CSE 4309 - Assignments - Assignment 7

List of assignment due dates.

The assignment should be submitted via Canvas. Submit a file called assignment7.zip, containing the following two files:

- answers.pdf, for your answers to the written tasks, and for the output that the programming task asks you to include. Only PDF files will be accepted. All text should be typed, and if any figures are present they should be computer-generated. Scans of handwriten answers will NOT be accepted.
- value_iteration.py, containing your Python code for the programming part. Your Python code should run on the versions specified in the syllabus, unless permission is obtained via e-mail from the instructor or the teaching assistant. Also submit any other files that are needed in order to document or run your code (for example, additional source code files).

These naming conventions are mandatory, non-adherence to these specifications can incur a penalty of up to 20 points.

Your name and UTA ID number should appear on the top line of both documents.

Task 1 (70 points, programming)

In this task you will implement the value iteration algorithm.

Function Name and Arguments

File value_iteration_main.py contains incomplete code that implements the value iteration algorithm.

To complete that code, you must create a file called `value_iteration.py`, where you implement a Python function called `value_iteration`. Your function should be invoked as follows:

```
value_iteration(environment_file, non_terminal_reward, gamma, K)
```

The arguments provide the following information:

- The first argument, `environment_file`, is the path name of a file that describes the environment where the agent moves (see details below). The path name can specify any file stored on the local computer.
- The second argument, `non_terminal_reward`, specifies the reward of any state that is non-terminal.
- The third argument, `gamma`, specifies the value of γ that you should use in the utility formulas.
- The fourth argument, `K`, specifies the number of times that the main loop of the algorithm should be iterated. The initialization stage, where `U[s]` is set to 0 for every state s, does not

count as an iteration. After the first iteration, if you implement the algorithm correctly, it should be the case that `U[s]=R[s]`.

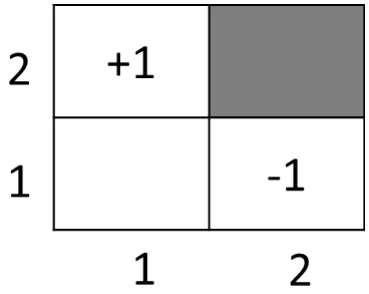

Figure 1: The environment described in file environment1.txt.

The environment file will follow the same format as files environment1.txt and environment2.txt. For example, file environment1.txt describes the world shown in Figure 1, and it has the following contents:

```
1.0,X
.,-1.0
```

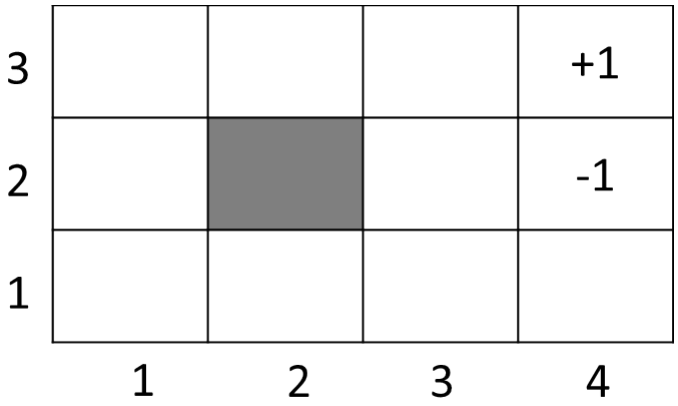

Figure 2: The environment described in file environment2.txt.

Similarly, file environment2.txt describes the world shown in Figure 2, and it has the following contents:

```
.,.,.,1.0
.,X,.,-1.0
.,.,.,.
```

As you see from the two examples, the environment files are CSV (comma-separated values) files, where:

- Character '.' represents a non-terminal state.
- Character 'X' represents a blocked state, that cannot be reached from any other state. You can assume that blocked states have utlity value 0.
- Numbers represent the rewards of TERMINAL states. So, if the file contains a number at some position, it means that that position is a terminal state, and the number is the reward for reaching that state. These rewards are real numbers, they can have any value.

Implementation Guidelines

- For the state transition model (i.e., the probability of the outcome of each action at each state), use the model described in pages 9-10 of the MDP slides.
- For terminal states, your model should not allow any action to be performed once you reach those states. For those states, you can just hardcode that their utility is equal to their reward.
- For blocked states, your code should capture the fact (by implementing the appropriate transition model) that they cannot be reached from any other state. You should hardcode the utility values of blocked states to be 0.

Output

At the end of your program, you should print out the utility values for all states, as well as the optimal policy.

The output should follow this format:

```
utilities:
%6.3f %6.3f...
...

policy:
%6s %6s...
...
```

More specifically:

- In your utilities printout, each row corresponds to a row in the environment, and you use the %6.3f format specifier for each utility value. For blocked states, just print a utility of 0.
- For the policy printout, again each row corresponds to a row in the environment, and you use the %6s format specifier for each action. To encode each action (or special state) use these characers:
    - For the four actions, use characters `'<'` for `left`, `'>'` for `right`, `'^'` for `up`, and `'v'` for `down`.
    - For blocked states, use character `'x'`.
    - For terminal states, use character `'o'`.

Do NOT print out this output after each iteration. You should only print out this output after the final iteration. As an example, this is the output of my implementation, for `<environment_file>="environment2.txt"`, `<non_terminal_reward>=-0.03`, `<gamma>=0.98`, and `<K>=20`:

```
utilities:
 0.781  0.846  0.906  1.000
 0.724  0.000  0.646 -1.000
 0.662  0.608  0.570  0.354

policy:
     >      >      >      o
     ^      x      ^      o
     ^      <      ^      <
```

Output for answers.pdf

In your answers.pdf document, you need to provide the complete output for the following invocations of your function:

```
value_iteration('environment2.txt', -0.04, 1, 20)
value_iteration('environment2.txt', -0.04, 0.9, 20)
```

Task 2 (10 points)

Suppose that you want to implement a reinforcement learning algorithm for learning how to play chess.

- What value would you assign for the reward of the non-terminal states? Why?
- What value would you use for the discount factor γ? Why?

Your choices should be the best choices that you can make so that your algorithm plays chess as well as possible (i.e., it wins in as many situations as possible). Do not worry about running time, assume that you have enough computing resouces to run your algorithm sufficiently fast.

Task 3 (20 points)

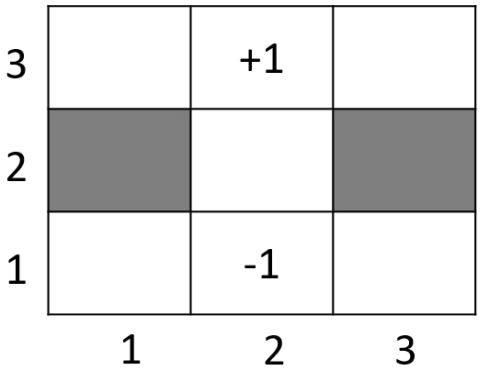

Figure 3: The environment to be considered in this task.

Consider the environment shown on Figure 3. States (1,2) and (3,2) are terminal, with utilities -1 and +1. States (2,1) and (2,3) are blocked. Suppose that actions and state transition models are as described in pages 9-10 of the MDP slides.

Part a: Suppose that the reward for non-terminal states is -0.04, and that γ=0.9. What is the utility for state (2,2)? Show how you compute this utility.

Part b: Suppose that γ=0.9, and that the reward for non-terminal states is an unspecified real number r (that can be positive or negative). For state (2,2), give the precise range of values for r for which the "up" action is not optimal. Show how you compute that range.

Task 4 (Extra Credit, maximum 10 points).

In this extra credit task, we revisit supervised learning. Now that you have learned several supervised learning methods, try to come up with the best approach that you can, to get the best accuracy on any of the three UCI datasets (pendigits, satellite, yeast) that we have been using in previous assignments. You are free to use, or invent, any supervised learning method you want, and apply it with any design choices and hyperparameters you want. If you do this task, please submit a Python file called supervised_learning.py, that implements your modifications in a function invoked as `supervised_learning(training_file, test_file, hyperparameters)`. The third argument is a Python object that contains all the hyperparameters that your function needs.

To produce output in a uniform manner, your function should print the result for every test object, as we did in previous assignments, using a line like:

```
print('ID=%5d, predicted=%10s, true=%10s, accuracy=%4.2f' % (object_id,
predicted_class, true_class, accuracy))
```

You should use the same rules for ties that we have been using in previous assignments.

After you have printed the results for all test objects, you should print the overall classification accuracy, which is defined as the average of the classification accuracies you printed out for each test object. To print the classification accuracy in a uniform manner, use this Python line:

```
print('classification accuracy=%6.4f' % (classification_accuracy))
```

A maximum of 10 points will be given to the submission or submissions that, according to the instructor and GTA, achieve the best accuracy (on any of the three datasets). In your answers.pdf document, under a clear "Task 4" heading, explain what method you used, what design choices you made, what hyperparameters you selected, and what results you achieved.
