistics on the depth of the vtree, (e) number of edges and nodes in the circuit.

Task 2.1 Implement the pipeline in your preferred programming language (if no preference, use Python).

Task 2.2 Apply your pipeline to the examples used in previous tasks and report the results.

Task 2.3 Apply your pipeline to the example on `https://dtai.cs.kuleuven.be/problog/tutorial/tutslides/04_bayesian_learning.html` and report the results.

Task 2.4 Apply your pipeline to the alarm Bayesian network on `http://www.bnlearn.com/bnrepository/#alarm`. The public version of ProbLog uses an inefficient compilation strategy for this type of Bayesian networks, can you beat ProbLog? Try to compute the marginal probability of as many nodes in the network as possible and report timings (since there is no evidence, you can drop leaf nodes incrementally until your solution works).

3 Parameter learning (2/10)

One of the key tasks in machine learning is learning the parameters that fit a given dataset. In this task you implement your own Expectation-Maximisation learning algorithm based on Gutmann, Thon, and De Raedt

[2] `http://reasoning.cs.ucla.edu/minic2d/`
[3] `http://reasoning.cs.ucla.edu/sdd/`
[4] `https://dtai.cs.kuleuven.be/problog`

[4]. As this is a rather technical paper it is not required to read this paper and the relevant formula (Eq. 3) is repeated here in a simpler form:

$$\widehat{p_n} = \frac{1}{M} \sum_{m=1}^{M} P(f_n \mid I_m) \tag{1}$$

where M is the number of interpretations or examples and I_m is the m-th interpretation[5]. The parameter p_n is the probability in $p_n :: f_n$. Repeat applying this formula to all p_n until the values converge or a maximum number of iterations is reached.

Task 3.1 Generate 4 interpretations or examples for the Cancer example in Task 1.1 using the command `problog sample -h` and drop each positive and negative observation with probability 30% to create a partially observed dataset. Show them in your report.

Task 3.2 Generate 10, 100 and 1000 interpretations for the Cancer example and use your pipeline together with the above formula to estimate parameters p_n. Show the learned parameters.

Task 3.3 Explain your observations for different number of interpretations.

Task 3.4 Much more efficient solutions than what is proposed here exist. Can you come up with strategies to speed up learning?

References

[1] Mark Chavira and Adnan Darwiche. "On probabilistic inference by weighted model counting". In: *Artificial Intelligence* 172.6 (2008), pp. 772–799.

[2] Arthur Choi, Doga Kisa, and Adnan Darwiche. "Compiling Probabilistic Graphical Models using Sentential Decision Diagrams". In: *Proceedings of the 12th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU)*. 2013.

[3] Daan Fierens, Guy Van den Broeck, Ingo Thon, Bernd Gutmann, and Luc De Raedt. "Inference in probabilistic logic programs using weighted CNFs". In: *Theory and Practice of Logic Programming* 15 (2015).

[4] Bernd Gutmann, Ingo Thon, and Luc De Raedt. "Learning the Parameters of Probabilistic Logic Programs from Interpretations". In: *European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)* (2011), pp. 581–596.

[5] Umut Oztok and Adnan Darwiche. "A top-down compiler for sentential decision diagrams". In: *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI)*. 2015.

[5] For examples, see `https://dtai.cs.kuleuven.be/problog/tutorial/learning/01_bayes.html`