Tania Soutonglang

CS 581 Spring 2024 Programming Assignment #02

Due: **Sunday, April 7, 2024, 11:59 PM CST**
Points: **100**

Instructions:

1. Place **all your deliverables (as described below) into a single ZIP** file named:

`LastName_FirstName_CS581_Programming02.zip`

2. Submit it to Blackboard Assignments section before the due date. **No late submissions will be accepted**.

Objectives:

1. (100 points) Implement and evaluate a K-Armed Bandit algorithm.

Input data file:

Your input file will be a CSV (comma separated values) file (see Programming Assignment #02 folder in Blackboard – `input.csv`).

You **CANNOT** modify nor rename input data files.

First row of that file is always going to contain column labels.

Each column (starting at row 2) in that input file will represent time series data for some system (sample file data represents traces of Galvin Library WiFi channel occupancy – there are 11 channels/columns with numerical values representing signal strength in dBm).

Deliverables:

Your submission should include:

- Python code file(s). Your py source code file should be named:

  `cs581_P02_AXXXXXXXX.py`

  where `AXXXXXXXX` is your IIT A number (this is REQUIRED!). If your solution uses multiple files, makes sure that the main (the one that will be run to solve the problem) is named that way and others include your IIT A number in their names as well.

- this document with your results and conclusions. You should rename it to:

  `LastName_FirstName_CS581_Programming02.doc or .pdf`

Problem description:

In probability theory and machine learning, the multi-armed bandit problem (sometimes called the K- or N-armed bandit problem) is a problem in which a decision maker iteratively selects one of multiple fixed choices (i.e. arms or actions) when the properties of each choice are only partially known at the time of allocation, and may become better understood as time passes. A fundamental aspect of bandit problems is that choosing an arm does not affect the properties of the arm or other arms [Wikipedia].

Your task is to implement the **epsilon-greedy variant of the K-Armed Bandit** problem and use it to learn the probabilities of success using provided input (data)

Your program should:

- Accept four (4) command line arguments corresponding to two states / state capitals (initial and goal states) so your code could be executed with

```
python cs581_P02_AXXXXXXXX.py FILENAME EPSILON TRAIN% THR
```

  where:
  - `cs581_P02_AXXXXXXXX.py` is your python code file name,
  - `FILENAME` is the input CSV file name,
  - `EPSILON` is the ε parameter in epsilon-greedy approach
    - it is a real number from the interval [0; 1],
    - if illegal value provided, set to 0.3.
  - `TRAIN%` is representing the percentage of data (first `TRAIN%` of rows in the input csv file) that will be used for training purposes:
    - it is an integer number from the interval [0; 50],
    - if illegal value provided, set to 50.
  - `THR` is representing a "success threshold" (time series data below threshold - success, otherwise failure | In the provided file success will mean "no occupancy = if we choose to use unoccupied channel, we will not cause any interference; failure: we will cause interference):
    - for WiFi data you can use -90, **but DON'T HARDCODE it**.

  Example:

```
python cs581_P02_A11111111.py INPUT.CSV 0.3 30 -90
```

  If the number of arguments provided is NOT four your program should display the following error message:

```
ERROR: Not enough or too many input arguments.
```

and exit.

- Load and process input data file provided (assume that input data file is ALWAYS in the same folder as your code - this is REQUIRED! DO NOT HARDCODE YOUR LOCAL FILE PATH). Make sure your program is **flexible enough to accommodate different input data set** (with different labels, number of columns, and number of rows, but structurally the same). Your submission will be tested using a different file!

  - Report results on screen in the following format:

```
Last Name, First Name, AXXXXXXXX solution:
epsilon: eeee
Training data percentage: pppp %
Success threshold: ssss

Success probabilities:
P(LABEL1) = PL1
P(LABEL2) = PL2
P(LABEL3) = PL3
...
P(LABELN-1) = PLN-1
P(LABELN) = PLN

Bandit [LABELX] was chosen to be played for the rest
of data set.
LABELX Success percentage: xxxx
```

where:

- `AXXXXXXXX` is your IIT A number,
- `eeee` is the ε parameter,
- `pppp` is training data percentage,
- `ssss` is the success threshold,
- `LABEL1, LABEL2, LABEL3, …, LABELN-1, LABELN` are input file column labels / bandit names
- `PL1, PL2, PL3, …, PLN-1, PLN` are probability of success estimates for each bandit obtained during the training phase
- `LABELX` is the bandit (input file column) name that was chosen to be "played" for the rest of the data (remaining 100% - `TRAIN%` rows)
- `xxxx` is the success percentage (how often did we succeed / won / did not interfere) for the chosen `LABELX` bandit.

Results and Conclusions

Run your algorithm THREE (3) (three runs for each parameter pair to account for randomness) for the ε and TRAIN% parameter pairings (keep the success threshold fixed) shown in tables below and report your findings (final success rates).

TABLE A: RUN 1 Results comparison					
	TRAIN%				
ε	10	20	30	40	50
0.1	0.98656	0.75895	0.98830	0.99058	0.99103
0.2	0.98656	0.99220	0.98830	0.99058	0.99298
0.3	0.99220	0.99220	0.99247	0.99285	0.99103
0.4	0.99220	0.99220	0.98830	0.99285	0.99298
0.5	0.99220	0.98952	0.98830	0.99285	0.99298
0.6	0.99220	0.99220	0.99248	0.99285	0.99103
0.7	0.99220	0.99220	0.98830	0.99285	0.99298
0.8	0.98656	0.99220	0.99248	0.99285	0.99103
0.9	0.98656	0.99220	0.99248	0.99285	0.99298

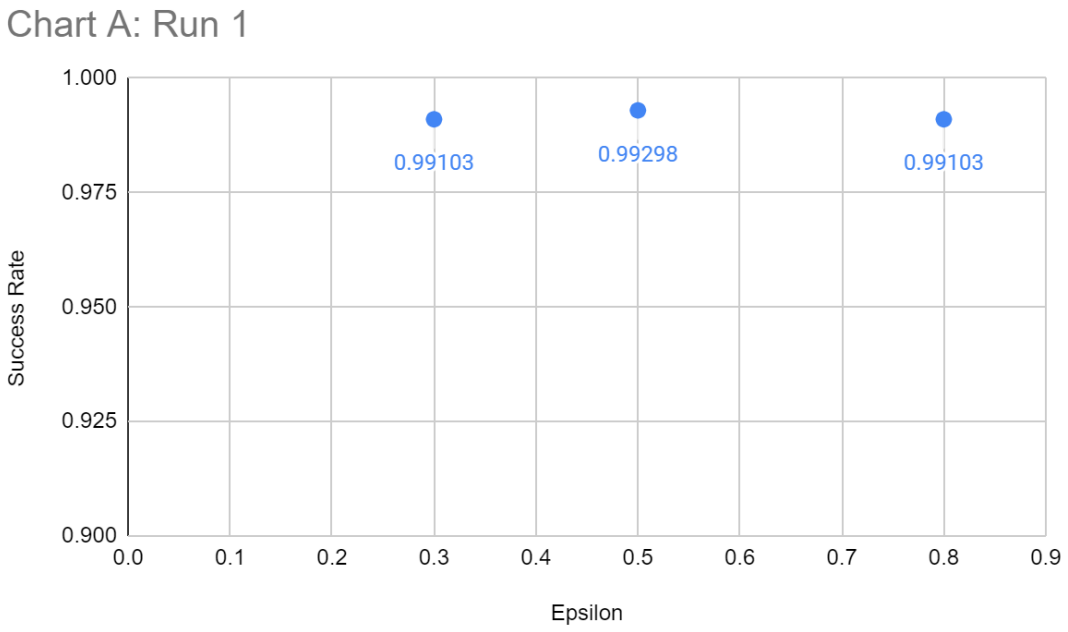


TABLE B: RUN 2 Results comparison					
	TRAIN%				
ε	10	20	30	40	50
0.1	0.98656	0.98951	0.99248	0.99285	0.99103
0.2	0.98656	0.99220	0.99248	0.99058	0.99298
0.3	0.85203	0.99220	0.98830	0.99285	0.99103
0.4	0.98656	0.99220	0.99248	0.99285	0.99298
0.5	0.99220	0.99220	0.99248	0.99285	0.99298
0.6	0.98656	0.99220	0.99248	0.99285	0.99298
0.7	0.88864	0.98952	0.98830	0.99285	0.99298
0.8	0.99220	0.98952	0.99248	0.99285	0.99298
0.9	0.99220	0.99220	0.99248	0.99285	0.99298

Chart B: Run 2

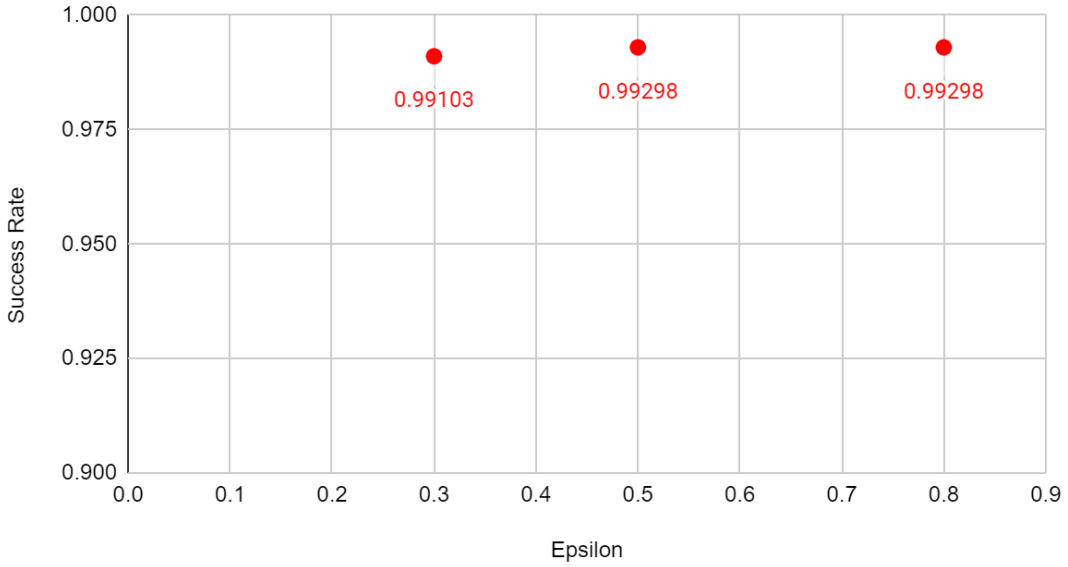
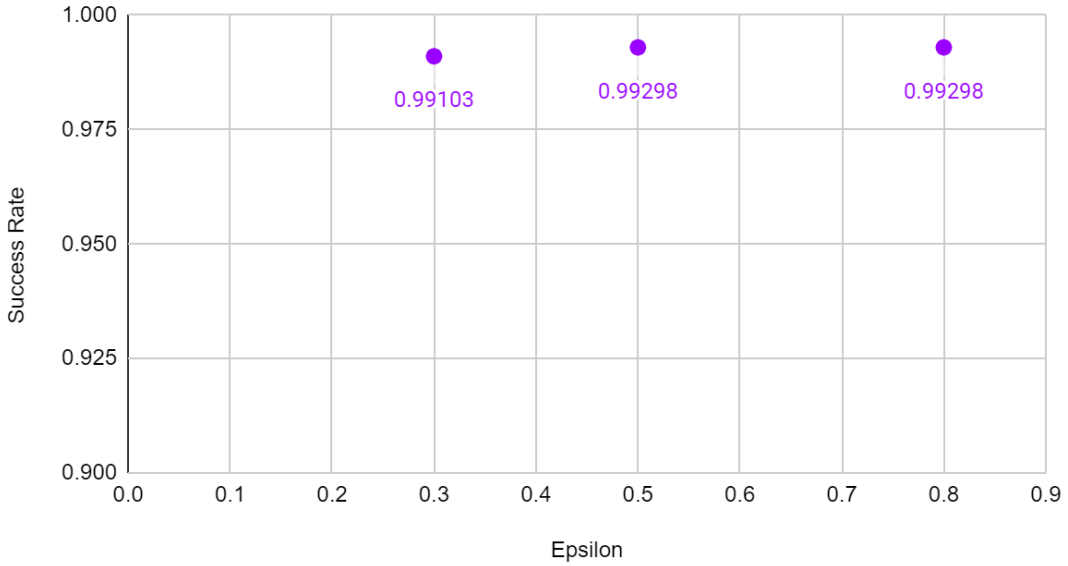


TABLE C: RUN 3 Results comparison					
	TRAIN%				
ε	10	20	30	40	50
0.1	0.76408	0.99220	0.98830	0.99285	0.99298
0.2	0.98656	0.99220	0.98830	0.99058	0.99298
0.3	0.98656	0.98952	0.98830	0.99058	0.99103
0.4	0.98656	0.98952	0.99248	0.99285	0.99298
0.5	0.99220	0.98952	0.99248	0.99285	0.99298
0.6	0.99220	0.99220	0.99248	0.99058	0.99298
0.7	0.99220	0.99220	0.98830	0.99058	0.99103
0.8	0.99220	0.98952	0.99248	0.99058	0.99298
0.9	0.99220	0.99220	0.99248	0.99285	0.99298

Chart C: Run 3



Figures: show

cumulative bandit success rate = f(time/row)

plots (NOT training rows | three traces/plots per figure | one figure per run) for the following ε and `TRAIN%` parameter pairings: 0.3/50, 0.5/50, 0.8/50. Feel free to add additional plots and comments if you discovered anything interesting.
What are your conclusions? What have you observed? Which algorithm/parameter set performed better? Was the optimal path found? Write a summary below.

Conclusions
From my observations, when a higher training percentage is used the success rate will be higher. The difference between different epsilon was not big enough to really make a difference.