CS589: Machine Learning - Fall 2017

Homework 4: Bayesian Inference

Assigned: Monday, Nov. 6^{th} Due: Monday, Nov 20^{th}

Getting Started: In this assignment, you will use Bayesian methods to make inference on a toy dataset. **Please install Python 3.6 via Anaconda on your personal machine**. Download the homework file HW04.zip via Moodle. Unzipping this folder will create the directory structure shown below,

```
HW04
--- HW04.pdf
--- Data
    |--B_sequences_10.txt
    |--B_sequences_15.txt
    |--B_sequences_20.txt
    |--B_sequences_25.txt
--- Submission
    |--Code
    |--Figures
    |--Predictions
```

The data files are in 'Data' directory respectively. You will write your code under the Submission/Code directory. Make sure to put the deliverables (explained below) into the respective directories.

Deliverables: This assignment has three types of deliverables:

- **Report:** The solution report will give your answers to the homework questions (listed below). Try to keep the maximum length of the report to 5 pages in 11 point font, including all figures and tables. Reports longer than five pages will only be graded up until the first five pages. If you have answered extra credit questions then you can exceed the five page limit but make sure put you solutions to extra credit questions on pages six and after only. You can use any software to create your report, but your report must be submitted in PDF format.
- **Code:** The second deliverable is the code that you wrote to answer the questions, which will involve implementing a MCMC algorithm to draw samples. Your code must be in Python 3.6 (no iPython notebooks or other formats). You may create any additional source files to structure your code. However, you should aim to write your code so that it is possible to reproduce all of your experimental results exactly by running *python run_me.py* file from the Submissions/Code directory.
- **Kaggle Submissions:** We will use Kaggle, a machine learning competition service, to evaluate the performance of your MCMC algorithm. You will need to register on Kaggle using a *.edu email address to submit to Kaggle. Please use an appropriate user name for us to map your Kaggle submissions to Moodle. You will generate one prediction file, save it in Kaggle format (helper code provided called Code/kaggle.py) and upload to Kaggle for scoring. Your scores will be shown

on the Kaggle leaderboard. The Kaggle link for this HW is `https://www.kaggle.com/t/ad84d5932cbc4b14bb1c69b45c6e7230`.

Submitting Deliverables: When you complete the assignment, you will upload your report and your code using the Gradescope.com service. Place your final code in Submission/Code. If you generated any figures place them under Submission/Figures. Finally, create a zip file of your submission directory, Submission.zip (NO rar, tar or other formats). Upload this single zip file on Gradescope as your solution to the 'HW04-Bayesian-Programming' assignment. Gradescope will run checks to determine if your submission contains the required files in the correct locations. Finally, upload your pdf report to the 'HW04-Bayesian-Report' assignment. When you upload your report please make sure to select the correct pages for each question respectively. Failure to select the correct pages will result in point deductions. The submission time for your assignment is considered to be the later of the submission timestamps of your code, report and Kaggle submissions.

Academic Honesty Statement: Copying solutions from external sources (books, internet, etc.) or other students is considered cheating. Sharing your solutions with other students is also considered cheating. Posting your code to public repositories like GitHub, stackoverflow is also considered cheating. Any detected cheating will result in a grade of -100% on the assignment for all students involved, and potentially a grade of F in the course.

Pseudocode: Given below is the pseudocode for Metropolis Hastings algorithm to sample from $P(\alpha, J|B)$ using a symmetric proposal distribution. To evaluate $P(\alpha|J, B)$, only α values should be sampled and for $P(J|\alpha, B)$, only J states should be sampled.

Algorithm 1 Metropolis Hastings

```
 1: function $MCMC(J, B, \alpha, iterations)$
 2:     for $i = 1$ to $iterations$ do
 3:         $\alpha_{\text{mean}} = 0$
 4:         $J_{\text{mean}} = (0, 0, ..., 0)$
 5:         $(\alpha_{new}, J_{new})$ = proposal$(\alpha, J)$
 6:         acceptance_ratio = $\frac{P(\alpha_{new}, J_{new}, B)}{P(\alpha, J, B)}$
 7:         if rand(0,1) $\leq$ acceptance_ratio then
 8:             $\alpha = \alpha_{new}$
 9:             $J = J_{new}$
10:         end if
11:         $\alpha_{\text{mean}} += \alpha$
12:         $J_{\text{mean}} += J$
13:     end for
14:     return $(J_{\text{mean}}/iterations, \alpha_{\text{mean}}/iterations)$
15: end function
```

Problem: Consider 2 jars Jar$_0$ and Jar$_1$ each containing black and white balls. Jar$_0$ contains 20% white balls and 80% black balls. Jar$_1$ contains 90% white balls and 10% black balls. At time one, Jar$_1$ is used, and a ball is drawn from it. Next, with probability α, we keep the same jar at the next time step, and draw another ball from it. With probability $1 - \alpha$, we switch to the other jar and draw a ball from it. This continues for n

time steps, so that n balls have been drawn. Formally the model can be defined as follows:

α is the transition parameter where $0 \leq \alpha \leq 1$.
J = $(J_1, J_2, ..., J_n)$ is the sequence of jar states where J_i ∈(0,1) for $i = 1...n$. Here, 0 refers to Jar$_0$ and 1 refers to Jar$_1$
B = $(B_1, B_2, ...B_n)$ is the sequence of ball states where $B_i \in (0, 1)$ for $i = 1...n$. Here, 0 refers to a white ball and 1 refers to a black ball.
Example: $J = (0, 0, 1)$ and $B = (0, 1, 1)$ implies that a white ball was drawn from Jar$_0$ followed by a black ball from Jar$_1$ and then a black ball from Jar$_1$.

The joint probability of the model can be written as

$$P(\alpha, J, B) = P(\alpha)P(B|J)P(J|\alpha)$$

where $P(\alpha)$ is the prior probability and $P(B|J) \times P(J|\alpha)$ is the likelihood $P(B|\alpha)$.

Given this model and input ball states B, the objective would be to infer jar states J and posterior probability α using the Metropolis Hastings algorithm. Specifically, we will draw a set of samples from $P(J, \alpha|B)$.

a. (*6 pts*) Implement a function to calculate $P(J|\alpha)$ with input arguments J, α and output $P(J|\alpha)$. $P(J|\alpha)$ can be evaluated as follows:

$$P(J|\alpha) = I[J_1 = 0] \prod_{i=1}^{n-1} X_i$$

where

$$X_i = \begin{cases} \alpha, & \text{if } J_i = J_{i+1} \\ 1 - \alpha, & \text{if } J_i \neq J_{i+1} \end{cases}$$

1. List your python code snippet to compute $P(J|\alpha)$ (No imports, main routines, etc)
2. Test your implementation using the following two test cases,
   Case 1 : Input : J=(0,1,1,0,1) , α=0.75 , Output : $P(J|\alpha)$ = 0.011718
   Case 2 : Input : J=(0,0,1,0,1) , α=0.2 , Output : $P(J|\alpha)$ = 0.1024
3. Provide your code outputs for,
   Case 3 : Input : J=(1,1,0,1,0,1) , α=0.2
   Case 4 : Input : J=(0,1,0,1,0,0) , α=0.2

b. (*6 pts*) Implement a function to calculate $P(B|J)$ with input arguments J, B and output $P(B|J)$. $P(B|J)$ can be evaluated as follows:

$$P(B|J) = \prod_{i=1}^{n} P(B_i|J_i)$$

1. List your python code snippet to compute $P(B|J)$ (No imports, main routines, etc)

2. Test your implementation using the following two test cases,
   Case 1 : Input : J=(0,1,1,0,1) , B = (1,0,0,1,1) , Output : $P(B|J)$ = 0.05184
   Case 2 : Input : J=(0,1,0,0,1) , B = (0,0,1,0,1) , Output : $P(B|J)$ = 0.00288

3. Provide your code outputs for,
   Case 3 : Input : J=(0,1,1,0,0,1), B = (1,0,1,1,1,0)
   Case 4 : Input : J=(1,1,0,0,1,1), B = (0,1,1,0,1,1)

c. (*2 pts*) Implement a function to calculate $P(\alpha)$ with input argument α and output $P(\alpha)$. $P(\alpha)$ can be evaluated as follows:

$$P(\alpha) = \begin{cases} 1, & \text{if } 0 \leq \alpha \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

1. List your python code snippet to compute $P(\alpha)$ (No imports, main routines, etc)

d. (*6 pts*) Implement a function to calculate $P(\alpha, J, B)$ with input arguments J, B, α and output $P(\alpha, J, B)$. $P(\alpha, J, B)$ can be evaluated as follows:

$$P(\alpha, J, B) = P(\alpha)P(B|J)P(J|\alpha)$$

The previously implemented functions for $P(\alpha)$, $P(B|J)$ and $P(J|\alpha)$ can be used here to get the final value.

1. List your python code snippet to compute $P(\alpha, J, B)$ (No imports, main routines, etc)

2. Test your implementation using the following two test cases,
   Case 1 : Input : J=(0,1,1,0,1) , B = (1,0,0,1,1) , α=0.75 , Output : $P(\alpha, J, B)$ = 0.0006075
   Case 2 : Input : J=(0,1,0,0,1) , B = (0,0,1,0,1) , α=0.3 , Output : $P(\alpha, J, B)$ = 0.0002963

3. Provide your code outputs for,
   Case 3 : Input : J=(0,0,0,0,0,1), B = (0,1,1,1,0,1), α=0.63
   Case 4 : Input : J=(0,0,1,0,0,1,1), B = (1,1,0,0,1,1,1), α=0.23

e. (*2 pts*) Implement a function to generate a new proposed value for J with input argument J and output J_{new}. This is calculated by randomly selecting a J_i and flipping its value.

Example: Input : J=(0,1,1,0,1) , Possible Outputs : J=(1,1,1,0,1) where J_1 was flipped, J=(0,0,1,0,1) where value of J_2 was flipped, etc. Note that this function is random, and that calling it repeatedly should generate all possible flippings.

1. List your python code snippet that proposes new J (No imports, main routines, etc)

f. (*10 pts*) Implement a function for Metropolis Hastings algorithm to draw samples from $P(J|\alpha, B)$ and then calculate the mean value of J in those samples. Use input arguments B, α, $iterations$ and output $P(J|\alpha, B)$. Run the algorithm for 10,000 iterations. In order to report the mean $P(J|\alpha, B)$ you will need to keep track of proposed J's over iterations like in line 12 of Algorithm 1 and finally divide by the number of iterations.

1. List your python code snippet to draw samples from $P(J|\alpha, B)$ (No imports, main routines, etc)

2. Test your implementation using the following test case,
   Case 1 : Input : B = (1,0,0,1,1) , α=0.5 , Output : $P(J_2 = 1|\alpha, B) \approx$ 0.8

3. Draw a bar chart for $P(J_2|\alpha, B)$ for the test case given above.

4. Provide your code outputs for J as a vector rounded to four decimal places for,
   Case 2 : Input : B = (1,0,0,0,1,0,1,1), α=0.11
   Case 3 : Input : B = (1,0,0,1,1,0,0), α=0.75

g. (*2 pts*) Implement a function to generate a new proposed value for α with input argument α and output α_{new}. This is calculated by selecting a new α value uniformly at random. We acknowledge that the input α has no effect and serves as a dummy input only.

1. List your python code snippet that proposes new α (No imports, main routines, etc)

h. (*10 pts*) Implement a function for Metropolis Hastings algorithm to draw samples from $P(\alpha|J, B)$ and then calculate the mean value of α in those samples. Use input arguments J, B, $iterations$ and output $P(\alpha|J, B)$. Run the algorithm for 10,000 iterations. In order to report the mean $P(\alpha|J, B)$ you will need to keep track of proposed α's over iterations like in line 11 of Algorithm 1 and finally divide by the number of iterations.

1. List your python code snippet to draw samples from $P(\alpha|J, B)$ (No imports, main routines, etc)

2. Test your implementation using the following test case,
   Case 1 : Input : J=(0,1,0,1,0) , B = (1,0,1,0,1) , Output : α_{mean} would have a value close to 0.16
   Case 2 : Input : J=(0,0,0,0,0) , B = (1,1,1,1,1) , Output : α_{mean} would have a value close to 0.81

3. Case 3 : Draw a histogram of $P(\alpha|J, B)$ for the input J=(0,1,1,0,1) , B = (1,0,0,1,1) over 10,000 iterations

4. Provide your code outputs for $P(\alpha|J, B)$ as a vector rounded to four decimal places for,
   Case 4 : Input : J=(0,1,1,1,1,1,1,0), B = (1,0,0,1,1,0,0,1)
   Case 5 : Input : J=(0,1,1,0,1,0), B = (1,0,0,1,1,1)

i. (*2 pts*) Implement a function to generate a proposed values for α and J with input argument α, J and output α_{new}, J_{new}. This is performed by invoking the proposal functions for α (1g) and J (1e) independently.

j. (*16 pts*) Implement a function for Metropolis Hastings algorithm to draw samples from $P(\alpha, J|B)$ and then calculate the mean values of α and J in those Samples. Use input arguments B, $iterations$ and output $P(J|\alpha, B)$, $P(\alpha|J, B)$. Run the algorithm for 10,000 iterations. Again here you will need to keep track of J's and α's and return the mean.

1. List your python code snippet to draw samples from $P(\alpha, J|B)$ (No imports, main routines, etc)

2. For the test case : Input : B=(1,1,0,1,1,0,0,0) make,
   1. Bar charts for $P(J|B)$. Your bar chart should have eight bars corresponding to the mean value of the eight jars
   2. Histogram for $P(\alpha|B)$. Your histogram should be the distribution of α over 10,000 iterations
   3. Plot α as a function of iterations

3. Use these three figure to write at most three sentences about using MCMC algorithm for this specific sequence of balls.

In the next part of this assignment we are going to predict which ball is most likely to arrive next given a sequence of balls. For example given a sequence of balls as $B = (0, 0, 0, 1)$ we would like to predict what would be the color of the 5^{th} ball in this sequence. Again, we are going to use our MCMC algorithm to draw some conclusions on the color of $N + 1^{th}$ ball. Specifically, we are interested in computing what is the probability that $N + 1^{th}$ is black as $P(B_{n+1} = 1|B_1, B_2, ..., B_n)$. In order to answer this question we will need to write two additional functions.

k. (*7 pts*) Implement a function to return the probability of a black ball in $N + 1^{th}$ jar given the N^{th} jar and α only. This can be computed exactly as,

$$f(J_n, \alpha) = P(B_{n+1}|J_n, \alpha) = \sum_{J_{n+1}} P(J_{n+1}|J_n, \alpha)P(B_{n+1} = 1|J_{n+1}) \tag{1}$$

where, $P(J_{n+1}|J_n, \alpha)$ is α if jars J_{n+1}, J_n are same and $1 - \alpha$ if jars J_{n+1}, J_n are different. $P(B_{n+1} = 1|J_{n+1})$ is $(0.8, 0.1)$ for jars 0 and 1 respectively.

1. List your python code snippet to return the probability of a black ball given α and J_n (No imports, main routines, etc)

2. Test your implementation using the following test cases,
   Case 1 : Input : $\alpha = 0.6$, $J_n = 1$, Output: $P(B_{n+1}|J_n, \alpha) = 0.38$
   Case 2 : Input : $\alpha = 0.99$, $J_n = 0$, Output: $P(B_{n+1}|J_n, \alpha) = 0.793$

3. Provide your code outputs for $P(B_{n+1}|J_n, \alpha)$ rounded to six decimal places for,
   Case 3 : Input : $\alpha = 0.33456$, $J_n = 0$
   Case 4 : Input : $\alpha = 0.5019$, $J_n = 1$

l. (*5 pts*) Implement a function to draw samples using Metropolis Hastings algorithm from $P(J, \alpha|B)$ and then perform an exact computation of $P(B_{n+1}|J_n, \alpha)$ for each of those samples. Use input arguments $B, iterations$ and output $P(B_{n+1}|J_n, \alpha)$. Run the algorithm for 10,000 iterations. Since black balls are set to 1, your returned mean is essentially computing the probability of $N + 1^{th}$ ball is black. This piece of code should be identical to the Algorithm 1 except for the fact that you will gather and return the mean of the $P(B_{n+1}|J_n, \alpha)$ (versus gathering and returning the mean of α and J). You can do this by combining your code from 1j and 1k.

1. List your python code snippet to compute the probability of $N + 1^{th}$ ball is black (No imports, main routines, etc)

2. Test your implementation using the following test cases (ball park only),
   1. B = (0, 0, 1, ?); $P(B_{n+1} = 1|B_1, B_2, ..., B_n) \approx 0.58$
   2. B = (0, 1, 0, 1, 0, 1, ?) $P(B_{n+1} = 1|B_1, B_2, ..., B_n) \approx 0.67$
   3. B = (0, 1, 0, 0, 0, 0, 0, ?) $P(B_{n+1} = 1|B_1, B_2, ..., B_n) \approx 0.69$
   4. B = (1, 1, 1, 1, 1, ?) $P(B_{n+1} = 1|B_1, B_2, ..., B_n) \approx 0.65$

m. (*26 pts*) This question is the same the previous question but with ball sequences of lengths 10, 15, 20 and 25 respectively. For each length there is a total of five sequences each. Hence your task is to predict the probability of $P(B_{n+1} = 1|B_1, B_2, ..., B_n)$ for each of twenty sequences, where $N + 1$ is 11, 16, 21 and 26 respectively.

The twenty sequences are organized into four files in the Data directory. There is helper code to read them in 'run_me.py' file. Your task is to predict $P(B_{n+1} = 1|B_1, B_2, ..., B_n)$ and create a single vector of length twenty corresponding to the twenty sequences. You will need to kaggleize (helper code given in Code/Kaggle.py) your predictions and submit to Kaggle using the URL above. Also, make sure to write your predictions to best.csv under the Predictions directory.

Unlike previous assignments there is **no** private leaderboard in this competition. Hence all twenty examples are in the public leaderboard. The best result you can achieve is a mean squared error of zero when your predictions match exactly to the solution. Do not panic if there is a discrepancy of ~$1e - 3$ or smaller, this is bound to happen since we are using a simple MCMC algorithm to arrive at probabilities. For this problem you can experiment with the number of iterations in the range of 10^4 to 10^6. Additionally, you can use the analysis above (convergence to stationary distribution, multiple runs with different number of iterations, etc) with the goal of getting the MSE close to zero.

1. Describe what you tried to get the best MSE on these twenty sequences. Make sure your result on Kaggle should match what is in your best.csv. Also, list your Kaggle user name so we can match your leaderboard scores to your name.

Extra Credit: Finally, here are some extra-credit problems. These are far more difficult than the above problems and have very small point values. These are also deliberately more open-ended, leaving you more space for creativity. As a result, you will need to carefully describe exactly what you did for each problem. To maximize your score with limited time, you should make sure the above problems are done thoroughly and ignore these. We will be very stingy in giving credit for these problems– do them only for the glory, and only at your own risk!

1. (*5 points*) **Extra Credit:** Try different (symmetric) proposal distributions to generate candidates for J and α. Your goal would be to come up with a proposal distribution such that the samples converge to the true value more quickly than with the values above. Describe what your proposal distribution is, and give evidence that the samples are converging more quickly. (E.g., first run the sampler with a large number of iterations to get the correct mean values of J and α. Then, run Metropolis Hastings sampling with different proposal distributions and look at how quickly the computed mean values converge to the true ones.