CS B551 - Assignment 2: Games and Bayesian Classifiers

Fall 2022
Due: Monday November 7 , 11:59:59PM Eastern (New York) time
Late deadline: Wednesday November 9, 11:59:59PM Eastern time, with a 10% penalty

This assignment will give you practice with game playing and probability. Please read the instructions below carefully; we cannot accept any submissions that do not follow the instructions given here. Most importantly: please **start early,** and ask questions on Q & A Community or in office hours.

Guidelines for this assignment

Coding requirements. For fairness and efficiency, we use a semi-automatic program to grade your submissions. This means you must write your code carefully so that our program can run your code and understand its output properly. In particular: 1. You must code this assignment in Python 3. 2. Make sure to use the program file name we specify. 3. Use the skeleton code we provide, and follow the instructions in the skeleton code (e.g., to not change the parameters of some functions). 4. You may import standard Python modules for routines not related to AI, such as basic sorting algorithms and data structures like queues, as long as they are already installed on the SICE Linux servers. 5. **IMPORTANT:** In addition to testing your programs on your own, test your code on silo.sice.indiana.edu using the provided test scripts.

For each of these problems, you will face some design decisions along the way. Your primary goal is to write clear code that finds the correct solution in a reasonable amount of time. To do this, you should give careful thought to the search abstractions, data structures, algorithms, heuristic functions, etc. To encourage innovation, we will conduct a competition to see which program can solve the hardest problems in the shortest time, and a small amount of your grade (or extra credit) may be based on your program's performance in this competition.

Groups. You'll work in a group of 1-3 people for this assignment according to your preferences. You should only submit **one** copy of the assignment for your team, through GitHub. All the people on the team will receive the same grade, except in unusual circumstances; we will collect feedback about how well your team functioned in order to detect these circumstances.

Coding style and documentation. We will not explicitly grade based on coding style, but it's important that you code in a way that we can easily understand. Please use descriptive variable and function names, and use comments when needed to help us understand the code that is not obvious.

Report. Please put a report describing your assignment in the README.md file in your GitHub repository. For each problem, please include: (1) a description of how you formulated each problem; (2) a brief description of how your program works; (3) and discussion of any problems you faced, any assumptions, simplifications, and/or design decisions you made. These comments are especially important if your code does not work as well as you would like, since it is a chance to document how much energy and thought you put into your solution.

Academic integrity. We take academic integrity very seriously. To maintain fairness to all students in the class and integrity of our grading system, we will prosecute any academic integrity violations that we discover. *Before beginning this assignment, make sure you are familiar with the Academic Integrity policy of the course, as stated in the Syllabus, and ask us about any doubts or questions you may have.* To briefly summarize, you may discuss the assignment with other people at a high level, e.g. discussing general strategies to solve the problem, talking about Python syntax and features, etc. You may also consult printed and/or

online references, including books, tutorials, etc., but you must cite these materials (e.g. in source code comments). We expect that you'll write your own code and not copy anything from anyone else, including online resources. *However, if you do copy something (e.g., a small bit of code that you think is particularly clever), you have to make it explicitly clear which parts were copied and which parts were your own. You can do this by putting a very detailed comment in your code, marking the line above which the copying began, and the line below which the copying ended, and a reference to the source.* Any code that is not marked in this way must be your own, which you personally designed and wrote.

Part 0: Getting started

For this project, you can find your team arrangement by logging into IU Github at `https://github.iu.edu/cs-b551-fa2022/` and navigating to the repository a2-release-res. Please fill out the form for team selection on https://forms.gle/DozNCas8FGYcGtYd6 by Wednesday, October 19 at 23:59 (if you do not fill out the form we will assign you to a team randomly). We will publish final team names by Friday, October 21 in the following format: `userid1-a2`, `userid1-userid2-a2`, or `userid1-userid2-userid3-a2`, where the other user ID(s) correspond to your teammate(s). You should submit your code in a repository with the same name as your team name. If you don't already know your team mates, you can write them on `userid@iu.edu`.

To get started, clone the github repository:

```
git clone git@github.iu.edu:cs-b551-fa2022/a2-release-res
```

If that doesn't work, instead try:

```
git clone https://github.iu.edu/cs-b551-fa2022/a2-release-res
```

(If neither command works, you probably need to set up IU GitHub ssh keys. See Canvas for help.)

Part 1: Raichu

Raichu is a popular childhood game played on an n × n grid (where n ≥ 8 is an even number) with three kinds of pieces (Pichus, Pikachus, and Raichus) of two different colors (black and white). Initially the board starts empty, except for a row of white Pikachus on the second row of the board, a row of white Pichus on the third row of the board, and a row of black Pichus on row n − 2 and a row of black Pikachus on row n − 1:

```
   1 2 3 4 5 6 7 8
1  . . . . . . . .
2  W . W . W . W .
3  . w . w . w . w
4  . . . . . . . .
5  . . . . . . . .
6  b . b . b . b .
7  . B . B . B . B
8  . . . . . . . .
```

Two players alternate turns, with White going first.

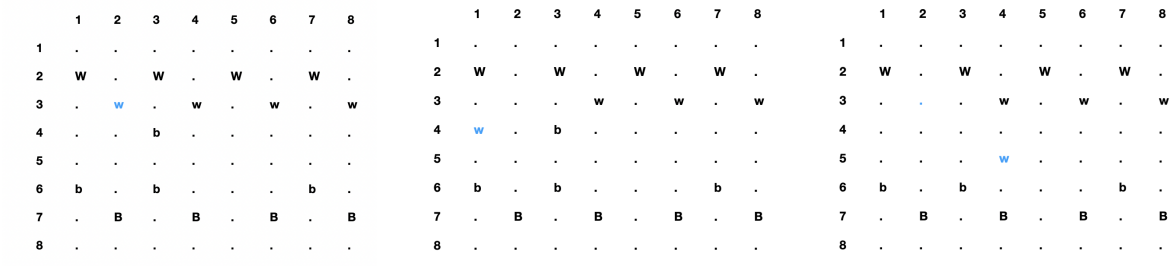
In any given turn, a player can choose a single piece of their color and move it according to the rules of that piece.

A Pichu can move in one of two ways:

- one square forward diagonally, if that square is empty.

- "jump" over a single Pichu of the opposite color by moving two squares forward diagonally, if that square is empty. The jumped piece is removed from the board as soon as it is jumped.

For example, for the highlighted Pichu in the following board at left, there are two possible moves, shown in the right two boards:

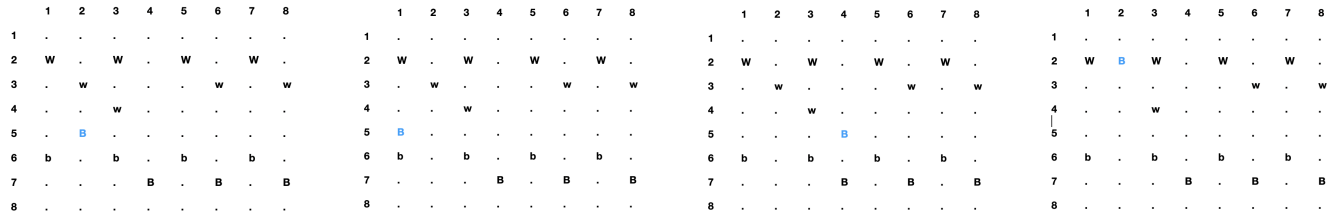


Initial board | Possible move #1 | Possible move #2

A Pikachu can move in one of two ways:

- 1 or 2 squares either forward, left, or right (but **not diagonally**) to an empty square, as long as all squares in between are also empty.
- "jump" over a single **Pichu/Pikachu** of the opposite color by moving 2 or 3 squares forward, left or right (**not diagonally**), as long as all of the squares between the Pikachu's start position and jumped piece are empty and all the squares between the jumped piece and the ending position are empty. The jumped piece is removed as soon as it is jumped.

For example, for the highlighted Pikachu in the following board at left, here are some (not all) possible moves:

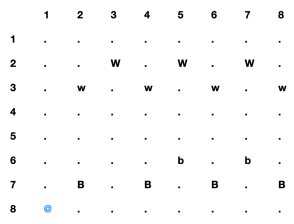


Initial board | Possible move #1 | Possible move #2 | Possible move #3

A Raichu is created when a Pichu or Pikachu reaches the opposite side of the board (i.e. when a Black Pichu or Pikachu reaches row 1 or a white Pichu or Pikachu reaches row n). When this happens, the Pichu or Pikachu is removed from the board and subsituted with a Raichu. Raichus can move as follows:

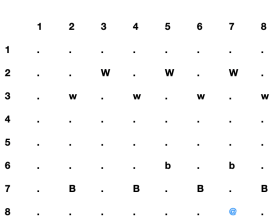
- any number of squares forward/backward, left, right or diagonally, to an empty square, as long as all squares in between are also empty.
- "jump" over a single **Pichu/Pikachu/Raichu** of the opposite color and landing any number of squares forward/backward, left, right or diagonally, as long as all of the squares between the Raichu's start position and jumped piece are empty and all the squares between the jumped piece and the ending position are empty. The jumped piece is removed as soon as it is jumped.

For example, some (not all) possible moves of the highlighted white Raichu are:

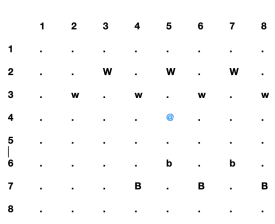


Initial board

Possible move #1



Possible move #2



Possible move #3

Note the hierarchy: Pichus can only capture Pichus, Pikachus can capture Pichus or Pikachus, while Raichus can capture any piece. The winner is the player who first captures all of the other player's pieces.

Your task is to write a Python program that plays Raichu well. Your program should accept a command line argument that gives the current state of the board as a string of `.`'s, `w`'s, `W`'s, `b`'s, `B`'s, `@`'s, and `$`'s, which indicate which squares have no piece, a white Pichu, a white Pikachu, a black Pichu, a black Pikachu, a white Raichu and a black Raichu respectively, in row-major order. For example, if $n = 8$, then the encoding of the start state of the game would be:

```
........W.W.W.W..w.w.w.w................b.b.b.b..B.B.B.B........
```

More precisely, your program will be called with four command line parameters: (1) the value of n, (2) the current player (w or b), (3) the state of the board, encoded as above, and (4) a time limit in seconds. Your program should then decide a recommended single move for the given player with the given current board state, and display the new state of the board after making that move. Displaying multiple lines of output is fine as long as the last line has the recommended board state. The time limit is the amount of time that your program should expect to have to make its decision; our testing code will kill your program at that point, and will use whichever was the last move your program recommended. For example, a sample run of your program might look like:

```
[djcran@macbook]$ python3 ./raichu.py 8 w '........W.W.W.W..w.w.w.w................b.b.b.b..B.B.B.B........' 10
Searching for best move for w from board state:
........
W.W.W.W.
.w.w.w.w
........
........
b.b.b.b.
.B.B.B.B
........
Here's what I decided:
..........W.W.W..w.w.w.wW..............b.b.b.b..B.B.B.B........
```

Your program should work for any reasonable value of n and reasonable time limit, although we plan to test it only for values of n close to 8 and time limits around 10-30 seconds.

Hint: Since our grading program only looks at the last solution that you output, your program can output multiple solutions. In other words, you can choose to completely ignore the time parameter passed to your program and simply output multiple answers until you run out of time and we automatically kill your program.

Note: Your code must conform with the interface standards mentioned above! The last line of the output *must* be the new board in the format given, without any extra characters or empty lines. Also, note that your program cannot assume that the game will be run in sequence from start to end; given a current board position on the command line, your code must find a recommended next best move. Your program can write files to disk to preserve state between runs (although this is not required or necessarily recommended), but should correctly handle the case when a new board state is presented to your program that is unrelated to the last state it saw.

Part 2: Truth be Told

Many practical problems involve classifying textual objects — documents, emails, sentences, tweets, etc. —into two specific categories — spam vs nonspam, important vs unimportant, acceptable vs inappropriate, etc. Naive Bayes classifiers are often used for such problems. They often use a bag-of-words model, which means that each object is represented as just an unordered "bag" of words, with no information about the grammatical structure or order of words in the document. Suppose there are classes A and B. For a given textual object D consisting of words w_1, w_2, ..., w_n, a Bayesian classifier evaluates decides that D belongs to A by computing the "odds" and comparing to a threshold,

$$\frac{P(A|w_1, w_2, ..., w_n)}{P(B|w_1, w_2, ..., w_n)} > 1,$$

where $P(A|w_1, ...w_n)$ is the posterior probability that D is in class A. Using the Naive Bayes assumption, the odds ratio can be factored into $P(A)$, $P(B)$, and terms of the form $P(w_i|A)$ and $P(w_i|B)$. These are the parameters of the Naive Bayes model.

As a specific use case for this assignment, we've given you a dataset of user-generated reviews. User-generated reviews are transforming competition in the hospitality industry, because they are valuable for both the guest and the hotel owner. For the potential guest, it's a valuable resource during the search for an overnight stay. For the hotelier, it's a way to increase visibility and improve customer contact. So it really affects both the business and guest if people fake the reviews and try to either defame a good hotel or promote a bad one. Your task is to classify reviews into faked or legitimate, for 20 hotels in Chicago.

We've provided skeleton code and a training and testing dataset to get you started. You can run it like this:

```
python3 ./SeekTruth.py deceptive.train.txt deceptive.test.txt
```

The code currently does not do anything interesting. Your job is to write a program that estimates the Naive Bayes parameters from training data (where the correct label is given), and then uses these parameters to classify the reviews in the testing data.

Hints: Don't worry, at least at first, about whether the "words" in your model are actually words. Just treat every unique space-delimited token you encounter as a "word," even if it's misspelled, a number, a punctuation mark, etc. It may be helpful to ignore tokens that do not occur more than a handful of times, however.

What to turn in

Create a private repository using the team name as described in Part 0, and keep the same directory structure as in `a2-release-res` (i.e., separating the programs into Part1 and Part2). Make sure that you stick to this naming scheme and directory structure so that our autograder can locate your submission correctly. And make sure it is private (not internal or public) so others will not see your submission. Turn in the three programs on GitHub (remember to add, commit, push) — we'll grade whatever version you've put there as of 11:59:59PM on the due date. To make sure that the latest version of your work has been accepted by GitHub, you can log into the github.iu.edu website and browse the code online.