Trabalho prático individual nº 2

Inteligência Artificial / Introdução à Inteligência Artificial
Ano Lectivo de 2018/2019

23 November 2018

I Observações importantes

1. This assignment should be submitted via *Moodle* within 32 hours after the publication of this description. The assignment can be submitted after 32 hours, but will be penalized at 5% for each additional hour.

2. Complete the requested functions in module `"tpi2.py"`, provided together with this description. Keep in mind that the language adopted in this course is Python3.

3. Include your name and number and comment or delete non-relevant code (e.g. test cases, print statements); submit only the mentioned module `"tpi2.py"`.

4. You can discuss this assignment with colleagues, but you cannot copy their programs neither in whole nor in part. Limit these discussions to the general understanding of the problem and avoid detailed discussions about implementation.

5. Include a comment with the names and numbers of the colleagues with whom you discussed this assigment. If you turn to other sources, identify those sources as well.

6. All submitted code must be original; although trusting that most students will do this, a plagiarism detection tool will be used. Students involved in plagiarism will have their submissions canceled.

7. The submitted programs will be evaluated taking into account: performance; style; and originality / evidence of independent work. Performance is mainly evaluated concerning correctness and completeness, although efficiency may also be taken into acount. Performance is evaluated through automatic testing. If necessary, the submitted modules will be analyzed by the teacher in order to appropriately credit the student's work.

II Exercices

Together with this description, you can find the modules semantic_network and bayes_net, similar to the ones used in practical classes. Note however that there are some additions. In particular,

we now have a method for adding events, simply defined as sets of relations (`insertEvent(user, relations)`). Declarations now have timestamps, the field `tick` in class `Declaration`. Finally, there is a new type of relation (`Fluent`). Fluents are similar to associations. However, while associations can have multiple values at the same time, fluents must have a single value in each moment. You can assume that the test scenarios comply with these rules. The `BayesNet` class has a new method to display the network.

The module `tpi2_tests` contains a semantic network to test the functionalities to be developed. Part of it is inspired by the "Sistema Operativo do Futuro"(SOF) scenario that you also know. The module already contains several tests. If needed, you can add other test code in this module.

Don't change the semantic_network and bayes_net modules. Module `tpi2` contais the class `MySemNet` where you will develop some methods. All code that you are asked to develop should be integrated in the `tpi2` module.

1. An entity `A` is predecessor (or ancestor) of an entity `B` if there is a chain of `member` and/or `Subtype` relations connecting `B` to `A`. Develop a function that, given two entities (two types, or one type and one object), return `True` if the first is the predecessor of the second, and `False` otherwise. In the special case in which the two arguments are the same, the function should return true.

   Examples:

   ```
   >>> z.predecessor('mammal','antonio')
   True

   >>> z.predecessor('cat','faceState')
   False
   ```

2. Develop a method `get_entity_events()` that returns a list of pairs `(entity,tick)`. Each such pair means that the entity was first argument of at least one fluent (relation `Fluent`) declared in instant `tick`.

   Example:

   ```
   >>> z.get_entity_events()
   [ ('marta', 55), ('carlos', 50), ('joana', 40),
     ('antonio', 35), ('teresa', 45) ]
   ```

3. Develop a method `get_occurrence(entity,tick,fluentnames)` that returns a dictionary associating, for each fluent name, the second argument of the fluent (value) to that fluent name (key) as declared in the given time instant.

   Example:

   ```
   >>> z.get_occurrence('joana',40,['hasEmailState','hasWorkLoad'])
   { 'hasWorkLoad': 'highLoad', 'hasEmailState': 'acumulated' }
   ```

4. Develop a method `abstract_from_descendents(type)` that recursively collects and abstracts information (associations) known for entities descending of the given `type`. The output takes the form of a list of pairs `(assoc,ent,count)` where `count` is the number of occurrences of entity `ent` as value (second argument) of the association with name `assoc` in the descendents of the given `type`. Note that `ent` can be the type of an object or the supertype of a type ocurring in a lower level. As information is propagated bottom-up,

the entities are progressively replaced by more abstract entities. When the association has the same value (entity) as its second argument in all declarations, this entity is propagated upwards. However, if the association has different entities of the same type in different declarations, this type is propagated upwards.

Example:

```
>>> z.abstract_from_descendents(tareco)
[ ('eats', 'sardine', 1) ]

>>> z.abstract_from_descendents(micas)
[ ('drinks', 'milk', 1), ('eats', 'sardine', 1) ]

>>> z.abstract_from_descendents(cat)
[ ('eats', 'sardine', 2), ('drinks', 'milk', 1) ]

>>> z.abstract_from_descendents(human)
[ ('drinks', 'milk', 3), ('eats', 'vegetable', 2), ('eats', 'meat', 2),
  ('eats', 'fruit', 2), ('eats', 'fish', 2) ]
```

5. The module `tpi2_tests` contains several events that correspond to problems experienced by the users of the SOF2019 operating system. This data can be used to compute conditional probabilities and create a Bayesian network similar to the one hand-coded in classes.

   Consider this in a more general setting. Suppose that the dependencies are specified by a dictionary as in the following example:

```
dependSOF = \
{ ('hasEmailState','acumulated')       : [ ('hasWorkLoad','highLoad') ],
  ('hasFaceState','worried')           : [ ('hasWorkLoad','highLoad'),
                                           ('needsHelp','yes') ],
  ('exageratedUsing','computerMouse') : [ ('using','SOF2019-PAL'),
                                           ('needsHelp','yes') ],
  ('needsHelp','yes')                  : [ ('using','SOF2019-PAL') ] }
```

   Develop a method `makeBN(dependencies)` that takes as input one such specification and returns the complete Bayesian network.

   This is the intended result:

```
>>> bn = z.makeBN()
>>> bn.display()
('exageratedUsing', 'computerMouse') :
  [(('needsHelp', 'yes'), True), (('using', 'SOF2019-PAL'), True)]-->0.5
  [(('needsHelp', 'yes'), False), (('using', 'SOF2019-PAL'), False)]-->0.5
  [(('needsHelp', 'yes'), False), (('using', 'SOF2019-PAL'), True)]-->0.0
  [(('needsHelp', 'yes'), True), (('using', 'SOF2019-PAL'), False)]-->1.0
('hasEmailState', 'acumulated') :
  [(('hasWorkLoad', 'highLoad'), True)]-->0.6666666666666666
  [(('hasWorkLoad', 'highLoad'), False)]-->0.0
('hasFaceState', 'worried') :
  [(('hasWorkLoad', 'highLoad'), True), (('needsHelp', 'yes'), False)]-->1.0
  [(('hasWorkLoad', 'highLoad'), True), (('needsHelp', 'yes'), True)]-->1.0
  [(('hasWorkLoad', 'highLoad'), False), (('needsHelp', 'yes'), True)]-->0.5
  [(('hasWorkLoad', 'highLoad'), False), (('needsHelp', 'yes'), False)]-->0.5
('needsHelp', 'yes') :
```

```
[(('using', 'SOF2019-PAL'), False)]-->1.0
[(('using', 'SOF2019-PAL'), True)]-->0.5
```

III Clarification of doubts

This work will be followed through `http://detiuaveiro.slack.com`. The clarification of the main doubts will be placed here.

1. Não consegui perceber a diferença entre `Association` e `Aluent`. Os exemplos dos testes parecem todos intermutáveis.

   Resposta: Do ponto de vista sintático, só muda o nome, mas o significado é também diferente. As associações do nosso módulo podem ter vários valores ao mesmo tempo Já o `Fluent` só pode ter um valor de cada vez. Note que há o `tick` (instante de tempo) de cada declaração; quando chega um `Fluent` com `hasFaceState=happy`, o valor anterior (`worried`, por exemplo) passa a história.

2. Como assim, as associações podem ter varios valores ao mesmo tempo?

   Resposta: Sempre usámos assim: `socrates professor` de `matematica` e de `filosofia`; `A` amigo de `B`, `C`, e `D` ... etc. Num dos exercícios do guião, propunha-se o acrescento de outras relações, que seriam outros tipos de associações (`AssocOne`, `AssumNum`).

   Neste trabalho saiu o `Fluent`. A rede semântica acaba por ser o histórico de tudo o que se passou. Os métodos de consulta à rede semântica é que têm que levar em conta o valor de cada declaração. Foi sempre essa a filosofia.

3. (5:04PM) Foram corrigidas gralhas no módulo `tpi2_tests`

4. A ordem do output dos exercicios 2 e 3 é relevante?

   Resposta: Não.

5. No exercício 3, podemos assumir que o argumento `fluentnames` são sempre nomes de relações `Fluent`?

   Resposta: Sim, podem assumir; tudo o que sejam pressupostos explícitos no texto, podem assumir que são satisfeitos. Neste caso, se o enunciado fala de fluentes, são fluentes.

6. (5:38PM) Mais gralhas corrigidas no módulo `tpi2_tests`. Exemplos do enunciado corrigidos em função disso.

7. No ex. 4, em `'cat'`, aparece `('eats', 'sardine', ... )`, mas em `'human'` aparece `'fish'`. Porquê?

   Resposta: Tem a ver com esta frase do enunciado: "When the association has the same value (entity) as its second argument in all declarations, this entity is propagated upwards."Quer dizer que só abstraimos para um tipo superior quando temos evidência de vários subtipos. No caso da `'sardine'`, como não aparece mais nada em `'cat'` nem nas instâncias, mantem-se a entidade mais especifica. No caso de `'human'`, como aparecem vários tipos de peixe, já abstrai para `'fish'`.

8. No exercicio 5, como é suposto chegar às probabilidades?

   Resposta: São probabilidades condicionadas a calcular com base nos fluentes dos vários eventos do tema 'SOF2019'. A fórmula da probabilidade é $P(A|B) = P(A\&B)/P(B)$. Os únicos dados são os do módulo `tpi2_tests`.

9. Poderia mostrar os resultados dos testes do exercício 4 com as entidades `'antonio'` e `'mammal'`?

   Resposta: Aqui está:

```
>>> z.abstract_from_descendents(antonio)
[ ('eats', 'cow', 1), ('eats', 'cabbage', 1), ('drinks', 'milk', 1),
  ('eats', 'sardine', 1) ]

>>> z.abstract_from_descendents(mammal)
[ ('drinks', 'milk', 4), ('eats', 'food', 10) ]
```

10. No exercicio 5, `('needsHelp','yes'): [(('using', 'SOF2019-PAL'), True)]` deveria dar 1 ou 0.5?

    Resposta: De facto, no exemplo estava 1.0, mas dá 0.5. Quando se corrigiu a gralha da duplicação do fluente 'using' no `tpi2_tests` o resultado deve ter-se alterado. Corrigi o exemplo.

11. Aproveitei também para melhorar o método `display()`, para apresentar a informação de forma mais ordenada, facilitando a comparação.

12. A probabilidade: `('hasFaceState', 'worried') : [(('hasWorkload', 'highLoad'), False), (('needsHelp', 'yes'), False)] --> 0.5` deverá estar errada, porque não existe nenhum utilizador que tenha `worriedFace`, `highLoad` e `needsHelp` ao mesmo tempo.

    Resposta: A probabilidade está certa. A razão é que, na ausência de evidência, a probabilidade atribuida é 0.5.

13. A solução do exercício 4 tem de ser estritamente recursiva? Há alguma valorização no uso exclusivo de recursividade?

    Resposta: Não é obrigatório; tem é que dar o resultado pretendido. Não há valorização por ser recursivo.

14. `Association('marta','eats','vegetable')` é possível?

    Resposta: Na verdade é indiferente para o trabalho, uma vez que os exercícios não misturam fluentes com associações.

15. `'vegetable'`, `'meat'`, `'fruit'` e `'fish'` são entidades diferentes do mesmo tipo que estão em declarações. O último exemplo do exer. 4 (`'human'`) não devia de ser `[('drinks', 'milk', 3), ('eats', 'food', 8)]` ?

    Resposta: Abstrai-se no máximo um nível de cada vez: das instâncias de `'homem'` para o tipo `'homem'`, abstrai-se dos tipos específicos de comidas (`'orange'`, `'pork'`, etc.) para os tipos intermédios (`'vegetable'`, ...)

16. Relativamente ao exercício 4, só as instâncias é que podem ter associações? Ou podemos ter `Association('human','eats','carrot')` e `Association('mammal','eats','carrot')`?

    Resposta: Nada sendo dito em contrário, o módulo funciona como era habitual, ou seja, pode haver várias declarações de vários utilizadores, pode haver associações nos tipos. Essas que refere em particular são possíveis.

17. Aquilo que tiver subtipos, poderá ter membros (e vice-versa)?

    Resposta: A existência simultânea de subtipos e membros num mesmo tipo não será muito normal, mas nunca impedimos isso, e também não há razão para impedir isso agora.

18. Ainda não consegui perceber o ex. 4.

    Resposta: No exercicio 4, a abstracção das entidades mais específicas para cima é feita considerando o valor das associações. Quando se diz 'valor', estamos a referir o segundo argumento da associação. Por exemplo, o valor de `'eats'` em `'tareco'` é `'sardine'`; como `'micas'` também `'eats' 'sardine'`, e como a `'micas'` e o `'tareco'` são `'cat'`, conclui-se que `'cat' 'eats' 'sardine'`, uma vez que não há nenhuma excepção a esta regra. Já no caso de `'human'`, há várias instâncias que comem diferentes tipos de coisas; nesse caso, vai-se abstrair o valor dessa associação; de `'cabbage'` e `'carrot'`, abstrai-se para `'vegetable'`, etc.