Ira A. Fulton Schools of Engineering
Arizona State University

CSE 535: Mobile Computing

Markov Chains Assignment

Purpose

Markov chains are a quintessential tool in statistical modeling with applications in a variety of fields, such as reinforcement learning, the PageRank algorithm, thermodynamic systems, finance risk analysis, and much more. In this assignment, you will gain firsthand experience creating and using Markov chains. By the end of the assignment, you will write Python code that inputs a sequence of states and outputs the transition matrix corresponding to the Markov chain, a new set of samples according to that distribution, and the stationary distribution of those states.

Objectives

Learners will be able to:

- Construct a Markov chain from an input sequence to analyze patterns in the given sequence.
- Design a context-aware application.
- Identify the advantages of using context in applications.
- Discuss various types of mobility models.

Technology Requirements

- Python 3.10.6
- NumPy 2.2.2

Description

In this project, you will be given an input sequence of daily weather patterns, categorized into multiple groups. Using that sequence of data, you will extract the probability distribution corresponding to transitions, build the Markov chain, generate new data, and compute the stationary distribution of the Markov chain in order to gain information about the input data sequence.

The stationary distribution represents the long-term behavior of the Markov chain and is defined as the vector π, such that $\pi P = \pi$, where P is the transition matrix. This means that once the system reaches the stationary distribution, the probabilities of being in each state no longer change with time.

There are several ways to compute the stationary distribution, such as setting up a system of simultaneous equations with a normalization constraint and using any method to solve that. The recommended method for this assignment makes use of the eigenvalue/eigenvector equation:

$Mv = \lambda v$ for matrix M, eigenvector v and eigenvalue λ. We can transform our original equation by taking the transpose of both sides, $(\pi P)^T = \pi^T => P^T \pi^T = \pi^T$. Thus, π is the eigenvector of the transpose of the transition matrix corresponding to eigenvalue 1. Remember that you may still need to normalize this vector since it is a probability distribution.

Directions

1. All methods should be defined within a class named A1 (and therefore take as input self as an additional first parameter. If an instruction requests two parameters, you would have self and then those two. Please name your file a1.py.
2. Define a method generate_markov_chain that takes as input two lists of strings, a list of potential states, and a sequence of states. You should store the list of potential states for later, to help with indexing the transition matrix in a later method.
    a. In this method, you should generate the transition matrix and store it as a 2d NumPy array of floats. Store this as a variable transition_matrix for later use. Your method should return this array.
3. Define a method generate_samples that takes as input a string representing the first state and two integers, a random seed and a length. It should return a list of strings containing the generated sequence. If the input length is nine (9), the generated sequence should start with the input string state and be a total of ten (10) states long.
    a. Using the Python random library, seed the rng using the given seed. Then, still using the random library, generate a sequence of new states of the given length according to your Markov chain's distribution.
    b. You can use random.random() to generate a probability. Note that random.random() cannot generate 1.0. While this can be resolved using random.uniform(0, 1), we will use open upper bounds on all our transitions as described below instead.

   c. If you have a transition from state a with probabilities [0.25, 0.3, 0.45] to transition to a, b, c respectively, a value in the range [0, 0.25) results in a, [0.25, 0.55) results in b, and [0.55, 1) results in c.

4. Define a method stationary_distribution that computes and returns the stationary distribution (also called steady-state vector) as a 1d NumPy array.
   a. The stationary distribution is defined above in the Description section. Generate it, ensuring it is normalized, and return it.

Submission Directions for Deliverables

You are given an unlimited number of attempts to submit your best work. You must submit your Markov Chains Assignment deliverable through Gradescope. Carefully review submission directions outlined in this overview document in order to correctly earn credit for your work. Learners may not email or use other means to submit any assignment or project for review, including feedback, and grading.

The Markov Chains Assignment includes one (1) deliverable:

- **a1.py:** Source code name must match exactly. Ensure you match the class and method naming conventions described in the directions.

Submitting to Gradescope

After completing work in GitHub, you must submit your work into Gradescope to receive credit for the course:

1. Go to the Canvas Assignment, "**Submission: Markov Chains Assignment**".
2. Click the "**Load Submission…in new window**" button.
3. Once in Gradescope, select the assignment titled "**Markov Chains Assignment**" and a pop-up window will appear.
4. In the pop-up:
   a. Submit your Python file “**a1.py**”.
   b. Click “**Upload**” to submit your work for grading.
5. If needed: to resubmit the assignment in Gradescope:
   a. Return to the Canvas submission and open Gradescope.

b. You will be navigated to the "Autograder Results" page (if it is not your first submission).

c. Click the "**Resubmit**" button on the bottom right corner of the page and repeat the process from Step 3.

Your submission will be reviewed by the course team and then, after the due date has passed, your score will be populated from Gradescope into your Canvas grade.

Evaluation

Your submission will be auto-graded. Please review the Evaluation Criteria for how your source code will be assessed. Submissions will be evaluated based on the criteria and will receive a total score. Submissions missing any part of the project will be graded based on what was submitted against the evaluation criteria. Missing parts submitted after the submission deadline will **not** be graded.

Review the course syllabus for details regarding late penalties.

Evaluation Criteria

Your submission will be evaluated and scored based on these criteria:

- Your code will be tested against a series of eleven (11) increasingly difficult test cases.
- Each test case will verify your transition matrix, generated samples, and stationary distribution.
- To ensure reasonable partial credit, the first test gives 40 points for the transition matrix, and 20 points each for the generated samples and stationary distribution.
- The latter ten (10) test cases offer 1, 0.5, and 0.5 points respectively.