Lab 7

Due: Tuesday 04/09/2024 @ 11:59pm EST

The purpose of labs is to practice the concepts that we learn in class. To that end you will be writing java code that uses a game engine called Sepia to develop agents that solve specific problems. In this lab we will be playing a zombie survival game called Zombayes, where we have to classify between human units (pretending to be zombies so they don't get eaten) and actual zombies. Zombies will attack you given the chance, but humans will not. Your job will be to use a decision tree to learn how to distinguish between zombies and humans: you want to attack the zombies but not the humans!

1. Copy Files

Please, copy the files from the downloaded lab directory to your cs440 directory. You can just drag and drop them in your file explorer.

- Copy `Downloads/lab7/lib/zombayes.jar` to `cs440/lib/zombayes.jar`.
  This file is the custom jarfile that I created for you.
- Copy `Downloads/lab7/data/labs/zombayes` to `cs440/data/labs/zombayes`.
  This directory contains a game configuration and map files.
- Copy `Downloads/lab7/src/labs` to `cs440/src/labs`.
  This directory contains our source code `.java` files.
- Copy `Downloads/lab7/zombayes.srcs` to `cs440/zombayes.srcs`.
  This file contains the paths to the `.java` files we are working with in this lab. Just like last lab, files like these are used to speed up the compilation process by preventing you from listing all source files you want to compile manually.
- Copy `Downloads/lab7/doc/labs` to `cs440/doc/labs`. This is the documentation generated from `zombayes.jar` and will be extremely useful in this assignment. After copying, if you double-click on `cs440/doc/labs/zombayes/index.html`, the documentation should open in your browser.

2. Test run

If your setup is correct, you should be able to compile and execute the given template code. You should see the Sepia window appear.

```
# Mac, Linux. Run from the cs440 directory.
javac -cp "./lib/*:." @zombayes.srcs
java -cp "./lib/*:." edu.cwru.sepia.Main2 data/labs/zombayes/easy/tunnel.xml

# Windows. Run from the cs440 directory.
javac -cp "./lib/*;." @zombayes.srcs
java -cp "./lib/*;." edu.cwru.sepia.Main2 data/labs/zombayes/easy/tunnel.xml
```

3. Survival Rules & Information

Our game has two phases. Initially, you and a partner must go and collect some gold. However, the gold is guarded by a horde of zombies with humans pretending to be zombies (so they don't get eaten) mixed in. There is also a special zombie unit who will continuously spawn new units whenever another unit is killed (this special zombie can spawn other zombies and also other humans, dont ask why). Because of this scenario, you will lose this game. However, we are interested in how many zombies you can take down with you before you are killed (and also how many humans you choose to kill as well).

The game will start off with your partner going to collect gold. Once your partner is adjacent to the gold, all units in the horde will rush them. Humans, who bear no ill will to you or your partner, will take advantage of the confusion and harmlessly pass your partner (and you) by while seeking to escape from the horde. The zombies however will seek to kill your partner (and you when it is your turn). Your partner has a lot of health but isn't very smart (or capable with their weapon), so will eventually succumb to the horde.

You have been paying attention this whole time, and will be supplied at the moment of your partner's death with a record of which units attacked them (and therefore are known to be zombies), and which units did not (and therefore are known to be humans). Each unit has a feature description which you cannot see on the game rendering. So, to make it easier to view, human units in the horde have a "h" character, and zombie units have a "z" character. The "S" character is the zombie spawning unit, and there is only one of them in the game.

Your agent however does not get to view the characters of the units in order to make its decisions. Instead, your agent is supplied with four features: 2 continuous and 2 discrete. The distribution of the features is controlled by how difficult the game is: humans and zombies look very different in the EASY game mode and look much more similar in the HARD game mode. You are to implement and train a decision tree model to classify whether or not a unit (represented with its feature vector) is a human or not. Any unit your decision tree classifies as a zombie (i.e. class 1) will be attacked by your agent. So be careful! Any time you waste shooting humans is time that you aren't shooting zombies!

Task 1: Naive Bayes (100 points)

In this task, I want you to fill in the empty methods and classes in `src.labs.zombayes.NaiveBayesAgent`. You will need to finish the implementation for a `NaiveBayes` type, specifically the `fit` and `predict` method. The `fit` method should train a Naive Bayes model on the provided training data, and `predict` should make a prediction using that trained model on the provided test point. I have hidden most of the Sepia-ness from you, so please focus entirely on making a good naive bayes model!

Task 2: Extra Credit (100 points)

In this task, I want you to fill in the empty methods and classes in `src.labs.zombayes.DecisionTreeAgent`. In general, Decision Trees are much harder to implement than Naive Bayes, as there is more software engineering that needs to go into it. I have already started this implementation for you.
Our Decision Tree implementation will be a standard tree implementation, using `Node` objects and children/parent relationships. I have provided an abstract `Node` datatype, and have begun the implementation for two subtypes: `InteriorNode` and `LeafNode`. You will need to finish these two classes in order to begin assembling a Decision Tree.
To build a Decision Tree, I have begun a recursive dfs-style implementation for you called `dfsBuild`. This implementation should build a `Node` from the provided data (only looking at the provided columns within that data), and should recurse to build all children of the `Node` before returning.
I have provided the implementation for `fit` and `predict` for you as a courtesy. Most notably is the `predict` method which will walk your tree until it encounters a `LeafNode`, at which point it will make a prediction. I have hidden most of the Sepia-ness from you, so please focus entirely on making a good Decision Tree model!

Task 3: Submitting your lab

Please submit `NaiveBayesAgent.java` on gradescope (just drag and drop in the file). If you complete the extra credit, please also turn in `DecisionTreeAgent.java`.