EXPLORANDO CONCEITOS DE PROGRAMAÇÃO VISUAL PARA PROGRAMAÇÃO PROBABILÍSTICA

Gabriel Cardoso Candal

Dissertação realizada sob a orientação de *Prof. Hugo Sereno Ferreira*

1. Motivação

Existe, entre vários domínios com problemas relevantes e interessantes para resolver (visão por computador, criptografia, biologia, deteção de fraude, sistemas de recomendação, ...) [1], a necessidade recorrente de tomar decisões em ambientes de incerteza usando métodos de *machine learning*.

Um dos métodos de ML que pode ser usado para abordar este problema é construir um modelo probabilístico através uso de uma linguage de programação probabilística, que permite definir um modelo como um programa e oferece inferência pronta a usar [2]. Apesar do poder e flexibilidade destas linguagens, é difícil para a sua audiência (*data scientists*, estátisticos, matemáticos, ...) adaptarem-se à representação textual que estas linguagens oferecem, já que carecem da intuição gráfica dada por outras ferramentas às quais estão habituados. Acreditamos que isto afeta negativamente a produtividade e desacelera a adopção das linguagens de programação probabilística [3].

2. Objetivos

Quisemos ultrapassar as dificuldades de aprender uma nova linguagem, tanto para progrmaadores inexperientes como para os mais experientes, como aprender uma nova sintaxe ou adapter aos idiomas de uma linguagem. É sabido que linguagens típicas são difíceis de aprender e usar [4] e que existem vantagens em providenciar uma linguagem com uma interface gráfica [5].

O objetivo desta dissertação foi desenvolver uma linguagem de Programação Visual (PV) com capacidades de programação probabilística. A audiência são programadores e *data scientists* com conhecimentos em estatística que não estejam ainda confortáveis com linguagens de programação probablística, mas desejam educar-se neste tópico para que eventualmente possam tirar partido do poder desta nova abordagem de *machine learning*.

3. Descrição do trabalho

Durante esta dissertação desenvolvemos um Ambiente de Programação Visual (APV), uma ferramente que permite ao utilizador desenhar o seu modelo graficamente e depois ou executá-lo ou transoformá-lo em código.

Ao fazê-lo providenciamos uma ferramente que pode ser usada para fazer investigação de usabilidade. Por exemplo, permite estudos futuros em que seja avaliado empiricamente se realmente pessoas com conhecimento de estatística aprendem mais rápido a usar uma linguagme de programação probabilística usando o APV em vez da forma textual. Também mostramos ser possível representação um programa probabilístico de forma gráfica representando na nossa ferramenta alguns exemplos existentes em Infer.NET, estando um deles representado na Fig. 1.

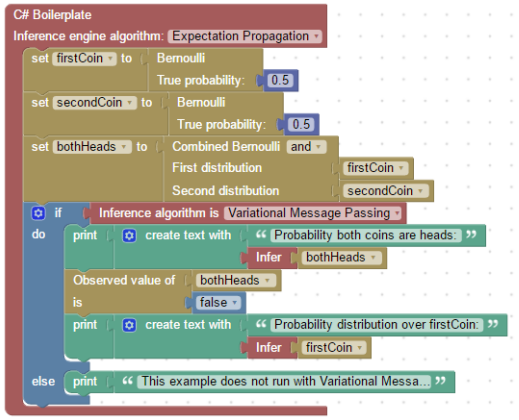

Fig. 1 – Exemplo de um modelo probabilístico no nosso ambinete.

During this process, we had to make choices regarding both the paradigms and the technologies in which we would base our VPE on, and we describe some of them below. In addition to detailing those choices, we are using the remainder of this section to elicit some VP concepts we find useful to have in a VP language for probabilistic programming.

3.1. Puramente Visual *vs* Ambiente

A diferença entre um APV e uma linguagem puramente visual é que, enquanto a segunda é uma linguagem *per se* (ou seja, há uma mapeamento direto entre os gráficos e a execução), os APV oferecem um meio termo entre linguagens textuais e puramente visuais: providenciam uma interface gráfica que pode ser usada para gerar código na linguagem alvo [6]. Estes dois conceitos são normalmente referidos na literatura como se fossem mutualmente exclusivos, mas incluímos ambos: temos um APV que pode executar

código, para que o utilizador possa ter *feedback* visual o mais cedo possível (não é necessário copiar o código para um ambiente diferente), o que é algo essencial em PV [7].

3.2. *Dataflow vs Blockly*

Ao ter que escolher como representar visual a linguagem, tivemos que optar entre duas alternativas. Enquanto programação *dataflow* é usada em aplicações comerciais bem sucessidades como *RapidMiner* ou *Blender*, representações *à Blockly* são mais vistas em ferramentas educacionais como o *Scratchy* do MIT.

Por causa da natureza da nossa audiência, pensamos que seria adequado para a nossa ferramente oferecer uma representação que pudesse gerar código mais semelhante ao escrito por pessoas, para que os utilizadores pudessem olhar para a nossa ferramente como uma rede de segurança para gerar código como se fossem eles a tê-lo escrito, até que fossem proficientes o suficiente para que não precisassem do APV. Devido a isto, decidimos usar uma representação *à Blockly*.

3.3. Conceitos de Programação Visual

Há um conjunto de conceitos de programação visual que parece ser essencial incluir para que os utilizadores possam ser mais rápidos e cometer menos erros aquando do desenvolvimento recorrendo a uma ferramente de PV, tais como:

- Comentários
- Duplicação de blocos
- Procurar blocos por nome
- Usar cores e ícones para transmitir significados
- Obter ajuda (ver detalhes da funcionalidade de um bloco)
- Serialização (gravar e carregar programas e também pode partilhá-los)
- Zoom, minimização e compressão de blocos (minimizar espaço utilizado quando os detalhes não importam)

3.4. Segurança de tipos

Devido a estarmos a usar um APV baseado em representações *à Blockly*, que é mais adequado para propósitos educacionais, é da maior importância ter segurança de tipos. Isto quer dizer que, além de não ser possível representar um programa com erros de sintaxe, também não é possível cometer erros de tipos (tais como usar um valor booleano para especificar a média de uma distribuição gaussiana). Isto é feito de forma natural: tal como num puzzle, blocos que violem as regras de tipos não encaixam. Esta abordagem é melhor do que esperar para ver erros de compilação ou de tempo de execução, já que o erro é apanhado mais cedo e o *feedback* visual é mais imediato.

4. Conclusões

É nossa crença que a representação gráfica é mais clara do que o seu dual textual, principalmente porque os parâmetros têm nomes e existem cores.

A maior razão para usar um APV ao invés da representação textual original é devido à rede de segurança que providencia e impede o utilizador de cometer certos erros. Por causa disto, acreditamos que este tipo de APV pode ser útil para ajudar *data scientists* que são inexperientes a programar a dar os primeiros passos no uso de uma linguagem de programação probabilística.

Contudo, se não olharmos para o APV como uma ferramente educacional mas sim como algo que não é transacional, pensamos que não é tão produtivo quando comparado com ferramentas como *RapidMiner*, *Excel* ou mesmo representações textuais. O maior motivo por trás disto tem a ver com o uso de uma representação *à Blockly* que faz com que não tenhamos uma linguagem puramente visual que não encapsula totalmente os conceitos de programação textual (como a declaração de variáveis). No caso de existir alguém a considerar desenvolver uma linguagem de PV para ser usada como substituto da forma textual de linguagens de programação probabilística, aconselhamos a explorar uma representação *dataflow*, já que funciona a um nível de abstração mais alto e tem sido usado com mais sucesso em aplicação usadas na indústria e na academia.

References

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

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

[3] 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.

[4] G Lewis and G Olson. Can principles of cognition lower the barriers to programming? *Empirical Studies of Programmers: Second Workshop*, 2(15):248–263, 1987.

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

[6] Margaret Burnett. Visual programming. *Wiley Encyclopedia of Electrical and Electronics Engineering*, 32(1-3):275–283, 1999.

[7] T.R.G. Green. Noddy's Guide to ... Visual Programming. *The British Computer Society • Human-Computer Interaction Group*, 1995.