Probabilistic Reasoning

Seyed Mahdi Basiri Azad (Erfan)

March 1, 2016

1 Overview

In this project we used probabilistic reasoning methods to find the probability distribution of an agent who moves in cardinal directions in a 4 by 4 maze similar to the maze in project 2. I used the Maze.java file from project 2 to read in a maze from a text file. The agent in this project is different from the agent from project 2 because it is not "blind" anymore instead it just have a slightly "bad eyesight"! The maze floor is colored such that each square has its own color from a color list of **red, green, blue and yellow**. The robot has a sensor directly looking down that %88 of the time it reads the correct color but there is also a %4 chance for other colors to be read instead. The task is to use probabilistic reasoning to correctly guess the location of the robot given its sensor reading. The robot moves randomly (although in the cardinal directions) in the maze, one square at a time.

2 Implementation

I broke the problem down by creating classes for ColorSensor, Agent, MarkovModel and ReaseningProblem. There is also the Maze.java that is used for reading in the maze from a textfile.

1. ColorSensor.java

   This is a simple class that represents the sensor. The main method for a sensor is to read the color of the floor. It will return the correct color %88 of the time!

2. Agent.java

   This class represents the agent itself. The agent is aware of the outline of the maze. I used a HashMap of location and color to represent the maze. It has a ColorSensor and has a memory! It can memorize the color of squares that is has (either correctly of incorrectly) seen so far. I used a LinkedList to represent the agent's memory.

3. MarkovModel.java

   This class represents the MarkovModel and is used to calculate the transition probability distribution and event the probability distribution of being in each square using the result of the ColorSensor.
   It uses the transition-prob method to calculate the probability of being in a location by checking to see from how many adjacent squares could it have possibly moved to its current location and multiplies the probability of the adjacent squares by the probability of it moving from them with is uniform for each of the four directions(0.25). Below is a snippet of how the code works:

```java
private double transition_prob(Pair<Integer, Integer> loc){
   Pair<Integer, Integer> NORTH = new Pair<>(loc.getKey(), loc.getValue()+1);
   Pair<Integer, Integer> SOUTH = new Pair<>(loc.getKey(), loc.getValue()-1);
   Pair<Integer, Integer> EAST = new Pair<>(loc.getKey()+1, loc.getValue());
   Pair<Integer, Integer> WEST = new Pair<>(loc.getKey()-1, loc.getValue());
   double transProb = 0.0; //transition probability
        //if the square to the north of current location can be reached
   if(mazeWorld.containsKey(NORTH)){
   //increment according to the probability of north
  transProb += 0.25 * probabilityMap.get(NORTH);
   }else{
   //increment according the probability of being in the same place
  transProb += 0.25 * probabilityMap.get(loc);
   }
   ...
```

   There are also **observation-prob()** method which returns the confidence level of the sensor reading.

```java
  private double observation_prob(Color c, Pair<Integer, Integer> loc){
  if(mazeWorld.get(loc) == c){ //if the color of square at loc is indeed c
  return 0.88; //will give the correct result with 0.88 confidence!
  }else{
  return 0.04;
  }
  }
```

Next is the **updateLocProb()** method which returns a double representing the updated probability for the current location by multiplying the probability of being in the current location and confidence level of the current observation. Basically it follows the predict-update scheme that we have been discussing in class.

```
protected double UpdateLocProb(Pair<Integer, Integer> loc, List<Color> memory){
        //using only the last color atm
  return transition_prob(loc) * observation_prob(memory.get(0), loc);
}
```

Here i am using a first order MarkovChain model and hence only using the most recent observation. I am using a LinkedList so it will be easier to compute higher order MarkovChain models in the future. Increasing the Order of the Markov model will increase the accuracy of the probability distribution calculated for each location.

4. ReasoningProblem

   This is the driver for the project. It initializes the probability map to a uniform distribution in the beginning. The main method here is the **RunReasoningProblem()** method. In this method we move the agent toward a random direction first then we update the probability map and search for the highest location in the probability map. If the square with highest probability is in fact the square that the agent is in then the agent has guessed its location correctly; otherwise the probability distribution is wrong and the agent has guessed incorrectly.

```
private void runReasoningProblem(){
  Color currentColor = agent.getLocationColor();//see where you are
  agent.memorize(currentColor);//remember the color of current square
  updateProbMap();//update the probability map of the whole maze
  System.out.println("Starting...");
  System.out.println("Agent is at: " + agent.location);
  System.out.println("Agent sees: " + currentColor + " actual color is: "
  + mazeWorld.get(agent.location));
  //print the maze

  int numIterations = 100;
  for (int k=0; k<numIterations; k++){
    //make a random move
    Pair<Integer, Integer> move = MOVES.get(rand.nextInt(4));
    System.out.println(move);
    agent.move(move);
    agent.memorize(agent.getLocationColor());//read and memorize the color
    updateProbMap();
    Pair<Integer, Integer> guessedLoc = getGuessedLoc();
    System.out.println("Guessed location: " + guessedLoc +
    " actual location: " + agent.location);
  }
}
```

Unfortunately I did not get a chance to print the maze and So I have not tested the correctness of the algorithm at this point.

3 Improvements

The main improvement to be made here is to increase the order of the Markov chain. The current code only updates the prediction based on one evidence which is the color of the floor from the last time that a sensor reading was done!