Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.3800 Introduction to Inference
Fall 2024

Computational Lab II

Issued: Wednesday, September 25 **Due:** Friday, October 11 at 11:59pm

---

Movie Rating

You are done with your psets and decide to reward yourself by watching a movie. But which one? In this lab, you will build a Bayesian inference engine to determine which movie you should watch based on other viewers' ratings of the movies.

Let $X \in \mathcal{X} = \{0, \dots, M-1\}$ be a discrete random variable distributed that represents the true quality of a movie you are considering, where 0 corresponds to awful, and $M-1$ corresponds to amazing. Let $Y_n \in \mathcal{Y} = \{0, \dots, K-1\}$ be a discrete random variable that represents the number of stars user n gave this movie, $n = 0, \dots, N-1$. Given the movie's true quality X, the ratings of different users are independent and identically distributed.

(a) For any $x \in \mathcal{X}$ and $y_0, y_1, \dots, y_{N-1} \in \mathcal{Y}$, express the posterior probability $p_{X|Y_0^{N-1}}(x \mid y_0^{N-1})$ in terms of known distributions p_X and $p_{Y|X}$. There is no coding in this part.

Code and Data. Download file `movie-rating.zip`. from the course website and extract it to your working directory. It contains

- Folder `data` whose contents you need not look at.
- File `movie_data_helper.py`, which you need not edit, but whose functions you will need in your code:
  - `get_movie_id_list()` returns a 1D array containing ID numbers for the movies that you should process. The movie IDs are sequential numbers ranging from 0 to the number of movies included in the input minus 1.
  - `get_ratings(movie_id)` returns a 1D array containing the ratings given by users to the movie `movie_id`. The number of users who rated a given movie might vary, and as a result, the length of the returned vector is different for each movie.
  - `get_movie_name(movie_id)` returns the title of the movie `movie_id`.
- File `movie_recommendations.py` that contains a number of incomplete functions that you will fill in. This is the only file you will edit. You will need to read the comments in this file to understand what code you have to add.

Important: Do not change the function definitions in the code as they must remain consistent for automatic grading. Do not import additional modules since these modules might not exist in the autograder environment. The only parts of the code that you need to fill in are in blocks denoted by "`YOUR CODE GOES HERE`". Feel free to create additional functions as needed to help with the computation or to answer the questions in the lab, although this is not required. See the comment at the beginning of each function you complete for what the expected input and output are.

(b) Implement the function `compute_posterior` in `movie_recommendations.py` to perform the computation you derived in part (a).

Note: Multiplying many small numbers together can produce arithmetic underflow (whereby the computer does not have sufficient precision to store the number). To avoid underflow, you should use logarithmic representations for such numbers. Thus, if you want to compute $q = p_1 \cdot p_2$ from representations $c_1 \triangleq \ln p_1$ and $c_2 \triangleq \ln p_2$, the resulting representation is $d \triangleq \ln q = c_1 + c_2$, i.e., an addition. At the end of all calculations, you can, of course, produce q itself as needed via $q = \mathrm{e}^d$.

Accordingly, you should store the logarithms of your probabilities instead of the probabilities themselves throughout your calculations. In addition to multiplying small probabilities, you will also need to add small probabilities from logarithmic representations. For this purpose, you may find the `numpy` function `logaddexp` helpful for this, though there are also alternative methods you're free to use.

Now you will use the general purpose Bayes' rule calculator to create movie recommendations. In the remainder of this problem, we let $K = M$, i.e., the estimates of the movie's true quality will be on the same scale as the viewers' ratings. We assume a uniform prior

$$p_X(x) = \frac{1}{M} \qquad \text{for } x = 0, \ldots, M-1$$

to reflect our beliefs before we have seen any ratings. Moreover, we use the following likelihood model for ratings given the movie quality:

$$p_{Y|X}(y|x) = \begin{cases} \alpha/|y-x|, & \text{if } y \neq x, \\ 2\alpha, & \text{if } y = x, \end{cases}$$

where $\alpha > 0$ is a normalization constant.

(c) Fill in the function `compute_movie_rating_likelihood` that computes the likelihood $p_{Y|X}(\cdot \mid \cdot)$ above for all possible values of X and Y and returns it as a 2D array.

(d) Complete function `infer_true_movie_ratings` and use it on the provided data to determine the posterior distribution of the true quality of the movies.

Hints: Take advantage of the the function `compute_posterior` that you completed in part (b). You can process each movie separately since we assume that both the quality and the ratings are independent across different movies. This function takes as input the number of observed ratings to use per movie (`num_observations`), which will make the solution to part (g) below easier to implement. For this part, use `num_observations = -1` to utilize all of the available ratings for each movie.

(e) Suppose our goal is to watch a movie with the highest MAP estimate of quality (i.e., $\hat{x}_{\text{MAP}}(y_0^{N-1}) = 10$). Recommend a movie (or movies) based on your results in part (d). Provide the movie titles. Are there any movies that we should avoid based on your results in part (d)?

Finally, we investigate how the posterior distribution of the movie quality changes as the number of ratings available for that movie grows.

(f) Complete the function `compute_entropy` to compute the entropy of a given distribution.

Hint: Do not forget our convention that $0 \cdot \log 0 \triangleq 0$. You can assume that `distribution[j]` $= 0$ if `distribution[j]` $\leq 10^{-8}$.

(g) Complete the function `compute_true_movie_rating_posterior_entropies` to compute the entropy of the posterior distribution $p_{X|Y_0^{N-1}}(\cdot \mid y_0^{N-1})$ for all movies, where the number of observations N to be used is defined by the user.

(h) Plot the entropy of the posterior distribution of the movie quality averaged over all `movie_ids` as a function of N for $1 \leq N \leq 200$. Describe the behavior of average entropy as we receive more and more observations. What does this mean in terms of the remaining randomness in the distribution of the movie quality X after we get more and more observations? Our code for this part finishes in around 15 minutes.

What to hand in: Upload to Gradescope a PDF file with your writeup and file `movie_recommendations.py` with your code. The writeup should include your answers for parts (a), (e), (h), and your plot for part (h) . Scans of handwritten work are fine but please make sure it is legible. We cannot grade what we cannot decipher. Plots should be part of the writeup and not uploaded as separate files.

Gradescope will run some number of tests which are visible to you. These are to make sure your code runs correctly in the Gradescope system. Please make sure you pass these tests.

Optional Part (not graded—just for your enjoyment!)

I liked movie Z, what else might I like?

Thus far we have been considering how to recommend a movie to the whole population, based on estimating X which represents a measure of the overall quality of a given movie. More realistically, a movie's quality, or preference, depends on the user who watches it. If we can estimate a user's preference for a given movie before she watches it, then we would be able to recommend new movies to her. The problem is that a specific user might not have made sufficiently many ratings to reveal her preferences for all movies, so we will need to use the ratings from users with similar preferences to help with this estimation.

In this part of the lab, we give you a particular procedure to make use of ratings from similar users. This procedure implicitly defines a "similar" user in a particular way. Specifically, suppose we have just watched a movie with ID z_0, and we liked it. To find some movies that we would also like, we imagine a hypothetical user who strongly likes movie z_0, and the group of "similar users" as those who also give high ratings to movie z_0. Then, given a candidate movie $z \neq z_0$, we compute the Bayes estimate of its quality as in part (d), except that now include only this filtered group of users in our computation, and thus estimate a group-specific quality. This will become clearer in the following implementation.

(i) Create a new file named `movie_personalization.py` which imports as modules your code `movie_recommendations` and also `movie_data_helper`. Implement functions that, given an inquiry movie z_0,

- Filter the users by only selecting those who gave the inquiry movie z_0 a 9 or 10. (It might take some thought on how to do this efficiently.)
- Produce posterior distributions and MAP ratings for every movie exactly as in the early part of this lab, but now only using ratings from the group of users obtained after the filtering.

(j) Based on the inquiry movie z_0, we want to recommend the movies that the users in this filtered group would most likely give a rating of 10. Accordingly, rank all the movies based on their posterior probability of having a rating of 10. Produce as output the top 5 movies on this list.

Note that this method will sometimes recommend the inquiry movie z_0 as one of the top 5 choices; you should not include z_0 in the recommendations.

(k) Indicate what your code produces as the top 5 movies when your inquiry movie z_0 is

- (id 170) *Casablanca* (1942)
- (id 437) *Lord of the Rings: The Fellowship of the Ring, The* (2001)

- (id 261) *Star Trek II: The Wrath of Khan* (1982)
- (id 262) *Star Trek III: The Search for Spock* (1984)
- (id 179) *Mary Poppins* (1964)
- (id 350) *You've Got Mail* (1998)

Feel free to explore a bit more. Try the procedure on other movies of your choice. How well does this technique work? (Don't feel obligated to, but if you want to share some of your results for this part with us, we will be happy to read what you discover.)