CSE, IITB

CS 335 - AI & ML
Labs 7 & 8 - MDP & Ensemble Methods (11 marks)

Please read the following important instructions before getting started on the assignment

1. The assignment should be completed individually. Do not look at solutions to this assignment or related ones on the Internet.
2. Solutions to the theory questions must be submitted in a single pdf file.
3. All the hyperparameters must be specified in pdf file under **Hyperparameter** section and resources consulted must be duly listed in the **References** section of the pdf file. Remember to embed plots, if any, inside pdf. **Do not upload multiple pdf files.**
4. **Upload Guidelines**: Put all the assignment related files in folder with the convention **roll_no** and .zip the folder (**no tar.***).
5. **Not following folder guidelines will attract penalty.**
6. All source code are checked with code plagiarism checker. Strict action will be taken against the defaulters.

Q1 : MDP (5 marks)

Task 1 : (5 marks)

In this Problem you will be given a 2d NxM matrix with each cell labelled 0, 1, 2, 3, ..., N*M-1 in row major order. Each cell contains values from the set {0,1,2,3} where 2 is start state, 1 is a obstacle, 0 is free state and 3 is the goal state. Our aim is to find a path from start state to goal state. You have do 4 actions from a state - {0, 1, 2, 3} indicating {North, East, South, West} directions respectively. It is guaranteed that there will be only 1 start state and 1 goal state. You need to model this problem as an mdp instance. You can use value iteration algorithm to find the optimal value function & optimal policy.
`python3 mdp.py --input input.txt --mdpfile mdp.txt.`
output mdp.txt in the format given below.
`python3 solve.py --mdpfile mdp.txt --output output.txt.`
output output.txt in the format given below.
Example of input.txt format:
1 1 1 1
1 2 1 1
1 0 0 3
1 1 1 1

mdp.txt format:
numStates x
numActions a

start s
end e1 e2 e3 ... en
transition s1 ac s2 r p
transition s1 ac s2 r p
. . .
transition s1 ac s2 r p
discount gamma

where x are total number of states, a are number of actions, s is the start state label, e1,e2,e3,...,en are end state labels, transition definition remains same, 's1' upon action 'ac' goes to 's2' with probability 'p' and gets a reward 'r'. and discount is 'gamma'. you may not print transitions with probability 0 and can be in any order.

output.txt format:
V*(0) π*(0)
V*(1) π*(1)
. . .
V*(numstates-1) π*(numstates-1)

where V* is optimal value function and π* is optimal policy.

- We are going to use your mdp.txt as input to our code and check if your reported output.txt is correct or not
- We also check your solve.py with different test cases along with maze formulated mdp-file. So do code it for general form of mdp format.
- Please mention the assumptions you have taken while formulating the mdp (they have to be sensible) , your design of mdp, algorithm used, convergence criteria
- states can be assumed to be labelled from 0 to numstates-1 in solve.py and similarly with actions(0 to numactions-1)
- output 'end -1' if there are no end states
- for optimal value maintain precision upto 6 decimal places
- submit mdp.py & solve.py for this task

Q2 : Bagging & Boosting (6 marks)

Ensemble Learning, a form of meta-learning, is a machine learning paradigm where multiple learners are trained to solve the same problem. In this assignment, you will code up the meta-learning algorithms Bagging (short for Bootstrap Aggregating) and Boosting (specifically

AdaBoost). These meta-learners will operate on perceptron as a base learner, which you have already worked on in Lab 3.

The data set on which you will run your meta-learning algorithms is a collection of hand-written numerical digits (0-9). You have to solve only a two-class problem – classes (0-4) have the label -1, and classes (5-9) have the label 1. You can reuse the code for perceptron from Lab 3 for filling in `perceptron.py`; you will have to write new code in `bagging.py` and `boosting.py`.

Task 1 : Bagging (3 marks)

Your task is to implement the 2-class bagging classifier with perceptron as the weak learner. Fill out code in the `train()` and `classify()` functions at the location indicated in `bagging.py`.

`train()` should contain the code to sample points from the data set with replacement and then training the weak classifier using the sampled data set. `classify()` should contain the code to get the labels from individual classifiers and then running voting to find the majority.

Run your code with this command:

```
python dataClassifier.py -c bagging -t 1000 -s 1000 -r 1 -n 20
python autograder.py -t 2
```

Your classifier will be evaluated for its accuracy on the test set after the Bagging algorithm has been run with weak learners trained for the default 3 iterations. You will be awarded 3 marks if the accuracy exceeds 75%, 2 marks if the accuracy exceeds 73%, 1 mark if the accuracy exceeds 71%, and 0 marks otherwise.

Task 2 : AdaBoost (3 marks)

In this task you have to implement the 2-class AdaBoost classifier with perceptron as the weak learner. Fill out code in the `train()` and `classify()` function at the location indicated in `boosting.py`.

`train()` should contain the code to compute the weights for each data point after training the weak classifier. `classify()` should contain the code to get the labels from individual classifiers and then have weighted summation to get the final output.

Run your code with this command:

```
python dataClassifier.py -c boosting -t 1000 -s 1000 -b 20
python autograder.py -t 3
```

Your classifier will be evaluated for its accuracy on the test set after the AdaBoost algorithm has been run with weak learners trained for the default 3 iterations. You will be awarded 3 marks if the accuracy exceeds 75%, 2 marks if the accuracy exceeds 73%, 1 mark if the accuracy exceeds 71%, and 0 marks otherwise.