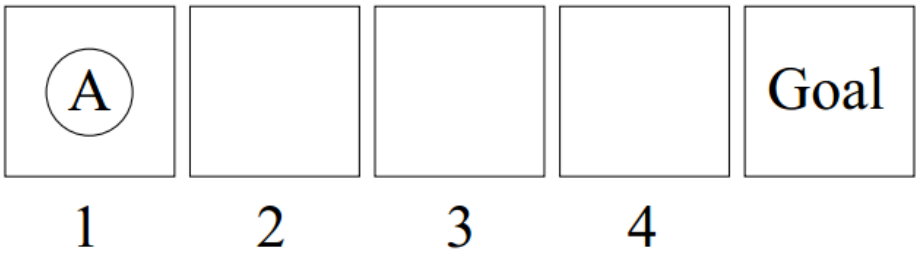
FIT 3080: Assignment 3 (15%)

Expectimax, MDP & RL, Machine Learning

Due: October 21st (Friday), 3pm

(You are encouraged to work in groups of two for this assignment)

1. [MDP&RL] Soccer [6*8 marks]. A soccer robot A is on a fast break toward the goal, starting in position 1. From positions 1 through 3, it can either shoot (S) or dribble the ball forward (D). From 4 it can only shoot. If it shoots, it either scores a goal (state G) or misses (state M). If it dribbles, it either advances a square or loses the ball, ending up in M. When shooting, the robot is more likely to score a goal from states closer to the goal; when dribbling, the likelihood of missing is independent of the current state.



In this MDP, the states are 1,2,3,4, G and M, where G and M are terminal states. The transition model depends on the parameter y, which is the probability of dribbling success. Assume a discount of γ=1.

$$
\begin{aligned}
T(k,S,G) &= \frac{k}{6} \\
T(k,S,M) &= 1-\frac{k}{6} \\
T(k,D,k+1) &= y \quad \text{for } k \in \{1,2,3\} \\
T(k,D,M) &= 1-y \quad \text{for } k \in \{1,2,3\} \\
R(k,S,G) &= 1
\end{aligned}
$$

Rewards are 0 for all other transitions.

a. What is $V^{\pi}(1)$ for the policy π that always shoots?

b. What is Q*(3,D) in terms of y?

c. Using y =3/4, complete the first two iterations of value iteration.

d. After how many iterations will value iteration compute the optimal values for all states?

e. For what range of values of y is Q*(3,S) >= Q*(3,D)?

f. Now consider Q-learning, in which we do not have a model (T, y and k above), and instead learn from a series of experienced transitions. Using a learning rate of $\alpha=1/2$ execute Q-learning on these episodes:

- 1-D, 2-D, 3-D, 4-S, G
- 1-D, 2-D, 3-D, 4-S, M
- 1-D, 2-D, 3-S, G
- 1-D, 2-S, M
- 1-D, 2-D, 3-D, M

2. [Machine Learning] Implementation of Decision Trees [2*26 Marks].

(a) Implement decision tree learning as described in Section 18.3 of the textbook. Your implementation should work for data sets with binary features and binary labels (it can work for other kinds of features and labels too, but we only require that it works for binary features and binary labels). Implement the recursive training procedure given in Figure 18.5 with two extensions:

- Your code should take in a parameter depth which specifies the maximum height of the tree. If depth is 0 then the tree produced should be a single node. If depth is 1 then the tree should be at most a node whose children are leaf nodes (this is sometimes called a decision stump). More generally, if depth is n then there should be at most n non-leaf nodes along any path from the root to a leaf node.

- Your code should also take in a parameter splitting Rule which is either INFO GAIN or RANDOM. If this value is set to INFO GAIN then your training procedure should use the information gain heuristic described in the book to pick which feature to use at each node. If this value is set to RANDOM then it should use a randomly selected feature at each node.

Also implement a procedure for predicting the label of an unlabeled test example.

To help you out, we've provided some skeleton python code on Moodle in a3.zip file. Here are some implementation hints:

- When calculating the entropy of a Boolean random variable, make sure to correctly handle the cases where p = 0 and p = 1.

- Similarly, watch out for division by zero when computing probabilities

- Stick to binary valued features. This will simplify your code.

- If you're getting strange results, try printing out your information gain values. These should all be between 0 and 1.

(b) Test your implementation on the data set provided in a3.zip file. In this zip file you'll find a file "train.txt" and "test.txt" containing training and test data respectively. Each file is in CSV (comma separated value) format. The first 38 values on each line are binary features and the last value on each line is the binary label. The task in this data set is to predict the winner in a chess end game involving King and Rook versus King and Pawn. The features are derived from the chess positions and the label is 1 if white can win and 0 if white cannot win. There are 2109 examples in the training set and 1087 examples in the test set (about a 2/3 train/test split).

Write a brief (no more than 1 page) discussion of your results. This write-up should include one or more plots showing the training and test set accuracy on the provided data set for all values of depth between 0 and 30 and for both values of splittingRule. When computing accuracy values for splittingRule = RANDOM, average over 100 runs. A simple way to plot all this would be as a line graph with depth on the x axis and accuracy on the y axis. You can then include a training accuracy and test accuracy line for both values of splittingRule (4 lines total). Also discuss in your write-up:

- Which values of depth and splittingRule gave the best test set accuracy and why.
- Which values of depth and splittingRule gave the best training set accuracy and why.
- Which values of depth and splittingRule gave the largest difference between training and test set accuracy and why.

Your program for Question 4 must be executable using the following command-line:

python question2.py <train_file> [I or R] <depth> <test_file> <output_file>
or
java -jar question2.jar <train_file> [I or R] <depth> <test_file> <output_file>

where <train_file> is the name of the input training file, [I or R] is the splitting method which is either Information gain or Random (denoted by I or R respectively), <depth> is the maximum depth allowed for the decision tree, <test_file> is the name of the test file , and <output_file> is the name of the output file into which the accuracy of the learned model on the test set is written into a line. If the output file already contains some lines, the accuracy must be appended as the last line of the file.

For example, consider the following command-line:

python question2.py train.txt R 5 test.txt output.txt
or
java -jar question2.jar train.txt R 5 test.txt output.txt

It learns a decision tree of max depth 5 using the Random splitting based on the training data in train.txt, and then writes the accuracy of the learned model on the test.txt into the output file output.tst.

Submission Requirements

- You are allowed to work with **one** other student if you wish. Beyond that, make sure you haven't shared your work with other students.

- Submit on Moodle a zip file that contains (1) A PDF file containing your solutions to the questions, and (ii) jar or python files containing your implementation for Question 2. The zip file should be named A3StudentID.zip, where StudentID is your Student ID number. If you have done the work in pairs, use only the ID of one student in your submission, but put both names in the PDF report. In addition, both students must complete the electronic Assignment Cover Sheet.

- Hand in the report in hard copy as well, in the Assignment Box as it will be advertised.

- **You can only use Java or Python as the programming language.** Please use the skeleton code that we have provided in these languages for you on Moodle for this assignment.

- Your Python/Java file must be executable on **lab computers** by the command-line mentioned in the question 2. *If your program does not run smoothly on a computer lab with the command-line mentioned in Question 2, then you may get zero mark for that implementation question*. We emphasize the you need to use **exactly** the name "question2.py" (if you use Python) or "question2.jar" (if you use Java) for your python program.

- **Late submission policy:** 10% of the maximum mark will be deducted for every day a submission is late.