

HW9\Sample_Casino.py

```
1 # Import Statements
2 import numpy as np
3 from scipy import stats
4 from tqdm import tqdm
5 import matplotlib.pyplot as plt
6 import click
7
8 # Creating R.V. X, Y
9 # For Y, YF and YC are for the outcomes of fair and cheating rolls
10
11 class hidden_cas():
12     def __init__(self,a,YF,YC,T,b,n):
13         # a is probability of changing casino state
14         self.a = float(a)
15         self.YF = stats.rv_discrete(name = "YF",values = ((1,2,3,4,5,6), YF))
16         self.YC = stats.rv_discrete(name = "YC",values = ((1,2,3,4,5,6), YC))
17         # Case 0 = Fair, 1 = Cheat
18         self.cas_start = 0
19         self.burnin = b
20         self.n_iter = n
21         self.T = T
22         self.M = np.asmatrix([[1-self.a, self.a],[self.a, 1-self.a]])
23         self.emission = np.asmatrix([YF,YC])
24         self.xc,self.yc = self.build_instances()
25         self.mcmc_state = self.mcmc()
26         self.alphaF, self.alphaC = self.forward()
27         self.betaF,self.betaC = self.backward()
28         self.Z = self.find_Z()
29
30     def build_instances(self):
31         # For part a
32         # Take initial state, do t iterations
33         # xc and yc are states and rolls at each step i
34         xc = [self.cas_start]
35         y0 = self.YF.rvs(1)-1
36         yc = [y0]
37         cstate = 0
38         for i in tqdm(range(self.T)):
39             p = np.random.uniform(0,1)
40             if p < self.a:
41                 cstate = 1-cstate
42             xc.append(cstate)
43             if cstate == 0:
44                 yc.append(self.YF.rvs(1)-1)
45             elif cstate == 1:
46                 yc.append(self.YC.rvs(1)-1)
47         return xc,yc
48
49     def find_Z(self):
50         Z = np.dot(self.alphaF,self.betaF)+np.dot(self.alphaC,self.betaC)
51         return Z
52
```

```
53     def sample(self,t):
54         return self.xc[t-1],self.yc[t-1]
55
56     def mcmc_flip(self,cstate,ostate):
57         cstate = cstate
58         pstate = 1-cstate
59         ostate = ostate
60         if np.random.uniform(0,1) < self.M[ostate,pstate]:
61             cstate = pstate
62         return cstate
63
64     def prob_cheat(self,chain):
65         cc = chain.count(1)
66         return cc/(len(chain))
67
68     def mcmc_prob(self,chain):
69         p = 1
70         for i in range(len(chain)):
71             p *= self.M[chain[i-1], chain[i]]*self.emission[chain[i],self.yc[i]-1]
72         return p
73
74
75     def mcmc(self):
76         cstate = 0
77         mcmcstates = [0]
78         for i in tqdm(range(self.T)):
79             p = np.random.uniform(0,1)
80             if p < self.a:
81                 cstate = 1-cstate
82             mcmcstates.append(cstate)
83
84         for i in tqdm(range(self.burnin)):
85             mcstp = mcmcstates.copy()
86             r = np.random.randint(1,200)
87             if r == 1:
88                 mcstp[0] = 0
89             else:
90                 mcstp[r-1] = self.mcmc_flip(mcmcstates[r-1],mcmcstates[r-2])
91
92             acceptance = min(1,(self.mcmc_prob(mcstp)/self.mcmc_prob(mcmcstates)))
93
94             if np.random.uniform(0,1) < acceptance:
95                 mcmcstates = mcstp
96
97         for i in tqdm(range(self.n_iter)):
98             mcstp = mcmcstates.copy()
99             r = np.random.randint(1,200)
100             if r == 1:
101                 mcstp[0] = 0
102             else:
103                 mcstp[r-1] = self.mcmc_flip(mcmcstates[r-1],mcmcstates[r-2])
104
105             acceptance = min(1,(self.mcmc_prob(mcstp)/self.mcmc_prob(mcmcstates)))
106
107             if np.random.uniform(0,1) > acceptance:
```

```
108         mcmcstates = mcstp
109     return mcmcstates
110
111     def forward(self):
112         alphaF = [self.YF.pmf(self.yc[0])]
113         alphaC = [0]
114         for i in tqdm(range(self.T)):
115             aF = self.M[0,0]*alphaF[i-1]*self.emission[0,self.yc[i]-1] + self.M[0,1]
116             *alphaC[i-1]*self.emission[1,self.yc[i]-1]
117             aC = self.M[1,0]*alphaF[i-1]*self.emission[0,self.yc[i]-1] + self.M[1,1]
118             *alphaC[i-1]*self.emission[1,self.yc[i]-1]
119             alphaF.append(aF)
120             alphaC.append(aC)
121
122         return alphaF,alphaC
123
124     def backward(self):
125         betaFd = [1]
126         betaCd = [1]
127         for i in tqdm(range(self.T)):
128             bF = self.M[0,0]*betaFd[i-1]*self.emission[0,self.yc[i]-1] + self.M[0,1]
129             *betaCd[i-1]*self.emission[1,self.yc[i]-1]
130             bC = self.M[1,0]*betaFd[i-1]*self.emission[0,self.yc[i]-1] + self.M[1,1]
131             *betaCd[i-1]*self.emission[1,self.yc[i]-1]
132             betaFd.append(bF)
133             betaCd.append(bC)
134             betaF = betaFd[::-1]
135             betaC = betaCd[::-1]
136         return betaF, betaC
137
138     def t_is_cheat(self,t):
139         return self.betaC[t-1]*self.alphaC[t-1]*(1/self.Z)
140
141     def plots(self):
142         z = []
143         yfb = []
144         ya = []
145         ymcmc = []
146         ymcmcs = []
147         for i in tqdm(range(self.T)):
148             z.append(i)
149             yfb.append(self.t_is_cheat(i-1))
150             ya.append(self.xc[i-1])
151             ymcmc.append(self.prob_cheat(self.mcmc_state[0:i+1]))
152             ymcmcs.append(self.mcmc_state[i-1])
153
154         fig, ax1 = plt.subplots()
155
156         ax2 = ax1.twinx()
157         ax1.plot(z,yfb,'b*',label = "Forward/Backward")
158         ax2.plot(z,ya,'r--',label = "Actual States")
159         ax2.plot(z,ymcmc,'g o',label = "MCMC")
160         ax1.set_xlabel('Time')
161         ax1.set_ylabel('Probability of Cheating')
162         ax2.set_ylabel('State in True Chain')
```

```
159     ax1.legend(loc='upper right', bbox_to_anchor=(0.5, 1.15), fancybox=True, shadow=
True)
160     ax2.legend(loc='upper left', bbox_to_anchor=(0.5, 1.15), fancybox=True, shadow=
True)
161     ax1.set_title("Comparing F/B, MH predictions to Actual Casino")
162     plt.savefig('hidden_casino.png')
163
164     fig2, l2 = plt.subplots()
165     l2.plot(z,ya, 'r--',label = "State in True Chain")
166     l2.plot(z,ymcmcs,'b--',label = "MCMC States")
167     l2.legend(loc='upper right', bbox_to_anchor=(0.5, 1.20), fancybox=True, shadow=
True)
168     l2.set_title("Comparing MCMC State to True State")
169     plt.savefig('mcmc_compare.png')
170
171
172 @click.command()
173 @click.option(
174     '--a',
175     type = float,
176     default=.05,
177     show_default=True,
178     help='Probability of switching between Cheat and Fair'
179 )
180 @click.option(
181     '--yf',
182     type = np.array,
183     default=(1/6,1/6,1/6,1/6,1/6,1/6),
184     show_default=True,
185     help='Probability of Fair Die'
186 )
187 @click.option(
188     '--yc',
189     type = np.array,
190     default=(19/100,19/100,19/100,19/100,19/100,1/20),
191     show_default=True,
192     help='Probability of Cheat Die'
193 )
194 @click.option(
195     '--T',
196     default=200,
197     show_default=True,
198     help='T number of observed rolls'
199 )
200 @click.option(
201     '--T',
202     default=200,
203     show_default=True,
204     help='T number of observed rolls'
205 )
206 @click.option(
207     '--b',
208     default=10000,
209     show_default=True,
210     help='Burn in iterations'
211 )
```

```
212 @click.option(  
213     '--n',  
214     default=200,  
215     show_default=True,  
216     help='MCMC iterations'  
217 )  
218  
219  
220 def main(a,yf,yc,t,b,n):  
221     casino = hidden_cas(a,yf,yc,t,b,n)  
222     casino.plots()  
223  
224  
225 if __name__ == "__main__":  
226     plt.ion()  
227     main()  
228  
229
```