Mini Markov

A Python Module for performing Markov Chain Monte Carlo Simulations

The Mini Markov module contains one class, a subclass, and example code. Markov chains are mathematical models of situations that can be represented by a series of discrete states along with a probability table representing the likelihood of transitioning from one state to another.

The markov class is instantiated as

```
import mini_markov as mkv
mk = mkv()
```

The transition table shows the probability that once in a state, one may move to a different (or the same) state. The mini_markov class uses a Pandas dataframe for the transition table. It also requires numpy, math, and random libraries and supports a callback that can be used for a progress bar.

Class mini_markov

__init__(): Initializes the log and output dataframe

__version__(): Returns a string that is the version number of the module

run(transition_table, trials=100, epochs=100, startstate=0, log=None, logtype='none', seed=-1, progress=None): This is the main function of the class. It runs the number of trials specified (trials), each of which can have a maximum number of epochs. Each trial starts in the startstate specified, and a log file may be specified. The logtype specifies if the log will be 'none' (redundant with log=None), text, or csv. The data that will be written to the log will be in the format "Trial n Epoch x, State y" for each epoch if the logtype is 'text'. If logtype is 'csv' then the trial, epoch and state are written out comma separated, one line per epoch. 'seed' is the random number generator seed. If -1 or unspecified the results are randomized. Specifying a seed gives the ability to reproduce a particular outcome. 'progress' specifies a callback that can be used for a progress bar (or anything else) and is called once per trial. Note that a state which has no probability of transitioning to another state is called a terminal state and causes the trial to end, whether epochs has been exhausted or not.

build_df(columns, transitions): This function will build the transition dataframe from an array of arrays (transitions) and a set of column labels (columns). Returns the pandas dataframe.

Class result_set

Parameters

Output: output dataframe from a markov run

trials: number of trials from the markov run

original: original transitions dataframe

cols: original transitions dataframe column names (states)

Methods

counts(): returns a tuple of two dataframes. The first contains the counts of epochs spent in each state. The second contains the counts of epochs per trial for each state

mean_time(): returns a float of the mean time spent per trial

results_range(): returns a dataframe where the rows are the states and the columns are minimum, maximum, mean value, and standard deviation

value(value_array): Takes value_array as a parameter which would be a list of the values (or costs) of each state. Returns a dataframe with the product of the count and the values for each trial. This could be considered the cost or benefit for each trial.

Example:

Included as the __main__ function in the module is the below. This models leaving a hotel room. Note that the Exit state has zero probability of transitioning to anything else. This terminates the trial and moves to the next trial, whether the epochs has been exhausted or not.

```
if __name__ == "__main__":
    import progressbar
    p = progressbar.ProgressBar(maxval=1000).start()
    def callbac(progress):
        p.update(progress)
        return
    tmatrix = [[0.8,0.2,0,0],[0.25,0.5,0.25,0],[0,0.2,0.5,0.3],[0,0,0,1]]
    names = ['Room', 'Hall', 'Lobby', 'Exit']
    m = mini_markov()
    df = m.build_df(names, tmatrix)
    rset = m.run(df,trials=1000,epochs=100,startstate=0,seed=42, log='test.csv',
logtype = 'csv',progress=callbac)
    p.finish()
    print(f'Time spent in each state:\n{rset.counts()[0]} \n{rset.counts()[1]}')
    print(f'Average time through system: {rset.mean_time()}')
    print(rset.results_range())
    values = [20,35,30,10]
    print(rset.value(values))
```