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 secong argument of the fluent (value) to that fluent name (key).

   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', 'fish', 2), ('drinks', 'milk', 1) ]

>>> z.abstract_from_descendents(human)
[ ('eats', 'vegetable', 2), ('eats', 'meat', 1), ('eats', 'fruit', 2), ('eats', 'fish',
```

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

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. . . . **Resposta**: . . .