Markov Chain Project Writeup

Rocky Xia

Overview:

The project is a recursive implementation of a three state Markov chain used for weather forecasts based on a set of training data that consists of weather data from the past. The program consists of a State class, a MarkovChain class, and a MarkovChainRunner class that runs the program.

Weather Data:

The weather data used for training is gathered from **https://weather.com**, the training data is gathered from May 2019 through December 2019 in Amherst, MA. The ideal dataset would be the full year of 2019, however, data prior to May 2019 was unavailable. The results of the Markov chain would be compared to the weather data of January 2020 through April 2020 in Amherst, MA for testing. For easy file reading, the weather of each day is represented by a capital letter: S for sunny, C for cloudy, and R for rainy; followed by a single whitespace, in the data files.

Model:

A Markov chain is a mathematical model that has many states. The model would generate a sequence of events, where each one is based on its previous event and a probability weight. This probability weight can be determined through a set of training data. This nature of Markov chains makes it effective in fields that need to make predictions on recurring events, such as meteorology.

Algorithm Implementation:

The algorithm is implemented in Java, and consisting of three classes: State, MarkovChain, and MarkovChainRunner. This implementation is specifically designed for the purpose of reading in the specific formatting of data files (See Weather Data section) and is not capable of generic uses.

State:

The state class consists of a String variable that specifies the name of the state for console UI output, a State array that includes **all** states in the model, and an int array that includes the number of appearances of each state following the state represented by the class object in the training data. For example, in the case of the State class object for cloudy days, for the data segment of “C R S C C”, the appearance count of sunny states would be 0, that of cloudy states would be 1, and that of rainy states would be 1, as each of these states succeeded a cloudy state. Even though the arrays are initialized on construction of the State class object, the specific setup of these arrays are done in the Markov chain class on construction, which would be explained in the following part of this section. The State class also includes a method that returns the next State based on the probability weights derived from the training data. The method would first get the total number of states appearing after the state represented by the State object, and then generate a random positive integer within that range. This random integer would then be compared to the numbers of appearances in the int array that stores state appearances in the training dataset to determine the next state in the Markov chain.

Example:
The int array is defined as follows: 3, 7, 5; each representing the appearance counts of sunny, cloudy, and rainy states, respectively. The method generates a random integer in range of 0 and 15, the total number of the appearances of the states. The integer returned 6. This integer is then compared to the value of the first index, which 3 is less than or equal to 7. Because of this, the value of the first index is subtracted from the generated integer, which reduces it to 3. The resulting integer is then compared to the value of the next index, which 7 is not less than or equal to 3. Therefore the second index is selected and the corresponding state, which is stored in the State array at the same index, is returned.

MarkovChain:
The MarkovChain class consists of three State objects, which are implemented specifically for the application for this project, representing sunny, cloudy, and rainy days. The class includes a constructor, which initializes all three State objects and adds each one into the State array of each other. This sets up a recursive structure where each State object has access to every State object in the Markov chain, including itself, allowing the State objects to return the next state in its class method. The constructor then reads in the file of the weather data and sets up the int arrays that handles probability weights in each State object. The class also includes a method that generates a chain of states of a specified length using the method inside State objects.

MarkovChainRunner:
The runner class is simple, it constructs a MarkovChain object and prompts the user for chain generation through a console UI. The user is able to choose the starting state and the length of the generated sequence of states.

Evaluation:

Through comparing generated sequences of length 10 with similar segments in the testing dataset, the algorithm averages an error value of 2.7, yielding a 73% accuracy in prediction. This is a very good result as the training dataset of the program is rather limited, only including the data of less than a full year of weather records. If the algorithm has access to a larger dataset, it’s expected to have a much lower average error.

Examples:
The data would be displayed in pairs, with the top being the sequence from the testing set, and the bottom being the generated sequence, errors are marked with underscores. See Weather Data section for formatting.

C C S C C S S S C S
C C S C C S S $\underline{C}$ C S

R C C C C C S S R C
R C C C $\underline{S}$ C $\underline{C}$ S $\underline{C}$ C

S C C S S R C C S S
S C C S S R $\underline{R}$ $\underline{S}$ $\underline{C}$ S

Conclusion:

This recursive implementation of the Markov chain yielded a high accuracy in its predictions of future events despite limitations in training data. The program was originally designed to use a structure consisting of arrays and linked lists, however, the approach was not successful due to many development difficulties, as reflected in the development log. The biggest challenge in the development process is implementation itself. Markov chain is a very simple and straightforward concept, but its structuring is rather difficult to represent in code. There have been many challenges throughout the project, which created a great learning experience to think about models in much simpler terms and break them into smaller parts to implement.