CS 274A: Homework 3

Probabilistic Learning: Theory and Algorithms, CS 274A, Winter 2025

Due: 6pm Friday February 14th, submit via Gradescope

Instructions and Guidelines for Homeworks

- Please answer all of the questions and submit your solutions to Gradescope (either hand-written or typed are fine as long as the writing is legible).
- All problems are worth equal points (10 points) unless otherwise stated. All homeworks will get equal weight in computation of the final grade for the class (with lowest-scoring homework being dropped).
- The homeworks are intended to help you better understand the concepts we discuss in class. It is important that you solve the problems yourself to help you learn and reinforce the material from class. If you don't do the homeworks you may have difficulty in the exams later in the quarter.
- In problems that ask you to derive or prove a result you should submit a complete mathematical proof (i.e., each line must follow logically from the preceding one, without "hand-waving"). Be as clear as possible in explaining your notation and in stating your reasoning as you go from line to line.
- If you can't solve a problem, you can discuss the high-level concepts *verbally* with another student (e.g., what concepts from the lectures or notes or text are relevant to a problem). However, you should not discuss any of the details of a solution with another student. In particular, do not look at (or show to any other student) *any written material* directly related to the homeworks, including other students' solutions or drafts of solutions, solutions from previous versions of this class, etc. The work you submit should be your own original work.
- If you need to you can look up standard results/definition/identities from textbooks, class notes, textbooks, other reference material (e.g., from the Web). If you base any part of your solution on material that we did not discuss in class, or is not in the class notes, or is not a standard known result, then you should provide a reference in terms of where the result is from, e.g., "based on material in Section 2.2 in ....." or a URL (e.g., Wikipedia).
- Please read each problem carefully. If you believe there is a typo, or some information is missing, or the problem is unclear, please post a question on the Ed discussion board.

Reading for Homework 3:

- Required Reading: Note Set 4 on Bayesian Learning.
- Optional Reading: Chapter 4.6 in Book 1 of Probabilistic Machine Learning by Kevin Murphy provides more detail on Bayesian methods from a machine learning perspective, and Chapter 3 in Computer-Age Statistical Inference (Efron and Hastie) provides a broad statistical perspective on Bayesian thinking. This reading is purely optional and not required, e.g., may be of interest to students who want to learn more about Bayesian ideas beyond what we will discuss in this course.

Problem 1: Beta-Binomial Model

Consider a Bernoulli parameter θ (e.g., the probability that a coin toss is heads or tails), with a binomial IID likelihood model and with a beta prior, as we discussed in class. Let the number of successes (e.g., "heads") in a dataset be r out of n trials. Say the true unknown value of θ is $\theta = 0.8$. Consider the following 2 data sets:

1. $D_1 : r = 2, n = 2$
2. $D_2 : r = 78, n = 100$

Consider also two different possible priors:

1. $\alpha = 1, \beta = 1$
2. $\alpha = 20, \beta = 20$

1. Generate $2 \times 2 = 4$ plots on a single page, where rows 1 and 2 correspond to datasets D_1 and D_2 respectively, and columns 1 and 2 correspond to the first and second set of priors respectively. Plot the prior and the posterior density for each of the 4 dataset-prior combinations, with the x-axis defined over the range $\theta \in [0, 1]$. Mark on the x-axis on your plot where the true value of $\theta = 0.8$ is located as well as the location of the maximum likelihood estimate. Clearly indicate on your plot what the different curves and marks correspond to. Also comment briefly on what you observe from these plots. A few sentences are fine, e.g., compare the location of the MPE (mean posterior estimate, i.e., mean of $p(\theta|D)$ with the ML estimate.
2. A *posterior credible interval* in Bayesian analysis can be defined as a range for θ that contains a certain fraction of the posterior probability mass. For example we could define a 95% posterior credible interval as the range θ corresponding to the region between $\theta \in [\theta_l, \theta_u]$ where θ_l and θ_u are defined as values to the left and right of which (respectively) are 2.5% of the "tail" probability mass. Using this definition of a credible interval, for each of the 4 combinations of priors and datasets determine what the values of θ_l and θ_u are in terms of defining a 95% posterior credible interval (no need to plot this,

just state what this interval is for each of the 4 cases). Feel free to use whatever method you wish to determine θ_l and θ_u (if you are using Python you may want to take a look at `scipy.stats.beta`; other languages should have similar functionality you can use). Your answer should be accurate to within two decimal places.

You should generate your plots and credible intervals on a computer (rather than hand-drawn, etc), using whatever language you wish, but no need to submit any code.

Problem 2: Bayesian Estimation for the Discrete-Multinomial Model

Consider a data set $D = \{x_1, \ldots, x_N\}$, with $x_i \in \{1, \ldots, K\}$ where the x_i's are conditionally independent draws from a probability distribution for a discrete random variable X taking values x, with parameters $\theta_k = P(x_i = k), 1 \leq k \leq K$, and $\sum_{k=1}^{K} \theta_k = 1$ (i.e., we have a multinomial likelihood for the x_i's). Assume that we have a Dirichlet prior for the parameters θ, where the prior has parameters $\alpha_1, \ldots, \alpha_K$ and $\alpha_k > 0$ and $1 \leq k \leq K$.

1. Prove that the posterior density for $\theta_1, \ldots, \theta_K$ is also Dirichlet and find it's parameters. In your proof it is sufficient to show that the posterior density has the same functional form as a Dirichlet density as a function of the parameters θ.
2. Derive an expression for the maximum a posteriori (MAP) estimate for $\theta_k, 1 \leq k \leq K$. Your solution should be derived from first principles (i.e. using basic calculus to find the mode of the posterior density for θ_k, working from your solution for $P(\theta|D)$ from part 1).

Problem 3: Bayesian Estimation for the Exponential Model

Consider a data set $D = \{x_1, \ldots, x_N\}$, where x_i's are real-valued and $x_i \geq 0$, and where the x_i's are conditionally independent draws from the exponential density $p(x|\lambda) = \lambda e^{-\lambda x}, \lambda > 0$ for $x \geq 0$, with $p(x|\lambda) = 0$ for $x < 0$.

Define a Gamma prior for θ in the form $p(\lambda|\alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda^{\alpha-1} e^{-\beta\lambda}$, where $\alpha > 0$ and $\beta > 0$ are the parameters of the Gamma prior and $\Gamma(.)$ is the Gamma function.

1. Derive an expression for the posterior density for λ and state what type of density this.
2. Derive an expression for the posterior mean and show that it can be written as a convex combination of the prior mean and the maximum likelihood estimate of λ.
3. Derive an expression for the posterior mode.

Problem 4: Bayesian Estimation for the Mean of a Gaussian Model

Consider a data set $D = \{x_1, \ldots, x_N\}$, with real-valued scalar x_i values. Assume that the x_i's are conditionally independent draws from a Gaussian density with unknown mean μ and known variance σ^2. Assume we have a Gaussian prior on μ that has mean μ_0 and variance s^2. Given the facts above, derive the following expressions from first principles, clearly explaining and justifying all of your steps:

$$\mu_N \ = \ \gamma \hat{\mu}_{ML} \ + \ (1 - \gamma)\mu_0$$

and

$$\frac{1}{\sigma_N^2} = \frac{N}{\sigma^2} + \frac{1}{s^2}$$

where μ_N and σ_N^2 are the mean and variance of the posterior density for μ and

$$\gamma = \frac{N s^2}{N s^2 + \sigma^2}.$$

Problem 5: Bayesian Active Learning with Multi-Armed Bandits (40 points)

An active area of machine learning research is multi-armed bandits. In this problem we will see how Bayesian modeling can be very useful in this context. We will only explore one small aspect of multi-armed bandit problems: there is a large literature in machine learning on this topic in general.

Introduction and Algorithm Description

Note: it is important to read through the instructions below in detail and to understand the concepts before starting to write code.

Consider a setup where we have a multi-arm "bandit" with K "arms." You can think of each arm as being an arm that we can pull for a slot machine (the "bandit", as in gambling). Whenever we pull a particular arm k we get a stochastic reward $y_k \in \{0, 1\}$ that is drawn from some unknown Bernoulli distribution θ_k, where we have binary random variables Y_k (one per arm) with (unknown) parameters $\theta_k = P(y_k = 1)$, $k = 1, \ldots, K$.

The bandit problem we will investigate is how to sequentially explore/select arms to try to identify which of the arms has the highest expected reward, i.e., the goal is to identify $\arg\max_k\{\theta_k\}$ as quickly and accurately as we can. This problem shows up in a variety of real-world problems in areas like reinforcement learning, active learning, medical clinical trials, online advertising, and so on.

In general we will consider N sequential trials indexed by $i = 1, \ldots, N$. At the start of each trial i we select one arm k: we then "pull" that arm and we get a random y_k outcome (0 or 1) drawn from the true θ_k for the arm we pulled. We only pull one arm per trial; we then move on to trial $i + 1$, select one of the K arms based on the information available up to that point, and pull that arm; and so on.

In terms of notation, after we have conducted trial i, $1 \leq i \leq N$ we will have 2 numbers for each of the K arms. The first number is $n_k^{(i)}$, the number of times that arm k has been "pulled" (selected in a trial) up to and including trial i. The second number is $r_k^{(i)}$, the number of times that outcome $y_k = 1$ has been observed up to and including trial i. In general $0 \leq r_k^{(i)} \leq n_k^{(i)} \leq i$.

We will investigate three different algorithms for selecting an arm k at each trial and we will explore how they differ:

1. **Random**: At each trial i this algorithm selects an arm uniformly at random from the K arms. This is a baseline method that (intuitively) we should be able to improve on with cleverer algorithms.

2. **Greedy**: At each trial i this algorithm selects the arm k that has the highest mean posterior estimate (MPE) across the K arms, i.e., select $\arg\max_k\{\theta_k^{MPE(i-1)}\}$ where each $\theta_k^{MPE(i-1)}$ is the MPE of θ_k for arm k based on the previous $i-1$ trials, i.e.,

$$\theta_k^{MPE(i-1)} = \frac{r_k^{(i-1)} + \alpha_k}{n_k^{(i-1)} + \alpha_k + \beta_k}$$

   where α_k and β_k are parameters of a Beta prior for each θ_k. If there is a tie among multiple arms in the equation above, then select one of the tied arms uniformly at random.

3. **Thompson Sampling**: This algorithm keeps track of a Beta posterior density for each arm k, where the density is defined before it selects an arm for trial i. The Beta density is a function of the following numbers: $r_k^{(i-1)}, n_k^{(i-1)}, \alpha_k, \beta_k$. At the start of each trial i the algorithm then **samples** θ_k **values**, one from each of the K arms using the current Beta densities for each arm. It then selects the arm k whose **sampled value** has the highest value among the K sampled values. Ties are again broken by randomly selecting between tied values if there is a tie, but since we are sampling continuous-valued θs, the probability of a tie in sampling is near zero.

In general we would like to have an algorithm that can both (i) explore the arms (which Random will do, but Greedy won't do enough of), but (ii) also exploit the arms (by focusing on the more promising ones - which Random won't do, and Greedy will sometimes do too much of). Thompson Sampling provides a balance between these two extremes of exploring and exploiting. As you might imagine, the Greedy method can sometimes be quite suboptimal and since it can get "trapped" and not explore the set of arms enough.

The intuition for the sampling aspect of Thompson Sampling is that even if, at trial i, an arm only has a small probability of being the highest reward arm, there is still a chance of it being selected since it might possibly turn out to be the optimal arm in the long run once we see more data (hence, Thompson Sampling encourages some exploration, in a Bayesian fashion). The general idea of Thompson Sampling has some nice theoretical properties (see the discussion in Section 8 of the Russo et al tutorial).

For optional additional reading, for a more complete description of Thompson Sampling for Bernoulli bandits (which is the problem we described above) you can read Sections 1 through 3 of the following tutorial by Russo et al on Thompson Sampling: `https://arxiv.org/pdf/1707.02038.pdf`. Theoretical properties are discussed in Section 8.

Your goal is to implement in code (in any language you wish) the three algorithms, Greedy, Random, and Thompson Sampling and then evaluate these methods using the instructions below.

Experiments across Multiple Runs

You will implement the 3 algorithms and explore how they work on simulated problems. A single "run" consists of N trials: each run can produce different results since the y_k values are stochastic for each trial i, and in addition the Random and Thompson Sampling algorithms have randomness built into how they select arms to pull.

Thus, any single run can be quite noisy due to the stochasticity of the problem. So, to average over this stochastic noise (in terms of evaluating systematic differences between algorithms), you will generate M runs for each algorithm (independently, with different random outcomes for the selected arms each time, and potentially different random selections of arms at each trial), where each run is of length N. Thus, for each of the 3 algorithms you will end up with M runs each with N trials, and sets of numbers $n_k^{(i)}$ and $r_k^{(i)}$, $k = 1, \ldots, K$ arms, and $i = 1, \ldots, N$ trials, for each of the M runs.

Note 1: it is important that the runs are as random as you can make them. It is fine to used a fixed random seed at the very start of your runs (i.e., before you execute any of the M runs): this can be helpful in debugging so that you simulate the same "random" sequence of trials across runs. But don't reset the seed to the same value at the start of each run (since if you do reset the seed, each run will produce the same result, since each run would be using the same sequence of pseudorandom numbers, which is not what we want).

Note 2: since the outcomes y_k are random for any trial i, it doesn't matter how you order the execution of the 3 algorithms within your code. You could for example have an outer loop over the 3 algorithms and then generate all of the results for M runs for one algorithm before generating all results for M runs for the next. Or you could have a loop over $m = 1, \ldots, M$ runs and run each of the 3 algorithms per run before you go to the next run. Or you could loop over each trial $i = 1, \ldots, N$ (for each run), and have the loop over the 3 algorithms be in the inner-most part of the loop. Some versions may be more efficient than others in terms of execution time, but all orderings should produce the same results on average (at least for large M).

Evaluation Methods

To evaluate how good an algorithm is on average after i trials, you can compute the average accuracy (across M runs) after each trial $i = 1, \ldots, N$. The performance metric we will use is the success rate, i.e., the fraction of times (across runs) that an algorithm's estimate of the best arm $\hat{k}^{(i)}$, after i trials, is the same as the true best arm $k^* = \arg\max_k \theta_k$, where the θ_k's are the true parameters for each arm. The true best arm is known to us for evaluation purposes but is not known to the algorithms. For each value i, this type of evaluation measures the accuracy on average if we had stopped after only i trials instead of doing all N, and allows us to plot accuracy as a function of i.

At the end of each trial i, for each of the 3 algorithms we will use the MPE equation from earlier to select the best arm from that algorithm's perspective, at that point in the run, i.e., select $\arg\max_k\{\theta_k^{MPE}\}$. Note that θ_k^{MPE} can in general be different for each of the algorithms since they have each collected different data by selecting different arms along the way, i.e., we will in general have different values for $n_k^{(i)}$ and $r_k^{(i)}$, $k = 1, \ldots, K$ for each of the 3 algorithms.

You can then plot (on the y-axis) the fraction of correct identifications, i.e., how often the estimate of the best arm, for each of the 3 algorithms, agrees with the true best arm, averaged across the M runs, for each value of $i = 1, \ldots, N$ (on the x-axis). The y-axis will range from 0 to 1. The curves will generally start at low values (around $1/K$, i.e., random guessing) for small values of i and then increase (at least for Thompson Sampling and Random) towards 1. The rate at which they approach 1 will depend on the values of θ_k's and will differ between strategies. Greedy will tend to be the noisiest and least predictable across runs.

Note: to keep your code modular and well-organized, one suggestion is to first run the M trials and produce the results ($M \times N \times K$ numbers) for each of the 3 algorithms. Then, given all of this data, you can run separate evaluation code that computes the average accuracy (as a function of i), for each algorithm, for each of the 3 algorithms.

Handling Ties: It is important in your code to break ties randomly. In particular, be aware that the standard "max" or "argmax" function in many programming languages will not break ties randomly, e.g., given $[0.5, 0.2, 0.5, 0.1]$, in selecting which index corresponds to the largest value, many functions will always return the first index value of 1 rather than randomly returning 1 or 3. So you may need to write your own (simple) function to do this (i.e., to break ties randomly).

Priors: For simplicity in all the experiments in this problem we will use uniform (uninformative) prior values of $\alpha_k = \beta_k = 1, k = 1, \ldots, K$.

What to Submit:

1. Submit the plots and comments requested below for this problem to Gradescope (as part of your homework). Use $M = 500$ for all your plots and generate 3 graphs, one for each setting of the θ's below, where each graph has 3 curves (one per algorithm) and the x-axis runs from 1 to N. Feel free to make M much larger (e.g, $M = 10,000$) if you want to get smoother plots and if your code runs fast enough to do more runs.

2. **Please also submit your code** in your programming language of choice. The TA will provide instructions on Ed on where to upload your file. You can put everything in a single Zip file with a single main function or script that runs your code to generate all of the requested plots. Note that your code itself will not be graded directly (only your plots and comments will be graded) but we may look at your code and run it during grading to better understand what you did.

- Let $K = 10$ with $\theta_1 = 0.8, \theta_2 = 0.7, \theta_3 = \theta_4, \ldots = \theta_{10} = 0.5$. For $N = 1000$ trials plot the success rate (in identifying the best arm) for each of the three strategies.

- Repeat part 1 (i.e., generate another plot) but now with $\theta_1 = 0.8, \theta_2 = 0.78, \theta_3 = \theta_4, \ldots = \theta_{10} = 0.5$. Set $N = 3000$ trials.

- Repeat part 1 (i.e., generate another plot) but now with $\theta_1 = 0.8, \theta_2 = \theta_3 = \theta_4, \ldots = \theta_{10} = 0.75$. Set $N = 3000$ trials.
- Comment on (i) the differences in results between the 3 algorithms for each plot, and (ii) the differences in results in general between the 3 plots as the true θ's change.
- Optional: (no extra credit but fun to think about): for the random strategy, can you think of how you might analytically compute the success rate as a function of i (rather than simulating it) if you knew the θ's. You could just consider the special case of $K = 2$ bandits. What insights can you gain from your analysis? i.e., in terms of how hard or easy a particular identification problem is.

Note that you should be able to get your code to run fairly quickly, e.g., in less than a minute for each of the plots being requested, if your code is reasonably efficient. You may want to make sure your code is working first (e.g., try an "easy" problem first for debugging with just $K = 2$ arms and a large difference in the theta values).

Finally, as a sidenote for those of you who may be interested, here is a pointer to a paper in 2021 from my research group where we used ideas from Bayesian estimation, multi-arm bandits, and Thompson sampling to address the problem of assessing the accuracy of black-box classification models `https://arxiv.org/pdf/2002.06532.pdf` (Ji et al, AAAI Conference, 2021).