Homework 4: Bayesian & Stochastic Optimization

UC Irvine CS275P: Graphical Models & Statistical Learning

Homework due at 11:59pm on May 13, 2025

Full Name:	**UCINetID (e.g., ucinetid@uci.edu):**

Question 0: (5 points)

Canvas includes detailed guidelines on how homework solutions should be formatted for submission to Gradescope. To receive full credit on this question, be sure that you read and follow these guidelines carefully! In particular, please take care that:

- There are two Gradescope assignments for Homework 4. For the first assignment, submit a PDF containing your answers for questions 1-3. For the second assignment, submit your Python code for question 3, which will be checked by an autograder. *For this assignment, you do not need to submit your code for question 1.*
- For the PDF submission, you must fill in this PDF template with your answers. All pages must remain in their original order when uploading your assignment to Gradescope. *Do not rearrange, add, or remove any pages from the provided PDF template.*
- For multiple-choice questions, ensure that you *completely fill in the bubble next to your selected answer*, and provide an explanation or justification to support your answer.
- Enter your final answers (to a precision of 3 decimal places) in the answer boxes, and write your explanations and/or math justifications in the space provided below each question. *Always show the work needed to produce your answers, not just the final result.* Math may be handwritten, but if possible, please type any sentences.
- For question 3, use the provided Python file. After adding your code to the template, submit that file separately to the second Gradescope assignment. In addition to uploading your Python code, be sure you provide answers and explanations in the designated areas of the PDF. *Do not include code for question 3 in the PDF.*
- If any question asks you to create a plot, insert an image showing the plot in the PDF.
- Check that your answers (especially scanned math) are readable within Gradescope, and that content has not been cut-off by the page margins.

Question 1: (30 points)

Bayesian optimization algorithms use probabilistic models of functions, like Gaussian processes, to capture uncertainty given a limited number of evaluations of a complex objective function. Bayesian optimization is motivated by tasks like hyperparameter tuning for large-scale machine learning methods, where the objective function may take hours or days to evaluate. But so that your work on this assignment doesn't take too long, we will instead find the maximum of the "Branin-Hoo" function, which is fast to evaluate but has a complex shape. The support code evaluates and plots this function.

We will use the Python `BayesianOptimization` package. (Note that unlike the examples in lecture, this package assumes the objective should be *maximized*.) As shown in the support code, you must *always* initialize the optimizer with 2 randomly sampled points. Do this with `bayesian_optimizer.maximize(init_points=2, n_iter=0)`. Gaussian process regression algorithms (and kernel functions) are provided by the `sklearn.gaussian_process` package.

For every case below, run the Bayesian optimization search for 50 steps from 5 different initializations, and plot (with error bars) the maximum function value versus optimization step. The support code shows how to do this, and shows results for a simple baseline that evaluates points randomly, rather than intelligently searching with Gaussian processes.

a) *Let $\mu(x)$ be the mean and $\sigma(x)$ be the standard deviation of the GP predictions. The upper confidence bound (UCB) acquisition function evaluates points where $\mu(x) + \kappa\sigma(x)$ is largest, where $\kappa > 0$ is a hyperparameter. Test the UCB acquisition function with $\kappa = 1$, $\kappa = 5$, and $\kappa = 10$. Are results sensitive to the value of κ? If so, give a 1-2 sentence explanation for why some κ values are less effective.*

Brief discussion and explanation of how and why results vary with κ:

b) The probability of improvement (POI) acquisition function looks for points where the probability that the function improves by at least ξ *(the Greek letter* ***xi****) is largest, where* $\xi > 0$ *is a hyperparameter. Test the POI acquisition function with* $\xi = 0.01$, $\xi = 0.1$, *and* $\xi = 1.0$. *Are results sensitive to the value of* ξ*? If so, give a 1-2 sentence explanation for why some* ξ *values are less effective.*

Brief discussion and explanation of how and why results vary with ξ:

c) *The expected improvement (EI) acquisition function also ignores points whose improvement is less than $\xi > 0$, and further prioritizes points where the magnitude of the expected improvement is largest. Test the EI acquisition function with $\xi = 0.01$, $\xi = 0.1$, and $\xi = 1.0$. Are results sensitive to the value of ξ? If so, give a 1-2 sentence explanation for why some ξ values are less effective.*

Brief discussion and explanation of how and why results vary with ξ:

d) *Fix the acquisition function to EI with $\xi = 0.01$. By default, the Bayesian optimization package uses a Matern kernel with $\nu = 2.5$. The hyperparameter ν (the Greek letter **nu**) controls the amount of smoothness of the GP prior, where large ν leads to smooth functions and small ν leads to rougher functions. Test the Matern kernel with $\nu = 0.5$, $\nu = 2.5$, and $\nu = 10.0$. Are results sensitive to the value of ν? If so, give a 1-2 sentence explanation for why some ν values are less effective.*

Brief discussion and explanation of how and why results vary with ν:

e) Fix the acquisition function to EI with $\xi = 0.01$. Consider a different covariance kernel, the radial basis function (RBF) kernel with lengthscale hyperparameter $\sigma > 0$. Test the RBF kernel with $\sigma = 0.001$, $\sigma = 1.0$, and $\sigma = 3.0$. Are results sensitive to the value of σ? If so, give a 1-2 sentence explanation for why some σ values are less effective.

Brief discussion and explanation of how and why results vary with σ:

Question 2: (24 points)

We now consider a binary categorization problem, where $t_n \in \{0, 1\}$ is the output label for example n, and $x_n \in \mathbb{R}^2$ is a two-dimensional vector of input features. Assume that the two classes are equally likely *a priori*, so that $p(t_n) = \text{Ber}(t_n \mid 0.5)$. Under the true data generation process, the features are distributed according to class-specific Gaussians:

$$p(x_n \mid t_n = 1) = \text{Normal}(x_n \mid \mu_1, \Sigma), \qquad p(x_n \mid t_n = 0) = \text{Normal}(x_n \mid \mu_0, \Sigma).$$

The mean vectors μ_1, μ_0 are discussed below. The shared covariance matrix equals:

$$\Sigma = \frac{4}{3}\begin{bmatrix} 1 & 1/2 \\ 1/2 & 1 \end{bmatrix} = \begin{bmatrix} 4/3 & 2/3 \\ 2/3 & 4/3 \end{bmatrix}, \qquad \Sigma^{-1} = \begin{bmatrix} 1 & -1/2 \\ -1/2 & 1 \end{bmatrix}.$$

a) *Suppose that $\mu_0 = [0, 0]^T$, $\mu_1 = [2, 0]^T$. Given knowledge of the true joint distribution $p(x_n, t_n)$, derive a classification rule $y(x_n)$ which minimizes the probability of error. Plot the corresponding decision boundary graphically.*

Derivation of classification rule:

b) Suppose that $\mu_0 = [0, 0]^T$, $\mu_1 = [2, 2]^T$. Given knowledge of the true joint distribution $p(x_n, t_n)$, derive a classification rule $y(x_n)$ which minimizes the probability of error. Plot the corresponding decision boundary graphically.

Derivation of classification rule:

Now suppose that we do not have knowledge of the true data generating process, but instead assume a naive Bayes model with Gaussian features:

$$\begin{aligned}p(x_n \mid t_n = 1) &= \text{Normal}(x_{n1} \mid \theta_{11}, \nu_{11})\text{Normal}(x_{n2} \mid \theta_{12}, \nu_{12}),\\ p(x_n \mid t_n = 0) &= \text{Normal}(x_{n1} \mid \theta_{01}, \nu_{01})\text{Normal}(x_{n2} \mid \theta_{02}, \nu_{02}).\end{aligned}$$

Consider a training dataset with N observations (x_n, t_n) independently sampled from the true joint distribution $p(x, t)$. In each question below, assume that the parameters θ and ν of the naive Bayes model are estimated via the maximum likelihood (ML) criterion.

c) *Suppose that* $\mu_0 = [0,0]^T$, $\mu_1 = [2,0]^T$. *As* $N \to \infty$, *what classification rule will the naive Bayes classifier approach? Will it be as accurate as the optimal rule from part (a)? Justify your answer and plot the corresponding decision boundary graphically.*

Derivation of classification rule and discussion of accuracy:

d) Suppose that $\mu_0 = [0, 0]^T$, $\mu_1 = [2, 2]^T$. *As* $N \to \infty$, *what classification rule will the naive Bayes classifier approach? Will it be as accurate as the optimal rule from part (b)? Justify your answer and plot the corresponding decision boundary graphically.*

Derivation of classification rule and discussion of accuracy:

Question 3: (21 points)

Stochastic gradient descent (SGD) is a parameter estimation method that processes subsets (batches) of a large dataset at each step. Here, we compare SGD to "full dataset" gradient descent for maximum likelihood estimation of the weight vector for a simple logistic regression binary classifier. Recall that the logistic regression model is defined as

$$p(t_n = 1 \mid x_n, w) = \sigma(w^T x_n), \qquad \sigma(z) = \frac{1}{1 + e^{-z}},$$

where the weight vector $w \in \mathbb{R}^D$ and we have assumed the features are raw inputs x_n.

For this problem, you'll consider 11,791 handwritten digits from the MNIST dataset, each an example of the digit "4" or the digit "9"; see `MNIST_Digits49_19x19.npy`. These have been preprocessed so the labels are either +1 or -1, and so that instead of the original 28x28 pixel image, we have cropped out the center to produce a 19x19 image. This barely affects accuracy, but makes learning speedier.

For parts that require SGD, use the provided function `min_func_sgd` to perform minimization across many batches of data. The demo code provides example usage to get you started. We'll also use `minimize` from the `scipy.optimize` package as a baseline.

a) *As a baseline, use* `minimize` *to estimate a logistic regression weight vector using the entire dataset. Using the estimated weight vector $\hat{w}$, report the accuracy on the train and test sets, as well as the computation time required for convergence.*

Train accuracy:	
Test accuracy:	
Computation time:	

b) Train logistic regression models using SGD with three different divisions of the data into batches: 1 batch, 100 batches, and 11,791 batches (this last setting processes each example one at a time). Fix the step size schedule $\eta_t = (10 + t)^{-0.51}$. Run the estimation for 30 passes through the full dataset (one pass is called an "epoch"). Create two figures: one for training accuracy and another for test accuracy. In both figures, plot the accuracy recorded as a function of the number of passes through the full dataset (number of epochs), with one curve for each candidate batch size. What trends do you notice?

Discussion of how performance varies with the numbers of batches:

c) *Repeat the previous step for a different step size schedule:* $\eta_t = (10 + t)^{-0.7}$. *Again, plot training accuracy and test accuracy versus number of epochs. Does performance appear sensitive to the step size schedule?*

Discussion of how performance varies with the step size schedule:

d) *Compare the performance of the $\hat{w}$ estimated by L-BFGS in part (a) to the estimates achieved by SGD. Discuss tradeoffs in solution quality versus the number of passes through the full dataset. (Due to overhead in the way objective function handles are evaluated, raw computation time may not be the best metric for this Python code.)*