Colorado CSCI 5454: Algorithms
Homework 10

Instructor: Bo Waggoner
Due: Sunday, December 8, 2019 at 11:59pm
Turn in electronically via Gradescope.

Remember to list the people you worked with and any outside sources used (if none, write "none").

Problem 1 (6 points)

Part a (2 points) Recall that the Polynomial Weights Algorithm has a parameter ϵ chosen by the algorithm designer in advance. To prove the average regret bound of $O\left(\frac{1}{\sqrt{T}}\right)$, we set $\epsilon = \Theta\left(\frac{1}{\sqrt{T}}\right)$. What if we were to set ϵ much smaller, say, $\Theta\left(\frac{1}{T}\right)$? Explain why this could cause poor performance intuitively. Give an example instance where the algorithm has large regret.

Part b (2 points) What if we were to set ϵ much larger, say, $\Theta(1)$? (For example, $\epsilon = 0.1$.) Explain why this could cause poor performance intuitively. Give an example instance where the algorithm has large regret.

Part c (2 points) Suppose you were running the polynomial weights algorithm on the following kind of instance: One of the actions always has a loss of zero, while all other actions have high losses, say always between 0.5 and 1. What would you recommend as the choice of ϵ for this kind of instance? Why?

Problem 2 (18 points)

The dwarves of Moria have an ancient game called "Rock, Paper, Scissors, Mithril". It is a zero-sum game where each player simultaneously chooses one of the four options. **In this problem, you will use Polynomial Weights to find an equilibrium strategy in the game.** Attach all source code printed to PDF.

Below is the utility function u. The entries of the matrix give the utility of the *row* player. Note this game is *symmetric*: the set of actions available to both players is the same, and the payoffs are mirrored if they swap actions.

	Rock	Paper	Scissors	Mithril
Rock	0	-1	1	-0.5
Paper	1	0	-1	0.5
Scissors	-1	1	0	-0.5
Mithril	0.5	-0.5	0.5	0

Here is the main function you will implement:

Algorithm 1 Compute-Equilibrium(T, ϵ)

Input: Parameters T, ϵ
Initialize `weights` $\leftarrow [1, 1, 1, 1]$
Initialize `list_of_probs`, an empty list
for $t = 1, \ldots, T$ **do**
  Set `probs` $\leftarrow$ `probs_from_weights`(`weights`)
  Add `probs` to `list_of_probs`
  Set `response` $\leftarrow$ `best_response`(`probs`)
  Set `utilities` $\leftarrow$ `util_of_each_action`(`response`)
  Set `losses` $\leftarrow$ `convert_utils_to_losses`(`utilities`)
  Set `weights` $\leftarrow$ `update_weights`(`weights, losses,` ϵ)
end for
Return average of `list_of_probs`

Note the variables are (and this will be developed in the subproblems below):

- `weights` is an array of weights on the four actions Rock, Paper, Scissors, Mithril
- `probs` is an array of probabilities on the four actions, e.g. $[0.25, 0.25, 0.25, 0.25]$
- `list_of_probs` is a list, where each element is of the form `probs`. At the end, we average the probability placed on the actions and return that array of average probabilities as the equilibrium strategy for the row player.
- `response` indicates one of the four actions as a best-response to the current mixed strategy `probs`.
- `utilities` is an array of four utilities, the utility for playing each of the four actions against `response`.
- `losses` is an array of four losses corresponding to the utilities (see subproblem below).

Part a (2 points) Implement `probs_from_weights`. This should take in an array of weights on each action, as maintained by the Polynomial Weights algorithm, and output the corresponding probability distribution over actions.

Part b (2 points) Implement `best_response`. This should take in a mixed strategy of the row player (i.e. a probability distribution on actions) and output a single action for the column player. For example, if the input is $(0.5, 0.5, 0, 0)$, which represents a 50-50 mix between Rock and Paper, your function would compute the expected payoff for Rock (-0.5), Paper (0.5), Scissors (0), and Mithril (0) and conclude that the best-response action is Paper.

Hint: A good approach is to index the actions, for example, $0, 1, 2, 3$ *in a zero-indexed language. Then create an array* `utility_matrix` *where* `utility_matrix[i][j]` *is the utility for playing action* i *against action* j*. Then make a function that computes the expected utility for the column player to play action* i *against distribution* `probs`*. Finally, implement* `best_response` *by returning the action* i *that maximizes utility.*

Part c (2 points) Implement `util_of_each_action`. This returns an array with the utility we would have gotten for playing action $0, 1, 2$, or 3, given that the opponent played `response`.

Part d (2 points) Implement `convert_utils_to_losses`. Remember that Polynomial Weights needs to know, at each round, its *loss* if it had chosen each action. It also assumes all losses are between zero and one.

So this function must take a list of utilities (which are between -1 and 1) and convert each utility to a loss. Higher utility must be mapped to *smaller* loss. Specifically, if an action got utility 1, this is the best possible, so this should be mapped to a loss of zero. Similarly, if an action got utility -1, this is the worst possible, so this should be mapped to a loss of one.

Part e (2 points) Implement `update_weights`. This should use the Polynomial Weights update rule to update the weights of the algorithm according to the losses at that round.[1]

Part f (4 points) Implement the main learning algorithm, Compute-Equilibrium, using the above subroutines.

For each of the following T, decide which ϵ to use (tell us your choice and why), run Compute-Equilibrium, and report the output. Notice that the output is the *average* of the probabilities played over all the rounds, which our theory tells us is close to an equilibrium strategy.

- $T = 10$
- $T = 100$
- $T = 1000$
- $T = 10000$

[1]The weights may become very small over the course of many rounds of the algorithm. However, notice that the algorithm does not change if we scale all the weights by the same factor. To address this, you can optionally modify this function to scale all the weights so the largest is equal to one.

- $T = 100000$
- beyond? (optional)

Part g (2 points) For each experiment in the previous problem, let us see how close to equilibrium is the strategy we've computed. Compute the opponent's best response to the mixed strategy output by your algorithm, and report the opponent's expected utility in each case. (As $T \to \infty$, we should converge to the optimal equilibrium strategy, which minimizes the opponent's expected utility.)

Part h (2 points) For some $T \geq 10000$, run your algorithm and create the following plot. The horizontal axis is round $t = 1, \ldots, T$, and the vertical axis is probability. Add four curves on the figure, one for each action. The first curve should plot `probs[Rock]` at each round. The second should plot `probs[Paper]`, and so on.

In a sentence, what is the plot showing? (In terms of the algorithm learning over time.)

Now re-run the algorithm with ϵ larger (say two to five times as large) and generate the plot. Finally, do so with ϵ smaller by the same factor.

Describe what you see in the plots and explain why changing ϵ would have these effects.