Project: Probabilistic models in Pacman

Introduction to the project, programming highlights and insights

MDP Project. 2015-16

Schema

Summary and goals

- We will study a practical application of this course's lesson about Markov Decision Processes (MDPs).
- We will also apply automatic learning.
- The project will be evaluated according to your submission material.
- We will use a IA application package written in Python.
- **Supported platforms**:
  - Linux is the reference platform
  - The reference python version is 2.7

Structure

- Part 1:
  - Software installation
  - Running examples
  - Writing the value iteration algorithm code
  - Running tests with predefined examples, variation of parameters
- Part 2:
  - Writing the code for the MDP learning module
  - Running tests with predefined examples
- Part 3:
  - Design and testing of different scenarios

Differences with Java

- The code **HAS** to be indented (do not use keys)
- You have to define the functions or the clases before using them
- Comments start with # or with chunks identified by 3 simple quotation marks
- Most of the mistakes appear in execution time
- Logic operators are `and`,`not`,`or`
- Strings are defined with simple or double quotation marks. The strings' concatenation operator is '+'. They allow access with slices' sintaxis (see lists).
- The print command admits several arguments of different types separated by commas
- To list methods use `dir`

Loading the interpreter

- Execution of code and programs
  - Entering and existing the interactive interpreter (observe that it is the version 2.7)

```
yourname@yourhost:~$ python
Python 2.7.3 (default, Jun 22 2015, 19:43:34)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> quit()
```

  - You run a module indicating the file name (extension ".py") and the options

```
yourname@yourhost:~$ python gridworld.py -m
```

  - You load a file with the interpreter with `import <module> as <alias>`, you access its functions with `<alias>.funcion` and your reload and modify with `reload(<alias>)`

```
>>> import os
>>> import sys
>>> sys.path.append(os.path.abspath(<path-to-module>))
>>> import module as alias-module
```

  - You can request the help option with `help()` or search for a specific element with `help(list)`

There is several online information `https://wiki.python.org/moin/SpanishLanguage`, Coursera, etc

Variable Access

- Types
  - The language does not require the declaration of variable types
  - Besides de classic ones, we have collections: tuples (immutable), lists, sets and **dictionaries**[1]
  - All of them can be conformed by different types, including other collections
- Tuples are **initialized** with parenthesis, lists with brackets and dics with keys
- Type can be consulted with `type`

```
>>> myTuple=(1,"Second element",3.0)
>>> myList=[1,"Second element",3.0]
>>> myDictionary={"First":1,"Second":"Second Element","Third":3.0}
```

- To **access** content, write the name and use the notation `variable[]`
- Slicing (lists and tuples): `v[start:end]` access from both ends `v[-1]`. First element has index 0.
- **Important**: The element end is not included in the slice. Start and end are optional.

```
>>> myTuple[1]
'Second element'
>>> myList[0:2]
[1, 'Second element']
>>> myList[-1]
3.0
```

```
>>> myTuple[1:]
('Second element', 3.0)
>>> myDictionary["Third"]
3.0
```

[1] Hint: We will use this one in the project

Variable Modification

- You can access the index of a value through `index()` and the length of a collection through the function `len(<variable>)`.
- Lists admit methods `append()`, `pop()`, etc. Also the concatenation operator '+'.
- Dictionaries admit the function `del`, and the methods `keys()`, `values()`, `items()`.
- To **modify** content (save for tuples) user `variable[start:end]=value`

```
>>> myList.index('Second element')
1
>>> myList + [ 4, "A fourth element" ]
[1, 'Second element', 3.0, 4, 'A fourth element']
>>> myList.append([ 4, "A fourth element" ])
>>> myList
[1, 'Second element', 3.0, [4, 'A fourth element']]
>>> myList.pop()
[4, 'A fourth element']
>>> myList
[1, 'Second element', 3.0]

>>> myTuple[0]=6
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    myTuple[0]=6
TypeError: 'tuple' object does not support item assignment
>>> myList[0]=6
>>> myList
[6, 'Second element', 3.0]
>>> myList[0:2]=[ 100,"A different second element"]
>>> myList
[100, 'A different second element', 3.0]
```

Programming in Python

- Loops:

```
>>> for element in myList:
        print "Element: ", element


Element:  100
Element:  A different second element
Element:  3.0
```

- Functions:

```
>>> def pow2(n):
        return n*n

>>> pow2(4)
16
```

- Classes: you access attributes (members, methods, etc.) using a dot (.). Parenthesis are necessary to call a method
- When invoking methods with an object, yo do not provide the **self** parameter: `print mySquare.area()`

```
class Square:
        def __init__ (self,side):
                self.side = side
        def area(self):
            return pow2(self.side)


>>> mySquare=Square(5)
>>> mySquare.side
5
>>> mySquare.area
<bound method Square.area of <__main__.Square instance at 0x03085B20>>
>>> mySquare.area()
25
```

Frequent errors in Python

- **Problem**: ImportError: No module named py

  When using import, you do not include the extension ".py"
- **Problem**: NameError: name 'MYVARIABLE' is not defined

  To access a variable you have to write a name or alias of the module mdulo: `modulo.MYVARIABLE` o `modulo.MYFUNCTION`
- **Problem**: TypeError: 'dict' object is not callable

  To obtain the value of a `dictionary` variable you need to use brackets `dict[''element'']`, not parenthesis
- **Problem**: AttributeError: 'list' object has no attribute 'length' (or something similar)

  The length of a list can be obtained with the function `len(LISTA)`, **not** with a method `LISTA.len()`
- **Problem**: Changes made in a file are not working

  You need to save the changes and **reload** the module `reload(MODULE)` o `reload(ALIAS-MODULE)`

Counter() Class

- In the project we use the class util.Counter
- It is in the file util.py
- It is a dictionary for numeric values. It is automatically initialized to 0.
- It includes operations to compute the max value, mean, normalization, sum of Counters, etc.

```
class Counter(dict):
    """
    A counter keeps track of counts for a set of keys.
(...)
    >>> a['blah'] += 1
    >>> print a['blah']
    1

    The counter also includes additional functionality useful in implementing
    the classifiers for this assignment.  Two counters can be added,
    subtracted or multiplied together.  See below for details.  They can
    also be normalized and their total count and arg max can be extracted.
    """

    def incrementAll(self, keys, count):
    def argMax(self):
    def sortedKeys(self):
    def totalCount(self):
    def normalize(self):
    def divideAll(self, divisor):
    def copy(self):
```

MarkovDecisionProcess() Class

- It is in the file `mdp.py`
- Each example has an implementation of this class that defines its elements:
  - States
  - Actions
  - Transition probabilities
  - Rewards
- The class allows us to access the list of all these elements

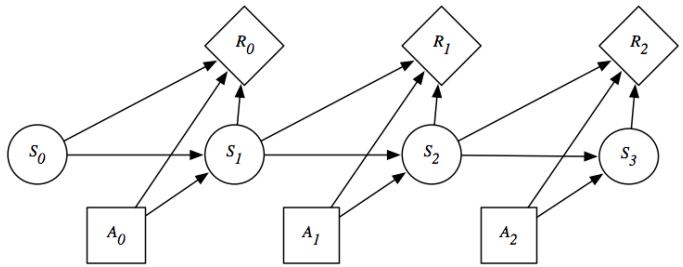
```
class MarkovDecisionProcess:

    def getStates(self):
    def getStartState(self):
    def getPossibleActions(self, state):
    def getTransitionStatesAndProbabilities(self, state, action):
    def getReward(self, state, action, nextState):
    def isTerminal(self, state):
```

Value with reward and discount

- In the most general case, reward values depend on the current state, the action, and the final state ($R(s, a, s')$). In this case Bellman's equation looks like:

$$Q(s, a) = \sum_{s'} P(s' \mid s, a)[R(s, a, s') + \gamma V(s')]$$

$$V(s) = \arg\max_{a} Q(s, a)$$



  - You need to first compute **Q-values** and then obtain the **maximum**
  - The reward R can be either negative or positive
  - The reward function will be a table or a function that depends on the state, that, must go inside the sum (if it depends only on the action you can take it out)
  - **Value Iteration**: Remember, we first compute the $i + 1$ iteration completely, next, we update the values.

- The parameter $\gamma \in [0, 1]$ is called **discount**, it represents the relative significance between the future reward and the current reward. Higher values give more relevance to future rewards. This is used to solve the convergence problem that occurs if there is not a finite number of transitions.

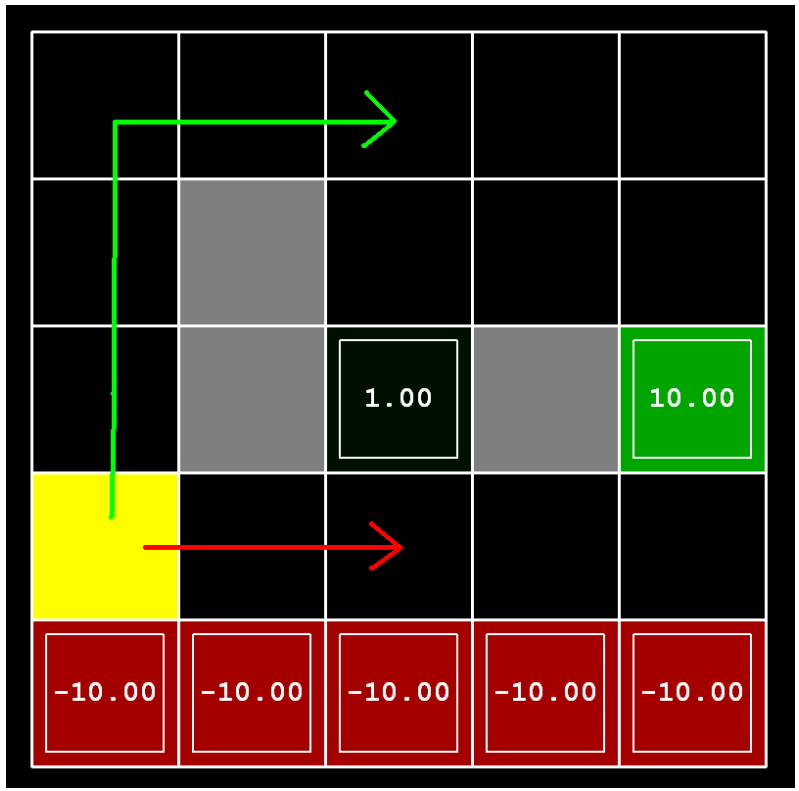
1 Image from: `http://artint.info/html/ArtInt_224.html`

Reward and output states

- In the grid examples, we consider that the reward function only depends on the current state: $R(s)$

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s' \mid s,a) V(s')$$

- In the states where no number appears $R(s)$ we face the *living-reward*, that is an input parameter. Its value is 0 by defect.
- In the states with a number, $R(s)$ is that number. In this states there is only one applicable function *exit* that takes the system to the final state (it is not represented graphically)
- The reward in the final state is 0.



Starting

- Use Linux.
- Use Python 2.7.
- You need to install python -tk if you do not have it.
- **1st Part**:
  - Study the value iteration formula and algorithm presented in the project description
  - Searching google for further information is a good habit
  - You need the value iteration algorithm to work properly for the rest of the project
  - It shouldn't take you more than 1 session, two at most to finish this part
  - Check python dics, check counter()

Session 2

- To test your code there is an autograder that performs tests and tells you if you have it right.
- To run the autograder you sometimes need to write your answers in a specific file.
- You need to play with the parameters to obtain the expected policy.
- Run in the autograder the different steps: q1, q2, q3
- Write your answers in the file analysis.py
- **2nd Part**:
  - You have an MDP class already implemented.
  - From MDP two other clases are inherited Mdp gridworld, Mdp pacman (this one is partial, pacmanMdp.py).
  - You need to use some of the things implemented in these classes.

Pacman part

- We are learning through frequencies
- We assume we do not know the mdp.
- We generate experiences so that we can obtain the frequencies.
- You need to explore the space in order to learn
- We learn the model
- With the tuple (state,action,next state, reward) we generate the transition function
- Each time we have a food ball we count it as one state
- The value iteration is partial

Pacman part II

- You do not need to touch the characteristics extractor (featureexractions.py) just to learn how it works
- We represent states through a matrix, low representation.
- However, we change the matrix to a tuple for the mdp
- The tuple is (xpacman,ypacman,xnearestghost,ynearestghost,xnearestfood,ynearestfood) = state representation, high representation
- The mdp has a tuple the transition function uses a matrix
- A high level representation state corresponds to several low level representations. As I do not know how mdp works I need to learn through rewards.
- Rewards are already programmed you do not need to touch them

Pacman part III

- You need to do your programming in pacmanmdp.py
- You should use a dictionary to do the recommendations
- Experiences are randomly generated, and this is already implemented.
- You need to do traces (prints) to check that your transition function is working ok.
- Estimate with pacmanMdpAgent.py
- x= number of training episodes
- n= x + number of tests you want to perform
- k= number of ghosts
- Do iterations with the value iteration function. Remember that the reward factor is 0.8
- Note that if you do not train enough there are states that will not appear in the value function.
- If I pass through a not trained state the behaviour will be random

Part III

- Define your own scenarios
- Study and define how it escalates, how it evaluates
- To define scenarios go to layouts (smallGrid.lay,original....lay)
- Remember that the bigger the scenario is the more difficult to learn is
- Describe them in the report