Programming Assignment 3 (due 4/14)

DS-563 / CD-543 @ Boston University

Spring 2023

Before you start…

Collaboration policy: You may verbally collaborate on programming assignments, however, you must write your code and final report independently, i.e., without seeing other students' solutions. If you choose to collaborate on a problem, you are allowed to discuss it with at most **four** other students currently enrolled in the class.

The header of each assignment you submit must include the field "Collaborators:" with the names of the students with whom you have had discussions concerning your solutions. A failure to list collaborators may result in a credit deduction.

You may use external resources such as textbooks, lecture notes, and videos to supplement your general understanding of the course topics. You may use references such as books and online resources for well known facts. However, you must always cite the source.

You may **not** look up solutions to a programming assignment in the published literature or on the web. You may **not** share written work with anyone else. If you wish to make your code public (for instance, by making it publicly available on GitHub or GitLab), please wait till the end of the semester. (If you want to publish it earlier for whatever reason, please contact us first.)

Submitting: Your solution is to be submitted via Gradescope (entry code: 4VYBJ6). In particular, you should submit a pdf with your project report and a zip file with your code (or an equivalent of these as long as it's allowed by Gradescope). Don't forget to provide information how to obtain your data sets. More information on this is provided below.

Late submission policy: No extensions. Submitting a solution one school day late may result in a deduction of 10% of points.

Your task

Our goal is to implement the uniformity test and apply it in two scenarios. Unfortunately, the known theoretical upper bounds on the number of samples make it relatively prohibitive. Hence our plan is to use experiments to determine bounds that are likely to be useful to detect the divergence from the uniform distribution in practice. In particular, we want to use experiments to determine the number of samples needed to distinguish the uniform distribution from a distribution that is ϵ-far from uniform, but minimizes the expected number of collisions.

1. Implement two subroutines for generating random samples:

   (a) A subroutine generating samples from the uniform distribution on $[n]$. Denote this distribution $\mathcal{U}$.

   (b) A subroutine generating samples from some distribution on $[n]$ that is ϵ-far from uniform in total variation distance and minimizes the expected number of collisions. Use the knowledge from the lecture to explain in your report why this distribution minimizes the expected number of collisions. Denote this distribution $\mathcal{D}_{\text{far}}$.

   Note: For simplicity, you can assume that n is even.

2. Given n and ϵ as above and an error parameter δ, write a subroutine for finding a sufficient number s of samples and threshold t that work with probability $1 - \delta$. More specifically, if you collect s samples from $\mathcal{U}$, the number of collisions should be at most t with probability (approximately) at least $1 - \delta$, and if you collect s samples from $\mathcal{D}_{\text{far}}$, the number of collisions should be more than t with probability (approximately) at least $1 - \delta$.

   Your subroutine does not have to be very efficient, but it should run in reasonable time for, say, $n = 10,000$, $\epsilon = 0.2$, and $\delta = 0.1$. It is sufficient to use it just once—for each setting of n, ϵ, and δ of interest—to determine parameters that can be used in the rest of this programming assignment.

   Hint: You can use the following approach. For a given candidate s and each of $\mathcal{U}$ and $\mathcal{D}_{\text{far}}$, collect s samples and write down the number of collisions you observed. Repeat this a sufficiently large number of times, at least $\gg 1/\delta$, to see how this number is distributed for $\mathcal{U}$ and $\mathcal{D}_{\text{far}}$. Can you select a threshold t so that at least a $1 - \delta$ fraction of collision numbers you observed for $\mathcal{U}$ are below the threshold and at least a $1 - \delta$ fraction of collision numbers you observed for $\mathcal{D}_{\text{far}}$ are above the threshold? If so, use this s and t. If not, increase s (for instance, multiply it by 1.1) and try again.

3. **(Optional, no credit)** Can you design an ϵ–far from uniformity destribution that is even worse for the uniformity test than $\mathcal{D}_{\text{far}}$? Or can you show that $\mathcal{D}_{\text{far}}$ is the worst–case distribution? If you know how to answer any of these questions, you can probably turn this into your final project, if you want to update your project topic.

4. Recall Question ~~7~~8 in Homework 1. Choose an arbitrary real–world data set that consists of numeric records for which it feels that the last x digits (or the middle x digits) should be distributed independently and uniformly at random.

   (a) Explain why you believe this should be the case.

   (b) What does the uniformity test applied to this data set say?

   (c) What does the uniformity test applied to pairs of consecutive data points say? As an example, if you look at the last two digits, for each data point, combine them into numbers consisting of four digits, before feeding them into the tester. (Also remember to update your parameters s and t correspondingly!)

   Note: `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, ... would pass the previous test. However, if you look at pairs, the sequence of samples you get is `01`, `23`, `45`, `67`, ..., and the uniformity test will detect that there is a relationship between consecutive data points.

   (d) **(Optional, no credit)** Can you design a statistical test that the uniform distribution with independent samples should pass, but this distribution does not?

5. The theme of this question is a random PIN or password generation. Say, you want to generate a random sequence of digits. If you try to just type one yourself, it may be difficult to select one uniformly at random from all sequences of specific length. However, consider the following *extreme* experiment that may boost the quality of randomness in a single digit you generate.

   - Try to type as randomly as possible 1000 digits.
   - Add them up and output the result modulo 10.

   While typing directly single digits uniformly and completely independently of other digits may be nearly impossible, we hope in this experiment that each generated digit is a result of combining a large number of sufficiently random digits that will allow for the result to be distributed nearly uniformly and independently of all other events.

   Your task: Instead of 1000, let q be the number of digits that you use to generate a single digit.

   (a) Experiment with different q and the distribution of digits you get. What is the lowest q for which you can pass the uniformity test designed for small ϵ and δ? (Sample settings that should not result in a prohibitively large number of samples required are $\epsilon = 0.1$ and $\delta = 0.1$, but decreasing ϵ to 0.05 should still be possible. See how far you can go!)

   (b) How about pairs, triples, etc. of consecutive digits? Do they pass the uniformity test for your q? If not, how much do you have to increase q to make them pass the test? Continue the experiments only until you do not need a prohibitively large number of keystrokes to perform it. Compared to the single digit test, you can increase ϵ and δ to lower the number of samples required.

   (c) **(Optional, no credit)** Can you come up with a statistical test that this approach does not pass if it passes the uniformity tests above?

   Hint: Instead of collecting keystrokes every time, consider creating a text file which contains a very long seqence of digits that you randomly typed and reuse it every time you run an experiment.

   Alternate version: Instead of typing a long sequence of digits, feel free to use ChatGPT, Google Bard, or similar technology to generate a long random sequence of digits. How good are these systems as random number generators?

6. How much time (approximately) did you spend on this homework? Was it too easy/too hard?

   Note: You will not be evaluated based on your answer to this question.

Deliverables

Your final submission should include:

- Code:
  - Provide your implementation of all subroutines.
  - Provide any auxiliary tools you develop for preprocessing data or generating artificial data sets.
  - Make your code readable (use reasonable variable and function names, etc.).
- Report:

- It should include all important implementation details and discuss implementation decisions. Briefly describe how to run your code with an example.
- It should address all questions from the previous section. Use tables and/or graphs to display numerical results of your experiments.

• Data: Describe your data sets and make sure we can access them. Think of what someone would need to reproduce your results.

  - For any public data set that you use, provide a link to where this data set can be obtained from and include code used for its preprocesssing (if needed).
  - For any artificial data set, provide code that generates it (try to make it deterministic, by for instance, fixing the random seed it uses) or share a link (Dropbox, Google Drive, etc.) that can be used to access the data set. If you provide a link, make sure it does not require logging in.