APPLICATION OF GRAPH THEORY IN MARKOV CHAINS & PAGE RANK ALGORITHM

April 22, 2019

Anupriy (150121)
Deepanshu Bansal (150219)
Raushan Kumar (150571)

1 Introduction

Markov chains are an important mathematical tool in stochastic processes. The underlying idea is the Markov Property, in order words, that some predictions about stochastic processes can be simplified by viewing the future as independent of the past, given the present state of the process. This is used to simplify predictions about the future state of a stochastic process. Markov chains are widely used in the modelling of various physical and conceptual process that evolve over time. For example, variation of stock market prices, Google's PageRank, the spread of disease within population etc. Each of these examples are characterized by the property that the system can be found in any of the finite number of states, and transition between states occur at discrete instants according to specified probabilities.

Our objective here is to analyze and implement different features of discrete time finite state markov chain processes with a graph theoretic approach. Further our aim is to implement some application of Markov Chain such as PageRank.

The graph theoretic representation immediately suggests several schemes for calculating important structural characteristics of the process. The properties of the corresponding Markov Chain can easily be deduced from transition matrix manipulation using appropriate data structures and algorithmic techniques. We try to use algorithms such as finding strongly connected component in a directed graph.

2 Markov Chain

A Markov Chain is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event.

Let $\left\{ X_n : n = 0,1,2,...,\right\}$ be a stochastic process that takes on a finite or countable number of possible values. if the present state is denoted by X_n, the future state by X_{n+1} and the past states are represented by X_{n-1} to X_0 , then,

$$Pr(X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1}, ..., X_1 = i_1, X_0 = i_0) = Pr(X_{n+1} = j | X_n = i) = Pr_{ij}$$

where Pr_{ij} is the transition probability from state i to state j. This property is known as *Markov Property.*

Our aim is to find the nth step transition probabilities, expected number of steps before absorption, probability of absorption into a particular absorbing state, stationary

distributions, finding communicating blocks and residual states etc. We use graph theory to find the communicating blocks and residual states which will further help in finding the other things such as overall stationary distribution.

3 Implementation Details

3.1 Finite State Markov Chain

Our implementation assumes that transition matrix is stationary and number of states is finite. We have written functions for finding communicating blocks, residual states, standard (canonical) representation of the merged matrix i.e. transition matrix obtained by merging the communicating blocks. We have also calculated the expected number of visits in transient states before absorption, hitting probabilities i.e. probability of absorption in different absorbing states from different states. Finally, using these functions, we calculated the stationary distribution of the marov chain.

Input : Transition matrix (matrix.txt), Starting Distribution (start.txt)

3.1.1 Reading the inputs and validation (*ReadMatrix.py*)

In this module, Transition matrix and starting distribution is read from *transition.txt* and *start.txt* respectively. Inputs are validated properly. For ex: Elements of transition matrix are between 0 and 1 (both inclusive), Row sum of transition matrix is 1, starting distribution is proper.

```
Matrix read successfully
Matrix is a Valid Transition Matrix
Start Distribution read successfully
Valid Starting Distribution
Input Transition Matrix :
[[0.6 0.4 0.  0.  0. ]
 [0.4 0.6 0.  0.  0. ]
 [0.2 0.2 0.2 0.2 0.2]
 [0.  0.5 0.  0.5 0. ]
 [0.  0.  0.  0.  1. ]]

Input Starting Distribution :
[0.333333 0.       0.333333 0.       0.333334]
```

Figure 1: Output after reading transition matrix and starting distribution

3.1.2 Graph formulation (*Graph.py*)

In this module, transition matrix is formulated as a directed graph. Each of the states is represented as vertex. If there is a non-zero probability from state i to state j, there is an edge from vertex corresponding to state i to vertex corresponding to state j.
In the implementation, Graph is defined as a class which supports functions such as Depth first traversal (DFS), finding Strongly Connected Component (SCC) etc.

Strongly Connected Components : A directed graph is strongly connected if there is a path between all pairs of vertices. A strongly connected component (SCC) of a directed graph is a maximal strongly connected subgraph. We have used *Kosaraju's algorithm* to find all strongly connected component of the directed graph. Following are the steps :

1. Create an empty stack S and do DFS traversal of a graph. In DFS traversal, after calling recursive DFS for adjacent vertices of a vertex, push the vertex to stack.
2. Reverse directions of all edges to obtain the transpose graph.
3. One by one pop a vertex from S while S is not empty. Let the popped vertex be v. Take v as source and do DFS. The DFS starting from v prints strongly connected component of v.

3.1.3 Finding the communicating blocks and residual states (*findCB.py*)

Once strongly connected components are found using the previous module, we have to classify them as communicating block or residual states. For this, we check for each vertex of the SCC whether there is an edge out of the SCC. If there is at least one vertex in a SCC which has an edge out of the SCC then it belongs to Residual state otherwise it is a communicating block. This definition is taken from class notes. After this module,

```
These are the communicating blocks :
CB 1 : 0 1
CB 2 : 4
These are the residual states :
2 3
```

Figure 2: Output after finding communicating blocks and residual states

we have obtained communicating blocks and residual states.

3.1.4 Merging the Communicating Blocks (*mergeCB.py*)

This module takes as input transition matrix, communicating blocks and residual states and outputs the merged matrix. This module merges each of the communicating blocks into 1 state each. Now those communicating blocks are transformed into an absorbing state. Probability values of the transition matrix are updated accordingly. This new transition matrix is the output.

```
Merged Matrix :
[[1.  0.  0.  0. ]
 [0.  1.  0.  0. ]
 [0.4 0.2 0.2 0.2]
 [0.5 0.  0.  0.5]]
```

Figure 3: Output after merging transition matrix

3.1.5 Getting the canonical form of the merged matrix (*CanonicalForm.py*)

In this module, merged matrix obtained from previous module is converted to canonical (standard) form which is as follows.

$$\begin{bmatrix} \mathbf{Q} & \mathbf{R} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}$$

This operation required just few swapping. The above Q and R would be helpful in finding the hitting probabilities and expected number of visits to a particular transient state before absorption.

```
Absorbing States :  [0, 1]
Transient States :  [2, 3]
Standard Transition Matrix :
[[0.2 0.2 0.4 0.2]
 [0.  0.5 0.5 0. ]
 [0.  0.  1.  0. ]
 [0.  0.  0.  1. ]]

States :
[[2], [3], [0, 1], [4]]
```

Figure 4: Output after finding standard/canonical form of merged matrix

3.1.6 Expected number of visits (*ExpectedNumVists.py*)

This module calculates the expected number of visits to a transient state before absorption to a communicating block or absorbing state starting from a transient state. This is calculated by following formula :

$$\mathbf{W} = (\mathbf{I} - \mathbf{Q})^{-1}$$

$(i, j)th$ element of the matrix $\mathbf{W}$ represents expected number of visits to state j starting from state i before absorption.

```
*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.
To calculate Expected Number of Visits to transient states before getting absorbed.
*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.

Starting from state [2] expected number of visits in respective states before absorption is :
State [2] expected visits = 1.25
State [3] expected visits = 0.5
Time till absorption is 1.75 given we start in state [3]

Starting from state [3] expected number of visits in respective states before absorption is :
State [2] expected visits = 0.0
State [3] expected visits = 2.0
Time till absorption is 2.0 given we start in state [3]
```

Figure 5: Output after finding expected number of visits before absorption

3.1.7 Hitting Probabilities (*HittingProbs.py*)

This module calculates the probability of absorption to a communicating block or absorbing state starting from a transient state. This is calculated by the following formula:

$$\mathbf{U} = (\mathbf{I} - \mathbf{Q})^{-1}\mathbf{R}$$

$(i, j)th$ element of the matrix $\mathbf{U}$ represents probability of absorption to state j starting from state i. This is called as the *hitting probability*.

```
*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.
To calculate starting from certain transient state what is probability of getting absorbed in absorbing states.
*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.*.

Starting from state [2] prob. of absorption in different absorbing states is :
State [0, 1] absorption prob. = 0.75
State [4] absorption prob. = 0.25

Starting from state [3] prob. of absorption in different absorbing states is :
State [0, 1] absorption prob. = 1.0
State [4] absorption prob. = 0.0
```

Figure 6: Output after calculating hitting probabilities

3.1.8 Stationary Distribution (*stationaryDist.py*)

Finally stationary distribution is calculated for each communicating blocks and using these and hitting probabilities overall stationary distribution is calculated given the starting distribution.

```
Stationary Distribution :
[0.29166637 0.29166637 0.         0.         0.41666725]
```

Figure 7: Output after finding stationary distribution

3.1.9 Periodicity of Markov Chain and Limiting Distribution

In this module, we find the periodicity of Markov Chain using graph theory. This is required to conclude whether limiting distribution exists or not. If periodicity is 1 then limiting distribution exists and is same as the stationary distribution. But if periodicity is greater than 1 then limiting distribution doesn't exist.

Periodicity is found using the following algorithm :

- From an arbitrary root node, perform a breadth-first search of the graph constructed earlier producing the rooted tree T.
- The period g is given by $gcd\{val(e) > 0 : e \notin T\}$. Here e is an edge which does not belong to T and $val(e)$ is given by $level(i) - level(j) + 1$. Now $level(i)$ denotes the level of vertex i in rooted tree T as found in first step.

Proof of correctness of this algorithm is given in `http://cecas.clemson.edu/~shierd/Shier/markov.pdf`

3.2 Application - PageRank

PageRank is an algorithm used by Google Search to rank web pages in their search engine results.

PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.

The PageRank algorithm outputs a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page.

PageRank can be calculated for collection of documents of any size. The PageRank computations require several passes, called "iteration", through the collection to adjust approximate PageRank values to more closely reflect the theoretical true value.

Currently we have implemented the simplified algorithm which is based on the following formula :

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

Here, the PageRank value for a page u is dependent on the PageRank values for each page v contained in the set B_u (the set containing all pages linking to page u), divided by the number $L(v)$ of links from page v.

It can be understood as a Markov chain in which the states are pages, and the transitions, which are all equally probable, are the links between pages. As a result of Markov theory, it can be shown that the PageRank of a page is the probability of arriving at that page after a large number of clicks.

3.2.1 Dataset for training

- We used webbot crawl hollins.edu dataset
- It became publicly available in Jan '04
- It consists of 6,012 webpages.
- There are 23,875 connection between these webpages.

Here we find the stationary distribution of PageRank using the functions written above. Top 10 pages are returned after finding stationary distribution (probability of arrival at that page after large number of clicks).

```
Page at Rank  1  is  http://www.hollins.edu/calendar
Page at Rank  2  is  http://www.hollins.edu/calendar/null.htm
Page at Rank  3  is  http://www1.hollins.edu/faculty/saloweyca/clas%20395/Sculpture/Sculpture.ppt
Page at Rank  4  is  http://www1.hollins.edu/faculty/saloweyca/clas%20395/BronzeAGe/BronzeAGe.PPT
Page at Rank  5  is  http://www1.hollins.edu/faculty/saloweyca/clas%20395/kouroikorai/kouroikorai.ppt
Page at Rank  6  is  http://www1.hollins.edu/faculty/saloweyca/clas%20395/DAGEO/DAGEO.ppt
Page at Rank  7  is  http://www1.hollins.edu/faculty/saloweyca/clas%20395/Acropolis/Acropolis.ppt
Page at Rank  8  is  http://www.hollins.edu/admissions/index.html
Page at Rank  9  is  http://www1.hollins.edu/faculty/saloweyca/clas%20395/painting/painting.PPT
Page at Rank  10  is  http://www.hollins.edu/grad/apply/registration.pdf
```

Figure 8: Top 10 websites after stationary distribution of PageRank

3.2.2 Implementation

- *LoadDataset.py* is used to load data from hollins dataset.
- *PageRank.py* is used to find the top 10 pages which uses the functions and classes defined earlier.

3.3 Code

All code is available in GitHub. Here is the link : `https://github.com/dbansal97/IME625A`

4 Reference

1. Karlin Taylor - Introduction to Stochastic Modeling
2. Wikipedia (for PageRank)
3. `http://cecas.clemson.edu/~shierd/Shier/markov.pdf` (for periodicity)