AI 1013: PROGRAMMING FOR AI

ASSIGNMENT 5

DUE ON: 01 MAY 2025, 11:59 PM IST.

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Read each question carefully, and answer all parts. Write well-documented codes with contextual variable names. Put the codes (.py or .ipynb) and plots (.png, .jpg, .jpeg, .pdf) of all the questions into one single folder, compress this folder, and upload the .zip file. You should name the .zip file as `AI1013_assignment_5_<your roll number>.zip`.

Contents

1 Sampling via Inverse Transform Technique

Consider the cumulative distribution function (CDF) F given below:

$$F(x) = \begin{cases} 0, & x < 0, \\ \frac{x^2}{3}, & 0 \le x < 1, \\ \frac{1}{3}, & 1 \le x < 2, \\ \frac{x+2}{6}, & 2 \le x < 4, \\ 1, & x \ge 4. \end{cases}$$

1. Set a seed value of 100 to start with.
2. Generate $N = 1000$ independent and identically distributed samples from the uniform distribution on the interval $(0, 1)$. Store these generated samples in an array named `array_samples_uniform`.
3. Implement a Python function to compute the inverse function F^{-1} corresponding to the CDF F provided.
4. Using the inverse function computed in step 2, transform each uniform sample from `array_samples_uniform`. Store the resulting transformed samples in an array named `array_samples_custom`.
5. Carry out the following visualization tasks:
    - Plot the empirical CDF corresponding to the samples stored in `array_samples_custom`. Display this plot in one figure window.
    - In a separate figure window, plot the given theoretical cumulative distribution function F.
    - Utilize the `FuncAnimation` method from the `matplotlib.animation` library to animate the empirical CDF plot as the number of samples (N) increases. You may start the animation from $N = 10$ and go all the way up to $N = 1000$. For each value of N, you may choose a random subset of the 1000 samples in `array_samples_uniform`

    A sample output demonstrating the expected animation is provided for reference.

    Note: The empirical CDF plot must be animated progressively with respect to increasing values of N, while the plot depicting the theoretical CDF F remains unchanged.

2 Wordle: A Game in Probability

In this assignment, you will implement the popular word puzzle game, Wordle. By systematically analyzing Wordle's mechanics, you will deepen your understanding of probabilistic reasoning, empirical estimation, and how the law of large numbers can be exploited to estimate unknown probabilities in practice.

For a nice description of the Wordle game and its rules, see this Mashable article. For a visual demonstration of Wordle and the role probability theory plays in it, see this informative YouTube video from 3B1B.

Note: All words in this game are case-sensitive.

2.1 Dataset

You are provided with a file named `wordlist.csv` containing 3103 common five-letter English words. Each word in the list will serve either as a solution word or a guess word in the simulation.

2.2 Data Exploration and Setup

Load the provided word list into Python. Print the total number of words to verify that you have loaded the data correctly. Compute and visualize the histogram of letters appearing in each of the five positions (1 through 5) across all words. Create a clear histogram visualization for each of the five positions.

2.3 Simulating Wordle Feedback

Wordle provides feedback as follows:

- Green: correct letter in the correct position.
- Yellow: correct letter but in the wrong position.
- Gray: letter not in the word.

Write a Python function `wordle_feedback` that accepts two five-letter words `solution` and `guess`, and returns the number of green, yellow, and gray tiles as a tuple. Demonstrate your function clearly by showing outputs for the following examples:

- `guess` = CRANE, `solution` = SLATE,
- `guess` = BRAIN, `solution` = BRINE.

Before proceeding with the below subsections, set a seed value of 42 for reproducibility.

2.4 Probability Estimation via Averages

In this part, you will estimate certain unknown probabilities by carrying out independent trials and averaging the results obtained from these trials. This is an age-old trick to estimate probabilities, and has its roots in the strong law of large numbers, one of the key results in probability theory.

1. **Estimating the probability that a guess word results in at least one green tile:**
   Fix the word CRANE as the `guess` word. Simulate the game 1000 times, each time randomly selecting a `solution` word from the provided word list. Using your `wordle_feedback` function from an earlier exercise, compare the `solution` and the `guess` words, and come up with a 1/0 value for each `solution` word, based on whether it results/does not result in **at least one green tile** when compared with the `guess` word CRANE. From the 1/0 values obtained, provide an estimate of the probability that the word CRANE results in at least one green tile.

2. **Estimating the probability that a guess word results in exactly two yellow tiles:**
   Repeat the above exercise to estimate the probability that the word CRANE results in **exactly two yellow tiles**.

2.5 A Greedy Strategy and its Average Guessing Time

In this part, you will implement the Wordle game via a greedy strategy that uses the Wordle feedback to reduce the search space until eventually the underlying five letter word is identified.

1. Fix a `guess` word. Execute the following steps for a total of 1000 independent trials, and in each trial, record the number of word selections until the `guess` word is correctly identified.

   (a) Randomly select a `solution` word from the provided word list.

   (b) Compare the `solution` and `guess` words, and filter the word list to retain only consistent candidate words.

   (c) Select a `solution` word randomly from the filtered word list.

   (d) Iterate through (a)-(c) until the `guess` and `solution` words match (i.e., all five tiles are green).

   Based on the results of the 1000 trials, compute the average number of steps required to identify the `guess` word.

2. Run the greedy algorithm on two different `guess` words: VERVE and CRANE. Report the average number of word selections (computed across 1000 independent trials) corresponding to each of the words.

2.6 Evaluating Different Strategies for Choice of `guess`

Two students of IIT Hyderabad (let's call them S_1 and S_2) come up with two distinct strategies for finding a good `guess` word. Student S_1 proposes using a word with the largest probability of getting at least one green tile, while student S_2 proposes using a word with the largest probability of getting exactly two yellow tiles.

1. Identify a word from `wordlist.csv` satisfying student S_1's proposal. If there are multiple words, you may report any one of them. Also report the probability estimate of getting at least one green tile corresponding to using the identified word as the `guess` word.

2. Identify a word from `wordlist.csv` satisfying student S_2's proposal. If there are multiple words, you may report any one of them. Also report the probability estimate of getting exactly two yellow tiles corresponding to using the identified word as the `guess` word.

3. Students S_1 and S_2 pick a fight with one another on whose strategy is better, and a third student S_3 intervenes and attempts to amicably resolve the fight by proposing the following strategy for picking the best `guess` word. Given a word w belonging to the `wordlist.csv` file, let g_w denote the probability estimate (computed using the procedure of the previous section) of getting at least one green tile when w is used as the guess word, and let y_w denote the probability estimate (again computed using the procedure of the previous section) of getting at least two yellow tiles when w is used as guess word. For $\alpha \in [0,1]$, let $F(\alpha)$ be defined as

   $$F(\alpha) := \min_{w \,\in\, \texttt{wordlist.csv}} \alpha\, g_w + (1-\alpha)\, y_w.$$

   Student S_3's proposal is then as follows: find the optimal value of α (say α^*) where F attains its maximum. Then, pick a word w in `wordlist.csv` for which the value of $\alpha^* g_w + (1-\alpha^*) y_w$ is the smallest among all words in `wordlist.csv`. If there are multiple such words, pick any of them.

   Using the optimisation routines available in the `cvxpy` package, determine α^*, and identify a word from `wordlist.csv` satisfying student S_3's proposal.

   Note: `cvxpy` is a package in Python designed for solving convex optimization problems.