Matthew Muranaka

Objectives:

1. Implement and evaluate a K-Armed Bandit algorithm.

Input data file:

The input file will be a CSV (comma separated values) file (`input.csv`).

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

Each column (starting at row 2) in that input file represents 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:

- Python code file(s), named:

  `tester.py`

- This document with observations and conclusions, named:

  `Multi-Armed_Bandit_Write_Up.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].

The 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)

The program:

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

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

  where:
  - `tester.py` is the 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 I can use -90.

Example:

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

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

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

and exit.

- Load and process input data file provided (<u>assuming the input data file is ALWAYS in the same folder as code</u>. Make sure the program is **<u>flexible enough to accommodate different input data set</u>** (with different labels, number of columns, and number of rows, but structurally the same).
- Report results on screen in the following format:

```
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:

- `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 the 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	0.9924176776	0.9924445528	0.9927576602	0.9931751706	0.9914196568
0.2	0.9924176776	0.9897635876	0.9927576602	0.9931751706	0.9914196568
0.3	0.888864818	0.9924445528	0.9927576602	0.9931751706	0.9933697348
0.4	0.9867850953	0.9924445528	0.9927576602	0.9931751706	0.9933697348
0.5	0.9924176776	0.9924445528	0.9927576602	0.9931751706	0.9933697348
0.6	0.9924176776	0.9924445528	0.9927576602	0.9931751706	0.9933697348
0.7	0.9924176776	0.9924445528	0.9927576602	0.9931751706	0.9933697348
0.8	0.9924176776	0.9924445528	0.9927576602	0.9931751706	0.9933697348
0.9	0.9924176776	0.9924445528	0.9927576602	0.9931751706	0.9933697348

TABLE B: RUN 2 Results comparison					
	TRAIN%				
ε	10	20	30	40	50
0.1	0.888864818	0.9897635876	0.9885793872	0.9931751706	0.9933697348
0.2	0.9924176776	0.9924445528	0.9885793872	0.9909002275	0.9933697348
0.3	0.9924176776	0.9897635876	0.9927576602	0.9931751706	0.9933697348
0.4	0.888864818	0.9924445528	0.9927576602	0.9931751706	0.9933697348
0.5	0.9924176776	0.9924445528	0.9927576602	0.9931751706	0.9933697348
0.6	0.9924176776	0.9924445528	0.9927576602	0.9931751706	0.9933697348
0.7	0.9867850953	0.9924445528	0.9927576602	0.9931751706	0.9933697348
0.8	0.888864818	0.9924445528	0.9927576602	0.9931751706	0.9933697348
0.9	0.9867850953	0.9924445528	0.9927576602	0.9931751706	0.9933697348

TABLE C: RUN 3 Results comparison					
	TRAIN%				
ε	10	20	30	40	50
0.1	0.9924176776	0.9897635876	0.9885793872	0.9909002275	0.9933697348
0.2	0.9867850953	0.9897635876	0.9885793872	0.9931751706	0.9914196568
0.3	0.9867850953	0.9924445528	0.9885793872	0.9931751706	0.9914196568
0.4	0.9924176776	0.9924445528	0.9927576602	0.9931751706	0.9933697348

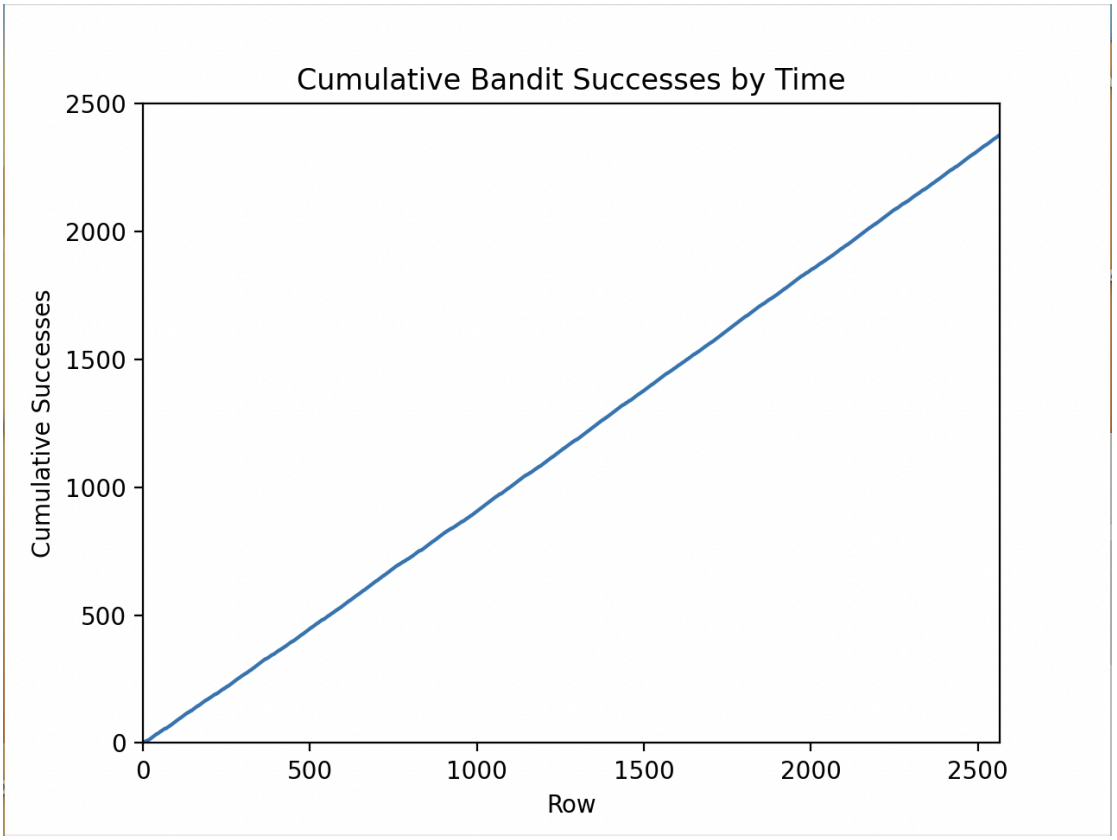
0.5	0.9924176776	0.9924445528	0.9927576602	0.9931751706	0.9933697348
0.6	0.9867850953	0.9897635876	0.9927576602	0.9931751706	0.9933697348
0.7	0.888864818	0.9924445528	0.9927576602	0.9931751706	0.9933697348
0.8	0.9867850953	0.9924445528	0.9927576602	0.9931751706	0.9933697348
0.9	0.9924176776	0.9924445528	0.9927576602	0.9931751706	0.9933697348

Figures: show

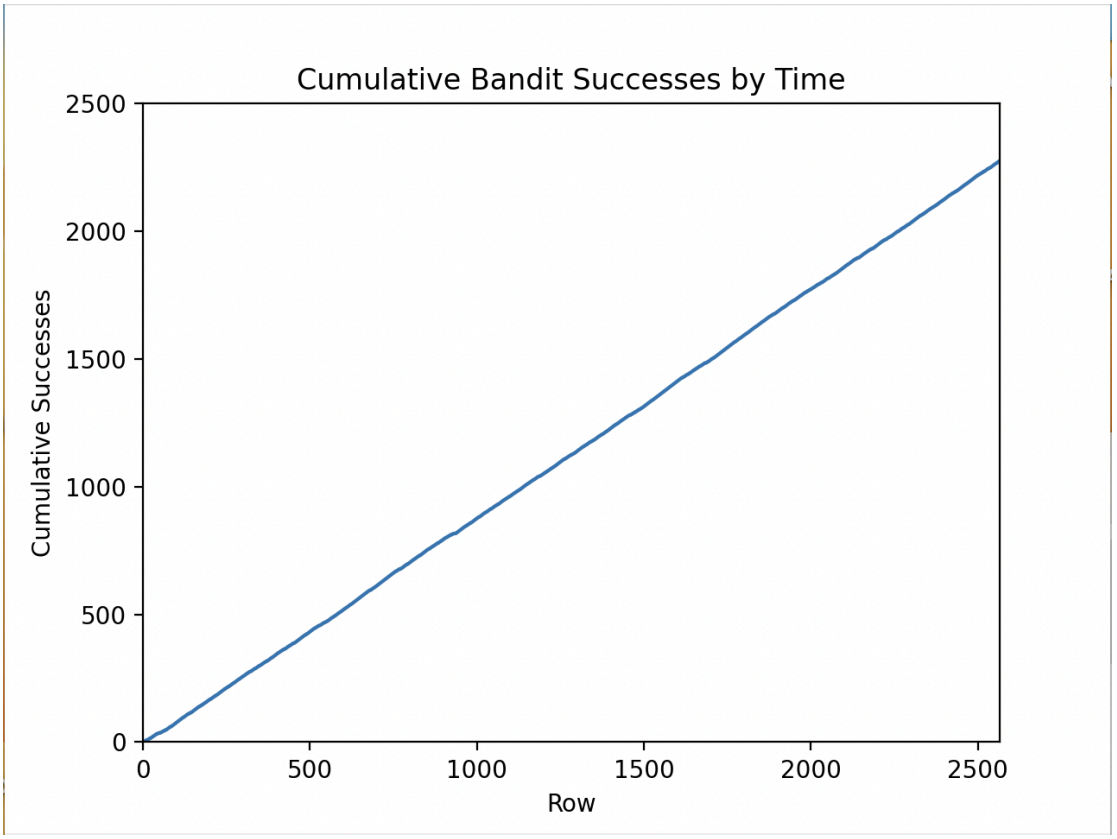
cumulative bandit success rate = f(time/row)

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.

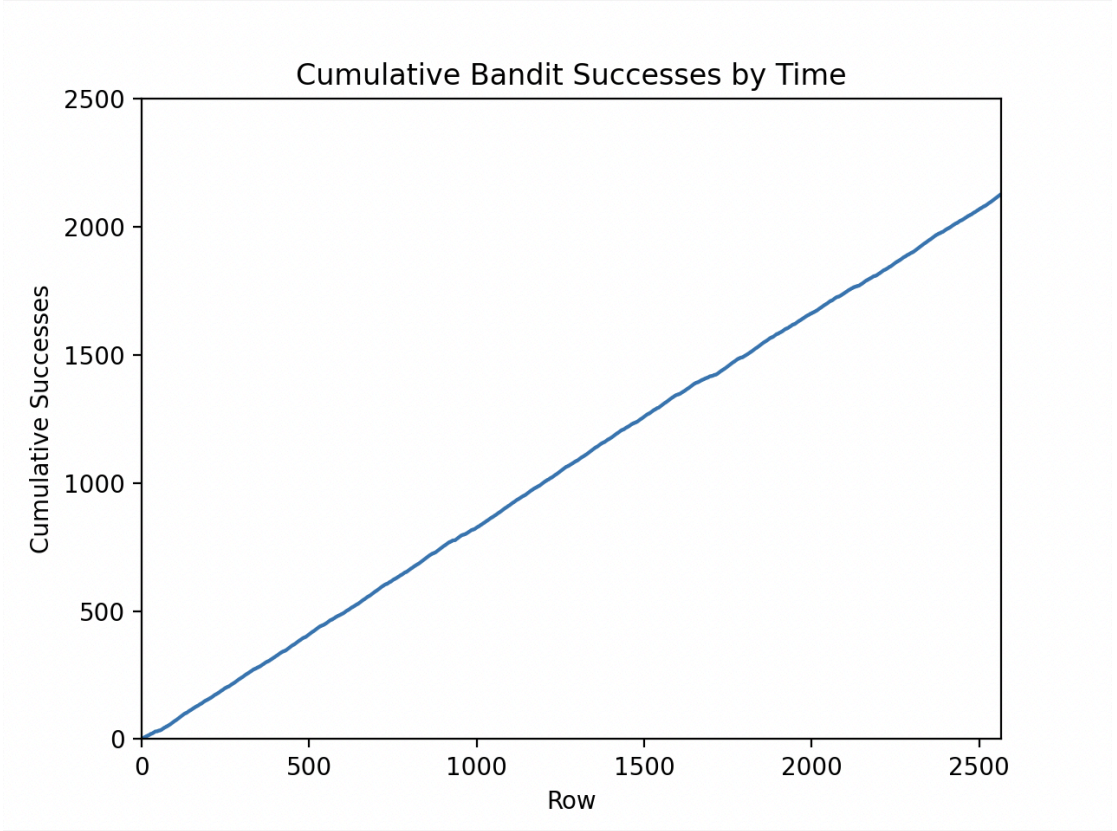
0.3/50



0.5/50


Cumulative Bandit Successes by Time
Cumulative Successes
Row
0
500
1000
1500
2000
2500

0.8/50


Cumulative Bandit Successes by Time
Cumulative Successes
Row
0
500
1000
1500
2000
2500

Conclusions

After plotting these graphs, I am a bit surprised with how they look. The growth of the successes is pretty much linear, meaning the bandit doesn't improve over time but instead keeps its process up. I thought it would be curved because choosing the best arm would result in more successes but I think there is an error in that logic where consistent exploitation would lead to deterioration.

I am also surprised with the fact that the lowest epsilon value resulted in the highest cumulative successes which I guess promotes exploration over exploitation.

For training size, it appears that with a larger training size, the algorithm is more likely to pick a certain arm as the best arm. I think part of this has to do with the data being top heavy where certain arms are better than others (3 and 4). This works out for this data set because those arms produce high success percentages, but the algorithm could run into errors when the data is more variable over time (wifi signal strength is altered, people moving around in library).