Guião 2
Representação de Conhecimento e Inferência

Ano Lectivo de 2016/2017

©Luís Seabra Lopes

Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro

Última actualização: 2016-10-12

I Objectivos

O presente guião centra-se no tema da representação do conhecimento e mecanismos de inferência associados. Vamos trabalhar com uma implementação simples de redes semânticas, sendo fornecida uma versão inicial que os alunos deverão extender com novas funcionalidades. Relembre desde já que uma rede semântica representa o conhecimento na forma de um grafo, em que os nós representam entidades (objectos ou tipos) e as arestas representam relações entre entidades. Vamos também usar, de forma mais limitada, um módulo de redes de Bayes.

Este guião é usado nas disciplinas de *Inteligência Artificial*, da *Licenciatura em Engenharia Informática*, e *Introdução à Inteligência Artificial*, do *Mestrado Integrado em Engenharia de Computadores e Telemática*.

Este guião será realizado em 4 a 5 aulas práticas. ***Para um bom aproveitamente das aulas, os exercícios que estejam no âmbito temático de uma dada aula devem ser completados antes da aula seguinte.***

II Redes semânticas

1 Apresentação do módulo inicial

O módulo semantic_network, fornecido em anexo, exporta um conjunto de classes para representar relações semânticas:

```
class Relation:
    def __init__(self,e1,rel,e2):
        self.entity1 = e1
        self.name = rel
        self.entity2 = e2
    def __str__(self):
```

```
        return self.relation + "(" + str(self.entity1) + "," + \
               str(self.entity2) + ")"


class Association(Relation):
    def __init__(self,e1,assoc,e2):
        Relation.__init__(self,e1,assoc,e2)

class Subtype(Relation):
    def __init__(self,sub,super):
        Relation.__init__(self,sub,"subtype",super)

class Member(Relation):
    def __init__(self,obj,type):
        Relation.__init__(self,obj,"member",type)
```

Usando estas classes, podemos representar associações genéricas (classe `Association`), relações de generalização (classe `Subtype`) e relações de pertença dos objectos aos seus respectivos tipos (classe `member`). As relações de generalização e de pertença permitem herança de propriedades.

Exemplos:

```
>>> a = Association('socrates','professor','filosofia')
>>> s = Subtype('homem','mamifero')
>>> m = Member('socrates','homem')
>>> str(a)
'professor(socrates,filosofia)'
>>> str(s)
'subtype(homem,mamifero)'
>>> str(m)
'member(socrates,homem)'
```

Naturalmente, uma aplicação típica de uma rede semântica é a sua utilização para representar o conhecimento de um agente. O agente poderá obter o seu conhecimento de várias fontes, incluindo interlocutores (utilizadores) com os quais interage. Usando este módulo, podemos associar os interlocutores às relações por eles declaradas, usando a seguinte classe:

```
class Declaration:
    def __init__(self,user,rel):
        self.user = user
        self.relation = rel
    def __str__(self):
        return "decl("+str(self.user)+","+str(self.relation)+")"
```

Aqui, usamos a classe `Declaration` para registar que o interlocutor `user` declarou a relação `rel`.

Exemplos:

```
>>> da = Declaration('descartes',a)
>>> ds = Declaration('darwin',s)
>>> dm = Declaration('descartes',m)

>>> str(da)
'decl(descartes,professor(socrates,filosofia))'
```

```
>>> str(ds)
'decl(darwin,subtype(homem,mamifero))'
>>> str(dm)
'decl(descartes,member(socrates,homem))'
```

Finalmente, a classe `SemanticNetwork` representa uma rede semântica através de uma lista de declarações de relações:

```
class SemanticNetwork:
    def __init__(self):
        self.declarations = []
    def __str__(self):
        pass
    def insert(self,decl):
        self.declarations.append(decl)
    def query_local(self,user=None,e1=None,rel=None,e2=None):
        self.query_result = \
            [ d \
                    for d in self.declarations \
                    if  (user == None or d.user==user) \
                    and (e1 == None or d.relation.entity1 == e1) \
                    and (rel == None or d.relation.name == rel) \
                    and (e2 == None or d.relation.entity2 == e2) ]
        return self.query_result
    def show_query_result(self):
        pass
```

A função `query_local` permite obter informação local (ou seja, propriedades não herdadas) sobre as várias entidades presentes na rede. Esta função pode receber como parâmetros o utilizador (`user`), o nome da primeira entidade envolvida na relação (`e1`), o nome da relação (`rel`) e o nome da segunda entidade envolvida na relação (`e2`). A função vai devolver todas as declarações que satisfazem os parâmetros especificados, podendo alguns deles ser omitidos.

Exemplo de criação e consulta de uma rede semântica:

```
>>> z = SemanticNetwork()
>>> z.insert(da)
>>> z.insert(ds)
>>> z.insert(dm)
>>> z.insert(Declaration('darwin',Association('mamifero','mamar','sim')))
>>> z.insert(Declaration('darwin',Association('homem','gosta','carne')))
>>> z.insert(Declaration('descartes', Member('platao','homem')))
>>> z.query_local(user='descartes',rel='member')
........
>>> z.show_query_result()
decl(descartes,member(socrates,homem))
decl(descartes,member(platao,homem))
>>>
```

2 Exercícios

O módulo descrito acima é genérico mas contém algumas limitações. Seria interessante implementar algumas funcionalidades adicionais. Para teste das novas funcionalidades, é fornecido em anexo o módulo sn_example, o qual contém um exemplo de uma rede semântica com diversas

declarações já introduzidas. Os exemplos dados em algumas das alíneas reportam-se ao conteúdo deste módulo.

1. Programe uma função que, dadas duas entidades (dois tipos, ou um tipo e um objecto), devolva `True` se a primeira for predecessora (ou ascendente) da segunda, e `False` caso contrário.

```
>>> z.predecessor('vertebrado','socrates')
True
>>>
```

2. Programe uma função que, dadas duas entidades (dois tipos, ou um tipo e um objecto), em que a primeira é predecessora da segunda, devolva uma lista composta pelas entidades encontradas no caminho desde a primeira até à segunda entidade. Caso a primeira entidade não seja predecessora da segunda, a função retorna `None`.

```
>>> z.predecessor_path('vertebrado','socrates')
['vertebrado','mamifero','homem','socrates']
>>>
```

3. Programe uma função que devolva a lista (dos nomes) das associações existentes.
4. Programe uma função que devolva a lista dos objectos cuja existência pode ser inferida da rede, ou seja, uma lista das entidades declaradas como instâncias de tipos.
5. Programe uma função que devolva a lista de utilizadores existentes na rede.
6. Programe uma função que devolva a lista de tipos existente na rede.
7. Programe uma função que, dada uma entidade, devolva a lista (dos nomes) das associações localmente declaradas.
8. Programe uma função que, dado um utilizador, devolva a lista (dos nomes) das relações por ele declaradas
9. Programe uma função que, dado um utilizador, devolva o número de associações diferentes por ele utilizadas nas relações que declarou.
10. Programe uma função que, dada uma entidade, devolva uma lista de tuplos, em que cada tuplo contém (o nome de) uma associação localmente declarada e o utilizador que a declarou.
11. A função query_local() não faz herança de conhecimento, permitindo apenas consultar declarações locais das entidades.

    (a) Desenvolva uma nova função query() na classe SemanticNetwork que permita obter todas as declarações de associações locais ou herdadas por uma entidade. A função recebe como entrada a entidade e, opcionalmente, o nome da associação.

```
>>> z.query('socrates','altura')
.....
>>> z.show_query_result()
decl(descartes,altura(mamifero,1.2))
decl(descartes,altura(homem,1.75))
decl(simao,altura(homem,1.85))
decl(darwin,altura(homem,1.75))
```

(b) Desenvolva uma nova função query2() na classe SemanticNetwork que permita obter todas as declarações locais (incluindo `Member` e `Subtype`) ou herdadas (apenas `Association`) por uma entidade. A função recebe como entrada a entidade e, opcionalmente, o nome da relação. (Nota: Pode utilizar a função da alínea anterior para construir parte do resultado.)

12. Desenvolva uma nova função de consulta, query_cancel(), similar à função query(), mas em que existe cancelamento de herança. Neste caso, quando uma associação está declarada numa entidade, a entidade não herdará essa associação das entidades predecessoras. A função recebe como entrada a entidade e o nome da associação.

13. Neste tipo de rede semântica, dada a acumulação de declarações de vários interlocutores, podem ocorrer inconsistências no conhecimento registado. Existindo divergência nos valores atribuídos a uma associação numa entidade, faz sentido levar também em conta os valores herdados. Assim, desenvolva uma função query_assoc_value() que, dada uma entidade, E, e (o nome de) uma associação, A, devolva o valor, V, dessa associação nessa entidade, de acordo com a seguinte análise de casos:

    - Se todas as declarações locais de A em E atribuem o mesmo valor, V, à associação, então é retornado esse valor, não levando em conta os valores eventualmente herdados.
    - Caso contrário, retorna-se o valor, V, que maximiza a seguinte função:

      $$F(E, A, V) = \frac{L(E,A,V)+H(E,A,V)}{2}$$

      em que $L(E, A, V)$ é a percentagem de declarações de V para A na entidade E, e $H(E, A, V)$ é a percentagem de declarações similares nas nas entidades predecessoras de E.
    - Caso a associação não seja herdada, a função retorna o valor mais frequente, ou seja, o valor que maximiza $L(E, A, V)$.

14. Desenvolva uma função query_down() na classe SemanticNetwork que, dado um tipo e (o nome de) uma associação, devolva uma lista com todas as declarações dessa associação em entidades descendentes desse tipo.

15. Por vezes, na ausência de informação geral conhecida, pode ser útil usar inferência por indução. Neste caso, usa-se informação sobre entidades mais específicas (subtipos e/ou instâncias) para inferir propriedades gerais de um tipo. Assim, desenvolva uma função query_induce() que, dado um tipo e (o nome de) uma associação, devolva o valor mais frequente dessa associação nas entidades descendentes.

16. Até agora, considerámos que as associações admitem múltiplos valores. Por exemplo, a associação gosta pode admitir vários valores (pode-se gostar de peixe e de carne).

    Num sistema profissional, encontram-se outros tipos de associações. Há associações que admitem apenas um valor. Por exemplo, pai admite apenas um valor (o pai da pessoa em causa). Já no caso de associações com valor numérico (por exemplo altura ou peso), a existência de vários valores declarados pode ser tratada como uma distribuição, tomando-se a média desses valores como uma boa aproximação à verdade.

    Para testar as duas alíneas seguintes, descomente as declarações que estão comentadas no módulo sn_example.

(a) Desenvolva duas classes derivadas da classe Relation para representar associações com com um único valor (classe AssocOne) e associações com valores numéricos (classe AssocNum).

(b) Desenvolva uma nova função query_local_assoc () na classe SemanticNetwork, a qual permite fazer consultas de valores das associações locais de uma dada entidade, devendo processar os diferentes tipos de associações da seguinte forma:

- Association - Devolve uma lista de pares $(val, freq)$ com os valores locais mais frequentes e respectivas frequências. Na selecção dos valores mais frequentes, os valores são acrescentados à lista por ordem decrescente da sua frequência, até a soma das frequências atingir um valor não inferior a 0.75.
- AssocOne - Devolve um par $(val, freq)$, em que val é o valor local mais frequente, e $freq$ é a frequência (percentagem) com que ocorre.
- AssocNum - Devolve a média dos valores locais.

A função recebe uma entidade e (o nome de) uma associação, e devolve o resultado tal como descrito acima.

Exemplos:

```
>>> z.query_local_assoc('socrates','pai')
('sofronisco', 0.67)
>>> z.query_local_assoc('socrates','peso')
77.5
>>> z.query_local_assoc('homem','gosta')
[('carne', 0.40), ('peixe', 0.40)]
>>>
```

III Redes de Bayes

1 Apresentação do módulo inicial

O módulo bayes_net, fornecido em anexo, exporta um conjunto de classes para representar probabilidades condicionadas (classe `ProbCond`) e redes de Bayes (classe `BayesNet`). Esta última permite criar uma redes de Bayes especificando a respectiva lista de probabilidades condicionadas, ou introduzindo-as uma a uma após a criação da rede. Está já implementado um método para calcular a probabilidade conjunta joint_prob (conjunction).

2 Exercícios

1. Desenvolva um novo método que, dada uma variável da rede e um valor booleano, calcule a respectiva probabilidade individual.

2. Crie uma nova rede de Bayes para representar o conhecimento dado no exercício III.11 do *Guião Teórico-Pratico*.

3. Usando a rede da alínea anterior, calcule a probabilidade de um utilizador precisar de ajuda.