FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Exploring Visual Programming Concepts for Probabilistic Programming Languages

Gabriel Cardoso Candal

WORKING VERSION

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Hugo Sereno Ferreira

January 19, 2016

Exploring Visual Programming Concepts for Probabilistic Programming Languages

Gabriel Cardoso Candal

Mestrado Integrado em Engenharia Informática e Computação

January 19, 2016

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

Abbreviations

ADT	Abstract Data Type
PP	Probabilistic Programming
PPL	Probabilistic Programming Language
PPAML	Probabilistic Programming for Advancing Machine Learning
ML	Machine Learning
VP	Visual Programming
VPL	Visual Programming Language
DARPA	Defense Advanced Research Projects Agency
PR	Probabilistic Reasoning

Chapter 1

Introduction

This first chapter aims to provide the reader with an overview of this dissertation. It starts by introducing the context this work is inserted in, identifying the problem which we aim to solve, how we plan on solving it, and the expected outcome. Lastly, it gives a bird's eye view of this report's structure.

1.1 Context

There is, among several domains with interesting and relevant problems to solve (computer vision [KT15], cryptography, biology, fraud detection, recommender systems [Alp10], . . .), the recurring necessity to be able to make decisions in the face of uncertainty using machine learning (ML) methods.

Successful ML applications include Google's personalized advertising and context-driven information retrieval, Facebook's studies of how information spreads across a network or UC Berkeley's AMPLab contributions towards Amazon Web Service and SAP's products [BAF+15].

Typically, there are two approaches for this class of problems): either use an existing machine learning model (such as KNN, neural networks or similar) [Sch08] and try to fit your data into the model, or build a probabilistic model for your particular problem so you can leverage domain knowledge [Gri13].

In the second approach one common way to tackle it is to use bayesian reasoning, where you model unknown causes with random variables, feed the model the data you have gathered and then perform inference to reverse the story and query for the desired variables [Dow12]. The tricky issue is this last step, since it is non-trivial to write an inference method [DL].

The solution to this has been building generic inference engines for graphical models, so that modeling and inference can be treated as separate concerns and people can focus on the modeling [JW96]. However, not all models can be represented as graphical models, and that's why we now

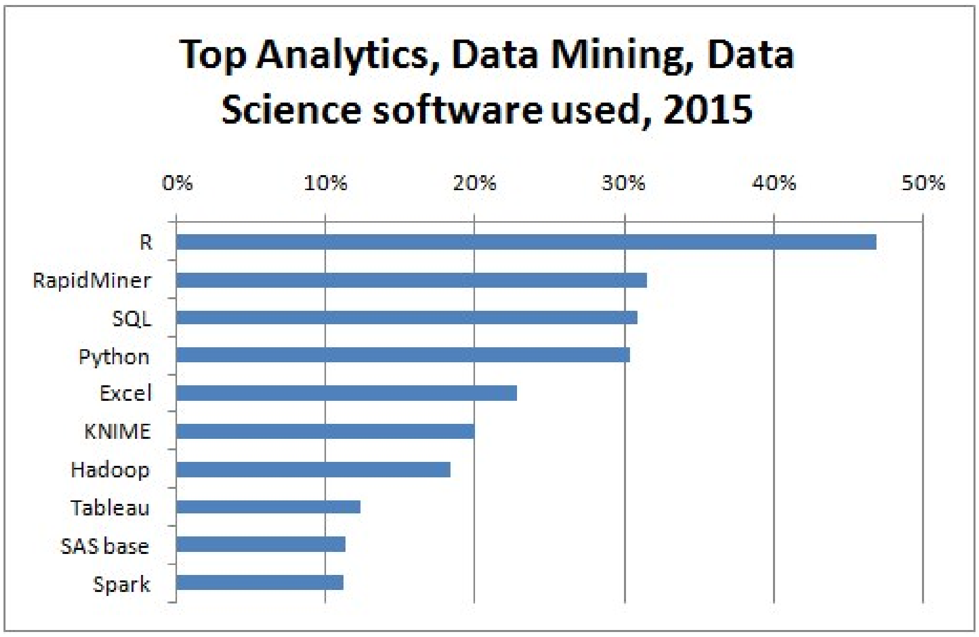


Figure 1.1: Top 10 Analytics Tools [Pia15]

have Probabilistic Programming Languages (PPLs). Probabilistic Programs let you write your model as a program and have off-the-shelf inference [Pre03].

1.2 Problem

In spite of these examples of applications in the industry, ML has been identified by Gartner, in its Hype Cycle annual review, to be in the "Peak of Inflated Expectations" stage, still far from the "Plateau of Producitvity" [Sta15].

It has also been said that ML's applications are rarely seen outside the academia, with Wagstaff claiming that there is a "frequent lack of connection between machine learning research and the larger world of scientific inquiry and humanity" [Wag12].

Arguably the scenario is even worse for PPLs, having even lesser adoption among tech companies. One factor which may be contributing to this lack of usage, despite PP's power and flexibility, is the difficulty for data scientists to adapt to textual interface these languages provide, which lack the graphical intuiton provided by other tools they are accustomed to. In a poll made to about 2800 data scientists (see Figure 1.1), half of the top 10 tools are graphically-interactable.

1.3 Motivation and Goals

The Defense Advanced Research Projects Agency (DARPA), one of the funders behind PPLs' research, has recognized some of the problems identified by Wagstaff and started a program called

Probabilistic Programming for Advancing Machine Learning (PPAML) to address the shortcomings of current ML methods [Jag13]. It identifies five strategic goals:

- Shorten machine learning model code to make models faster to write and easier to understand
- Reduce development time and cost to encourage experimentation
- Facilitate the construction of more sophisticated models that incorporate rich domain knowledge and separate queries from underlying code
- Reduce the level of expertise necessary to build machine learning applications
- Support the construction of integrated models across a wide variety of domains and tool types

The purpose of this work is to try addressing the first four. In order to do so we aim to overcome the difficulties in learning a new language, either for unexperienced developers or seasoned ones, such as learning yet another syntax or getting accostumed to the language's idioms. It is known that typical languages are difficult to learn and use [LO87] and that there are advantages in providing a language with a visual interface [AA92]. Also, studies have shown that programmers and data scientists alike resort to mental imagery when solving problems [Das02][PB99], so by providing such an interface we can approximate how people think and how they use the language to solve the problem at hand.

So, the goal of this dissertation will be to develop a Visual Programming Language (VPL) with probabilitics programming capabilities. The targeted audience are programmers and data scientists with background knowledge in statistics who aren't still comfortable with full blown PPLs, but wish to educate themselves in the topic so they can eventually leverage the power of this novel machine learning approach.

The way to do so would be developing a graphical node-based editor, similar to RapidMiner or Blender Composite Nodes, but that runs in the browser. The given editor would have the capability to compile its graph to the textual representation in the target PPL so that the user can run what it has designed, either as a standalone script or even by integrating it with his existing projects.

The hypothesis under consideration is this graphical representation is more intuitive and easy to learn that a full-blown PPL. We intend to validate such hypothesis by ensuring that classical problems solved in the literature by PPLs are also supported by our graphical representation, and then measure how quickly a group of people trained in statistics would produce a viable model in both alternatives.

1.4 Outline

Para além da introdução, esta dissertação contém mais x capítulos. No capítulo 2, é descrito o estado da arte e são apresentados trabalhos relacionados. No capítulo 3, ipsum dolor sit amet, consectetuer adipiscing elit. No capítulo 4 praesent sit amet sem. No capítulo 5 posuere, ante non tristique consectetuer, dui elit scelerisque augue, eu vehicula nibh nisi ac est.

Chapter 2

Background & State of the Art

This chapter has two purposes: describing the foundations on which this work is built on, namely Machine Learning (ML), Probabilistic Reasoning (PR), Prograbilistic Programming (PP) and Visual Programming(VP) while enumerating different tools which are based on one of these concepts.

2.1 Machine learning

Machine learning is a field which can be seen as a subfield of artifical intelligence that incorporates mathematics and statistics and is concerned with conceiving algorithms that learn autonomously, that is, without human intervention [Bri16][Sch08]. It has the potential to impact a wide spectrum of different areas such as biology, medicine, finance, astronomy [Ama13], computer vision, sales forecast, robotics [Alp10], product recommendations, fraud detection or internet ads bidding [Gri13].

Learning from data is commercially and scientifically important. ML consists of methods that automatically extract interesting knowledge in databases of sometimes chaotic and redundant information. ML is a data-based knowledge-discovering process that has the potential not only to analyze events in retrospect but also to predict future events or important alterations [Geo16].

2.2 Probabilistic Reasoning

Probabilistic reasoning is the formation of probability judgments and of subjective beliefs about the likelihoods of outcomes and the frequencies of events [?], it is a way to combine our knowledge of a situation with the laws of probability. There are subjective belifs because, in non-trivial decision-making there are unobserved factors that are critical to the decision in conjunction with several sources of uncertainty [GSD], such as:

- Uncertain inputs, due to missing or noisy data

- Uncertain knowledge, where multiple causes lead to multiple effects, or there is an incomplete knowledge of conditions, effects and causality of the domain or simply because the effects are inherently stochastic.

So, probabilistic reasoning only gives probabilistic results.

It is one way to overcome cognitive bias and be able to make rational decisions [SG01]. A trial has been made [CSB82] where physicians were asked to estimate the probability that a woman with a positive mammogram actually has breast cancer, given a base rate of 1% for breast cancer, a hit rate of about 80%, and a false-alarm rate of about 10%. It reported that 95 of 100 physicians estimated the probability that she actually has breast cancer to be between 70% and 80%, whereas Bayes's rule gives a value of about 7.5%. Such systematic deviations from Bayesian reasoning have been called "cognitive illusions,". We will describe both Bayes's rules and Bayesian reasoning in the next section.

2.2.1 Bayesian Reasoning

One way to approach PR is by using bayesian reasoning, which is inspired in the Bayes Theorem (or Rule, or Law). An equivalent formula to the theorem, in its simplest form (applied to a single event) is:

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

Where P(A|B) defines the probability of event A given that B occurred. The theorem defines how hidden causes (A) relate to observed events (B), given a causality model (P(A, B) or P(B|A)*P(A)) and our knowledge of the probability of the occurrence of events (P(B)). The inverse is also true, as we will see further ahead in this section. As an example, P(penalty | goal) defines the probability that a penalty kick was scored, knowing that there was a goal.

There are at least two interpretations to the theorem and regarding how one may think about its results [Fie06]:

- Frequentist interpreation: probabilities are defined by the relative frequency of events, given a natural sampling. Meaning, the probabiltiy of obtaining 'Heads' when rolling a dice is equal to the number of 'Heads' obtained after rolling the dice a sufficient number of times relative to the total number of times the dice has been rolled.

- Epistemological interpreation: probabilities represent a measure of belief. It can either be a result logical combination of probabilities through the usage of axioms (it's closely related to Aristotlean logic) or it can also reflect a personal belief (which is called a subjective view).

Table 2.1: Alarm system confusion matrix

	alarm	$\neg alarm$
burglary	0.09	0.01
$\neg burglary$	0.1	0.8

2.2.1.1 An example

One example of the application of this theorem is [GSD]: you know your home's alarm is ringing, but you don't know whether that was caused by a burglar or something else (maybe a bird triggered it, or there was a malfunction in the alarm system). How confident are you that you're being robbed? Consider that the alarm company, based on quality trials, defined in the confusion matrix for P(alarm, burglary) (Table 2.1).

You can interpret each table's cell as P(A, B). For instances, the top left cell is the probability that the alarm rings and there is a burglar, while the bottom left cell is the probability that the alarm rang but there was no burglar (a false positive).

If we substitute the values of Bayes' rule described above, we get:

$$P(burglar \mid alarm) = \frac{P(burglar, alarm)}{P(alarm)}$$

Where results is 0.09 / 0.19 = 0.47. So, even if the alarm is ringing, there is just a 47% probability that the house is actually being robbed.

The previous example illustrates the simplest case of applied BR, but it is also possible to combine several variables. One way to represent this kind of scenario is by expressing the variables in a directed acyclic graph, where the relation "Parent" stands for "May cause" and you can specify the conditional probabilities of a child given a parent's result. This graphical model is called a Bayesian Network.

2.2.1.2 Bayesian Networks

We can extended our alarm example further, by considering not only a burglar can trigger the alarm, but an earthquake also can (while there can still be false positives). Also, consider that we have 2 neighbors (Mary and John) who may call us whether the alarm is ringing or not. This problem is represented in figure **??**.

Some interesting question we can ask, given this scenario are:

If John calls saying the alarm is ringing but Mary doesn't, what are the odds it really is ringing?

If the alarm is ringing, was there an earthquake?

What are the chances that both my neighbors call, the alarm is ringing, but there is neither a burglary nor an earthquake?

This last example, for instances, would be calculated as: P(J, M, A, $\neg B, \neg E) = P(J|A) * P(M|A) * P(A|\neg B, \neg E) * P(\neg B) * P(\neg E) = 0.9 * 0.7 * 0.001 * 0.999 * 0.998 = 0.00062$.

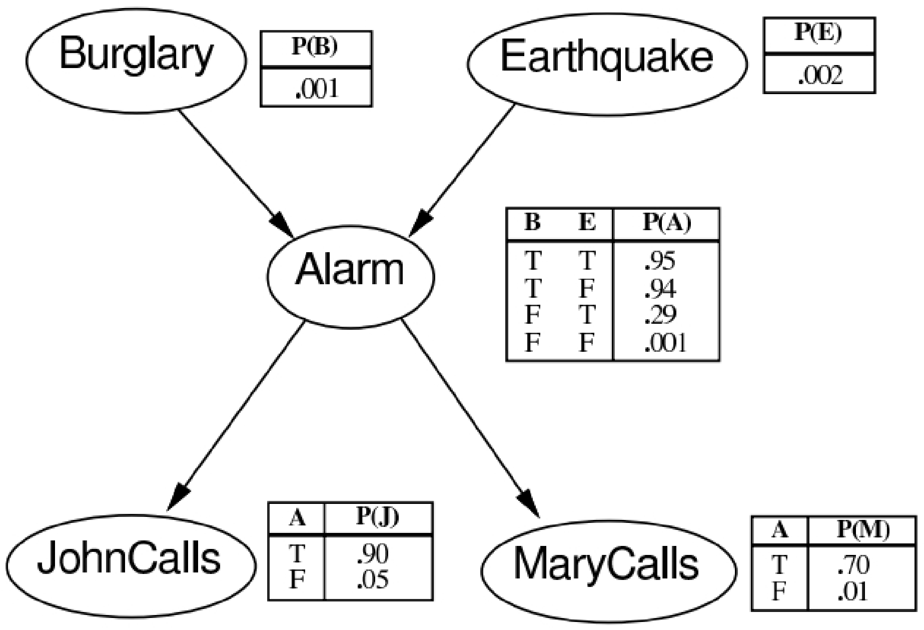


Figure 2.1: Belief network for the alarm problem [Wan02]

Notice how counter-intuitive this example is: the probability of there being an earthquake is about 32 times larger then there being an earthquake, the alarm ringing and the neighbors calling us, even if the conditional probabilities are reasonably high (0.95, 0.9 and 0.7). This is the result of the calculation of the joint probability being an highly combinatorial problem, which is yet another argument in favor of using PR rather than subjective heuristics.

2.2.1.3 Bayes' and data streams

In practical ML applications, it is often the case that there is an incoming stream of new data, rather than one-time batch calculations. BR can accomodate this way of thinking, which A. Downey called diachronic interpretation [Dow12], where diachronic means that something is happening over time (in this case the probability of the hypotheses, as new data arrives). In order to make sense of this definition, we may rewrite Bayes' rule as:

$$P(H \mid D) = \frac{P(D \mid H)P(H)}{P(D)}$$

Where:

- H: hypothesis
- D: data
- p(H): probability of the hypothesis before the new data is taken into account. Also called **prior**. It can either be calculated using background information or subjectively defined using

domain knowledge. Loses significance as new data is added, so its choice is not determinant to the model's performance in the long run.

- p(H|D): what we want to calculate, the probability of the hypothesis after considering the new date. It is called **posterior**.
- p(D|H): probability of the data if the hypothesis was true, called the **likelihood**.
- p(D): probability of the data under any hypothesis, called the **normalizing constant**.

Under this interpretation, you may continuously feed data into the model and see the probabilities getting updated. We will see more practical examples of this in section 2.2.2.

2.2.1.4 Beyond Bayesian Graphical Model

At first glance, someone who is learning for the first time about PR applied to ML, may think that graphical models such as the one presented in Figure 2.1 are the best there can be done in terms of using a graphical interface for solving this kind of problems and that the only thing is missing is an automated way to make the calculations.

While it is true we have never refered mentioned techniques or tools that automatically do inference over a Bayesian Network, there are several tools with that capability (including an R package [øjsgaard2013] or standalone tools [Res10]).

However, not all PR can be done via Bayesian Networks and not all graphical models are enough for complete PR [DL13]. PP are the largest class of models available, and there are also more algorithms for inference than just the calculation of joint probabilities (like we did in the alarm example), as we will discuss in Section 2.2.2.

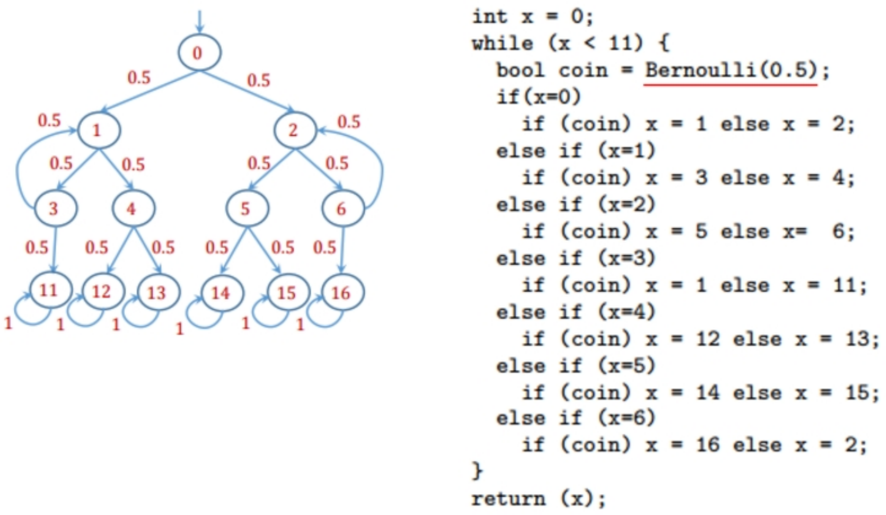
Bayesian Networks are not the only kind of graphical model. Another one would be Markov Chains, which is yet another example of a model which is not able to represent all PR problems. This is clear when we realize that, while PPLs support a large number of different distributions (such as Normal, Laplace, Gamma, Half-Cauchy or *t*), all Bayesian Networks and Markov Chain can be represented in a PPL by just using Bernoulli distributions [Gor14]. We can see an example of such a translation in Figure 2.2.

<—– rever com base em prob inference for graphical models

2.2.2 Probabilistic Programming Languages

2.2.2.1 The Probabilistic Program-Model duality

A probabilistic program (PP) is an ordinary program (that can be written in mainstream languages such as C, Java or Haskell) whose purpose is to specify a probability distribution of its variables. This is done by sampling over several executions of the program. The only needed construct the language has to support, in order to be able to write a PP, is having a random number generator [DL]. This whole concept couldn't be better explained than in this text by Freer and Roy, regarding Church (a PPL, which we describe in Section 2.4.3) but common to any PP:



```
int x = 0;
while (x < 11) {
  bool coin = Bernoulli(0.5);
  if(x=0)
    if (coin) x = 1 else x = 2;
  else if (x=1)
    if (coin) x = 3 else x = 4;
  else if (x=2)
    if (coin) x = 5 else x=  6;
  else if (x=3)
    if (coin) x = 1 else x = 11;
  else if (x=4)
    if (coin) x = 12 else x = 13;
  else if (x=5)
    if (coin) x = 14 else x = 15;
  else if (x=6)
    if (coin) x = 16 else x = 2;
}
return (x);
```

Figure 2.2: Translation of Discrete Time Markov Chain to a PPL [Gor14]

> “If we view the semantics of the underlying deterministic language as a map from programs to executions of the program, the semantics of a PPL built on it will be a map from programs to distributions over executions. When the program halts with probability one, this induces a proper distribution over return values. Indeed, any computable distribution can be represented as the distribution induced by a Church program in this way” [FR12]

One way to think about this notion is by considering that the program itself is the model. An example of the relation between a model (expressed in a PPL) and the implied distribution over its variables (obtained using an inference method) can be seen in Figure **??**, where a variable *flip* is set to be a Bernoulli distribution and *x* is defined in terms of *flip*. We can then see how the graphic of the inferred distributions of *flip's* and *x's* values looks like and confirm what was to be expected: for *flip's* values lower than 0.5 we see *x* follows a normal distributon, whereas for values greater than 0.5 it follows a gamma distribution instead. The goal of PP is to enable PR and ML to be accessible to most programmers and data scientists who have enough domain and programming knowledge but not enough expertise in probability theory or machine learning.

2.2.2.2 PPLs vs regular PLs

What is then, a Probabilistic Programming Language (PPL)? First of all, it can be a standalone language or an extension to a general purpose programming languag. We'll be analyzing examples of languages from either these categories in Section 2.4, but many more exist, such as as Figaro [RT15] (hosted in Scala), webppl [GS14] (embedded in Javascript) or Dimple [HBB$^+$12] (has both a Java and a MATLAB API). The key difference between these languages and a PPL is the latter has the added capability of performing conditioning and inference [ADDJ03].

```
flip = rand < 0.5
if flip
    x = randg + 2    % Random draw from Gamma(1,1)
else
    x = randn        % Random draw from standard Normal
end
```

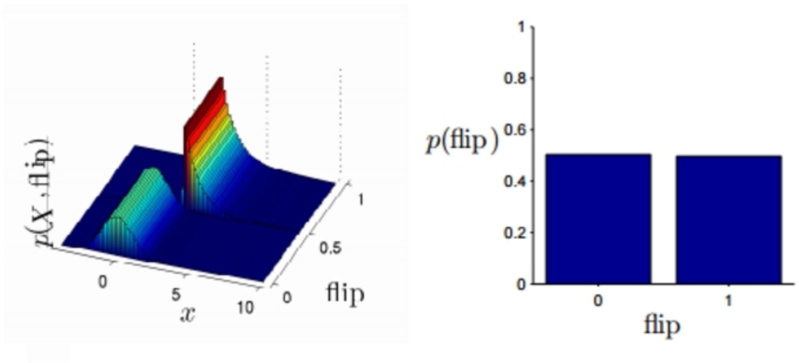


Figure 2.3: Implied distributions over variables [DL]

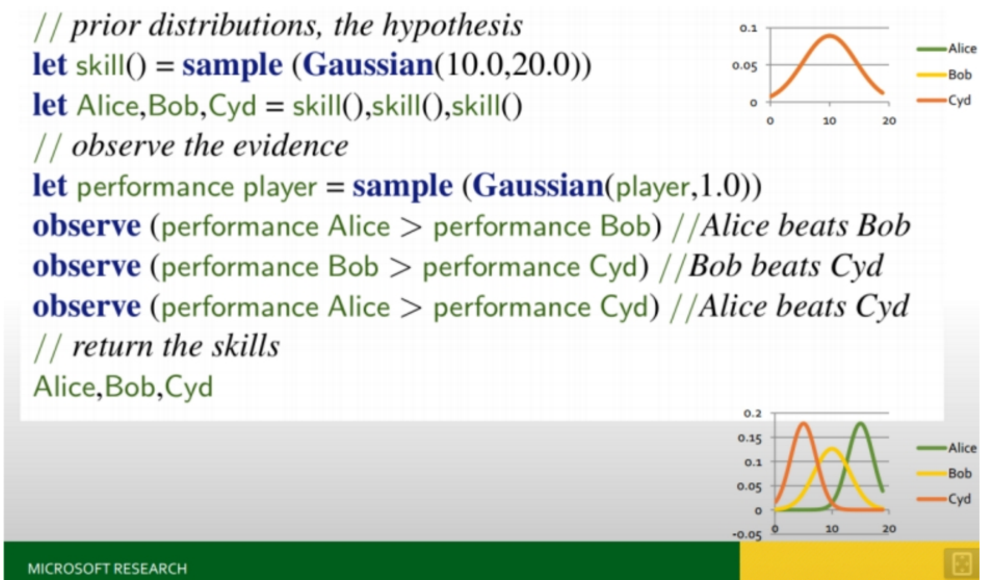
Conditioning is the ability to introduce observations about the real world in the program. That way, you update the prior probability based on those observations. Consider the example in Figure 2.4 (which is a simplified version of how Microsoft applies PP in its Xbox matchmaking algorithm [MWGK12]) where the prior is a normal distribution with equal parameters for all players (shown by the graphic in the top). Then, it defines how the performance of the player is based on his skill (which at the initial point in time, is equal to every one of them) and proceeds to make several observations regarding games between them. Finally, it shows the inferred probability distribution of the posterior on the bottom graphic.

2.2.2.3 Inference

We said that a PPL empowers the user to formalize a model and then query for the probability distribution of its variables, which is automatically done via inference. While general-purpose language require you to write one-time inference methods that tightly coupled to the PP you are inferring on, PPLs ship with an inference engine suited to most PP programs you can write [FMR10].

An inference engine of a PPL acts similarly to a compiler in a traditional language: rather than requiring each user to engineer its own, which requires significant expertise and is a non-trivial and error-prone endeavour, every PPL has one incorporated. Olivier Grisel called this separation of concerns between the language and its inference engine "openbox models, blackbox inference engine" [Gri13].

Having the engine work as a separate model opens up a myriad of new possibilities, mainly in the form of knowledge and tool sharing, as we have seen in the past in the compiler space. Examples of this would be new compiler compiler and interpreter techniques (such as working towards scalability or parallelization), optimizers, profilers or debuggers.



```
// prior distributions, the hypothesis
let skill() = sample (Gaussian(10.0,20.0))
let Alice,Bob,Cyd = skill(),skill(),skill()
// observe the evidence
let performance player = sample (Gaussian(player,1.0))
observe (performance Alice > performance Bob) //Alice beats Bob
observe (performance Bob > performance Cyd) //Bob beats Cyd
observe (performance Alice > performance Cyd) //Alice beats Cyd
// return the skills
Alice,Bob,Cyd
```

Figure 2.4: Microsoft Xbox Live True Skill [MWGK12]

Another great advantage of having a modular inference engine is that we can try different inference algorithms and pick the one that best fits the problem at hand. When analyzing if algorithm suits a certain use case, there are certain characteristics worth noting [Min99]:

- Determinism - in equal initial conditions, an algorithm always yields the same result.
- Exact result or approximation
- Guaranteed convergence - an algorithm may or not be guaranteed to reach a result at some point in time. If not, it's possible that it will run forever.
- Efficiency - related to how fast can it reach a result.

Microsoft's Infer.NET provides three inference algorithms [Res]:

- Expectation Propagation - deterministic, provides exact solutions only in some cases, is not guaranteed to converge and is labelled as "reasonably efficient".
- Variational Message Passing - also deterministic, but always gives approximates results and is guaranteed to converge. It's considered to be the most efficient of the three for most cases.
- Gibbs sampling - non-determinisc, may be able to reach exact result if given enough time to run, has guaranteed convergence and is regarded as not-so-efficient as the other two.

We can divide the three algorithms into two categories: variational bayesian methods [WBJ05][Min99] and sampling methods (in the case of Gibbs sampling, it's based on Markov chain Monte Carlo) [ADDJ03]. The main difference, as far as the end-user is concerned, is that variational methods provide faster results but are subject to bias, whereas sampling methods have the potential to produce more accurate results (the downside being it's slower and its convergence is hard to diagnose) [SAC$^+$10].

Please notice than none of these algorithms provides the same kind of exact solution as the calculation of the joint probabilities we did in Section 2.2.1. The reason for that is that calculating an exact solution takes time exponential in the number of variables to run, even if we have smarter algorithms than the naive calculations we did [ZP94].

2.2.2.4 Openbox models

When compared to traditional machine learning methods (such as random forests, neural networks or linear regression), which take homogeneous data as input (requiring the user to separate their domain into different models), probabilistic programming is used to leverage the data's original structure. This is done by empowering the user to write his own models while taking advantage of a re-usable inference engine. Olivier Grisel called this combination "Openbox models, blackbox inference engine"[Gri13].

Rather than using most of his time performing feature engineering (that is, trying to fit the problem and the data into an existing model), the user will have the tool necessary to design the model that best fits the domain he is working on. Plus, it provides full probability distributions over both the predictions and parameters of the model, whereas ML methods can mostly only give the user a certain degree of confidence on the predictions.

2.2.3 Conclusion

Summarizing, PPLs are a step forward in using PR to solve ML problems since it helps overcome the difficulties in using PR in real world problems. This is done by adding automated general inference on top of a precise specification (the program), where in the past models were communicated using a mix of natural language, pseudo code, and mathematical formulae and solved using special purpose, one-off inference methods.

This encourages exploration, since different models require less time to setup and evaluate, and enables sharing knowledge in the form of best practices and design and development patterns.

However, using a full fleged programming language might still be an entry barrier. We want to help statisticians and data scientists alike to learn faster and be more productive using a PPL in a way similar to the tools they are accostumed to. In order to do so, we'll be combining a PPL with Visual Programming (Section 2.3).

2.3 Visual Programming

2.3.1 Visual Dataflow Programming

2.4 State of the Art

2.4.1 Stan

2.4.2 WinBUGS

2.4.3 Church

2.4.4 Infer.NET

2.4.5 PyMC

2.4.6 VIBES

2.4.7 NoFlo

2.4.8 RapidMiner

2.4.9 Weka Knowledge FLow

2.4.10 GoJS

2.4.11 Blockly

2.5 Conclusions

Chapter 3

Solution prototype

todo: escolher frontend, escolher backend

3.1 The problem it solves

3.2 Outline

3.3 Architecture

3.4 Implementation

3.4.1 Picking a front-end

3.4.1.1 Blockly

3.4.1.2 GoJS

3.4.1.3 Custom implementation

3.4.2 Picking a target PPL

3.4.2.1 WebPPL

3.4.2.2 PyMC

3.4.2.3 Infer.NET

3.4.3 Defining a Grammar

3.4.4 Cycles

3.4.5 Inverse Compilation

3.4.6 Instant visual results

3.4.7 Opening to extension

3.5 Tutorial

3.6 Conclusions

Chapter 4

Evaluation

4.1 Problems solved

4.2 Problems detected

4.3 Conclusions

Chapter 5

Conclusions

5.1 Contributions

5.2 Future Work

References

[AA92] K S R Anjaneyulu and John R Anderson. The Advantages of Data Flow Diagrams for Beginning Programming. 1992.

[ADDJ03] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I. Jordan. An introduction to MCMC for machine learning. *Machine Learning*, 50(1-2):5–43, 2003.

[Alp10] Ethem Alpaydin. *Introduction to Machine Learning*. MIT Press, second edition, 2010.

[Ama13] Xavier Amatriain. Beyond data: from user information to business value through personalized recommendations and consumer science. In *Proceedings of the 22nd ACM international conference on Conference on information &#38; knowledge management*, CIKM '13, pages 2199–2200, New York, NY, USA, 2013. ACM.

[BAF$^+$15] Andrei Broder, Lada Adamic, Michael Franklin, Maarten de Rijke, Eric Xing, and Kai Yu. Big data: New paradigm or "sound and fury, signifying nothing"? In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, WSDM '15, pages 5–6, New York, NY, USA, 2015. ACM.

[Bri16] Encyclopædia Britannica. machine learning. Available at http://www.britannica.com/technology/machine-learning, accessed in 3rd Jan 2016, 2016.

[CSB82] Jay JJ Christensen-Szalanski and Lee Roy Beach. Experience and the base-rate fallacy. *Organizational Behavior and Human Performance*, 29(2):270–278, 1982.

[Das02] M. Dastani. The role of visual perception in data visualization. *Journal of Visual Languages and Computing*, 13(6):601–622, 2002.

[DL] David Duvenaud and James Lloyd. Introduction to probabilistic programming.

[DL13] David Duvenaud and James Lloyd. Talk on 'introduction to probabilistic programming', 2013.

[Dow12] Allen B. Downey. *Think Bayes*. Green Tea Press, 2012.

[Fie06] Stephen E. Fienberg. When did Bayesian inference become "Bayesian&#034;? *Bayesian Analysis*, (1):1–41, 2006.

[FMR10] Cameron E Freer, Vikash K Mansinghka, and Daniel M Roy. When are probabilistic programs probably computationally tractable? *NIPS 2010 Workshop on Monte Carlo Methods in Modern Applications*, pages 1–9, 2010.

[FR12] Cameron E. Freer and Daniel M. Roy. Computable de Finetti measures. *Annals of Pure and Applied Logic*, 163(5):530–546, 2012.

[Geo16] Pétia Georgieva. Machine learning for big data processing, 2016.

[Gor14] Sriram K. Gordon, Andrew D. and Henzinger, Thomas A. and Nori, Aditya V. and Rajamani. Probabilistic Programming. *International Conference on Software Engineering (ICSE Future of Software Engineering)*, pages 267–, 2014.

[Gri13] Olivier Grisel. Keynote on 'trends in machine learning and the scipy community'. Available at https://youtu.be/S6IbD86Dbvc, accessed in 3rd Jan 2016, 2013.

[GS14] Noah D Goodman and Andreas Stuhlmüller. The Design and Implementation of Probabilistic Programming Languages. `http://dippl.org`, 2014. Accessed: 2016-1-14.

[GSD] Andreas Geyer-Schulz and Chuck Dyer. Lecture notes on stanford's 'cs63: Knowledge representation and reasoning - chapter 10.1-10.2, 10.6'.

[HBB$^+$12] Shawn Hershey, Jeffrey Bernstein, Bill Bradley, Andrew Schweitzer, Noah Stein, Theophane Weber, and Benjamin Vigoda. Accelerating inference: towards a full language, compiler and hardware stack. *CoRR*, abs/1212.2991, 2012.

[Jag13] Suresh Jagannathan. Probabilistic programming for advancing machine learning (ppaml). Available at http://www.darpa.mil/program/probabilistic-programming-for-advancing-machine-Learning, accessed in 3rd Jan 2016, 2013.

[JW96] Michael I Jordan and Yair Weiss. Probabilistic inference in graphical models. *Lauritzen, S. L*, 16(510):140–152, 1996.

[KT15] Tejas D Kulkarni and Joshua B Tenenbaum. Picture : A Probabilistic Programming Language for Scene Perception. *Cvpr*, 2015.

[LO87] Clayton Lewis and Garay Olson. Can principles of cognition lower the barriers to programming? *Empiral Studies of Programmers:Second Workshop*, 17:248–263, 1987.

[Min99] Thomas P Minka. Expectation Propagation for Approximate Bayesian Inference F d. *Statistics*, 17(2):362–369, 1999.

[MWGK12] Tom Minka, John Winn, John Guiver, and David Knowles. Infer .net 2.5. *Microsoft Research Cambridge*, 2012.

[PB99] Marian Petre and Alan F Blackwell. Mental imagery in program design and visual programming. *Int. J. Hum.-Comput. Stud.*, 51(1):7–30, 1999.

[Pia15] Gregory Piatetsky. 16th annual kdnuggets software poll. Available at http://www.kdnuggets.com/2015/05/poll-r-rapidminer-python-big-data-spark.html, accessed in 4th Jan 2016, 2015.

[Pre03] Probabilistic Programming. *Handbooks in Operations Research and Management Science*, 10(C):267–351, 2003.

[Res] Microsoft Research. http://research.microsoft.com/en-us/um/cambridge/projects/infernet/docs/Working http://research.microsoft.com/en-us/um/cambridge/projects/infernet/docs/Working%20with%20different%20inference%20algorithms.aspx. Accessed: 2016-01-14.

[Res10] Microsoft Research. Microsoft bayesian network editor. Available at http://research.microsoft.com/en-us/um/redmond/groups/adapt/msbnx/, accessed in 10th Jan 2016, 2010.

[RT15] Michael Reposa and Glenn Takata. Figaro. Available at https://github.com/p2t2/figaro, accessed in 14th Jan 2016, 2015.

[SAC$^+$10] Yuan Shen, Cedric Archambeau, Dan Cornford, Manfred Opper, John Shawe-taylor, and Remi Barillec. A comparison of variational and Markov Chain Monte Carlo methods for inference in partially observed stochastic dynamic systems. *Journal of Signal Processing Systems*, 61(1):51–59, 2010.

[Sch08] Rob Schapire. Lecture notes in 'cos 511: Theoretical machine learning'. Available at http://www.cs.princeton.edu/courses/archive/spr08/cos511/scribe$_{n}otes/0204.pdf,accessedin3rdJa$

[SG01] P Sedlmeier and G Gigerenzer. Teaching Bayesian reasoning in less than two hours. *Journal of experimental psychology. General*, 130(3):380–400, 2001.

[Sta15] Conn Stamford. Gartner's 2015 hype cycle for emerging technologies identifies the computing innovations that organizations should monitor. Available at http://www.gartner.com/newsroom/id/3114217, accessed in 3rd Jan 2016, 2015.

[Wag12] Kiri Wagstaff. Machine Learning that Matters. *Proceedings of the 29th International Conference on Machine Learning*, pages 529–536, 2012.

[Wan02] DeLiang Wang. Ohio state cis 730, lecture notes on 'probabilistic reasoning - belief networks', 2002.

[WBJ05] John Winn, Cm Bishop, and T Jaakkola. Variational Message Passing. *Journal of Machine Learning Research*, 6:661–694, 2005.

[ZP94] Nevin L Zhang and D Poole. A simple approach to Bayesian network computations. *Proc. of the Tenth Canadian Conference on Artificial Intelligence*, pages 171–178, 1994.