CSE 515T (Fall 2021) Project

A portion of your grade this semester will be based on a significant project implementing and investigating Bayesian optimization algorithms. You will have two possible paths to satisfy this project requirement.

The main goal of the project is to give you hands-on experience applying Bayesian optimization to real-world problems. The use of real-world data can have many interesting (and potentially frustrating!) aspects that are difficult to convey without getting your hands dirty. The scope of the project is intended to be more than a homework problem, but less than a full-fledged research paper.

Note that I don't necessarily expect everythinga to "work." Part of research is trying out ideas, regardless of whether they're ultimately successful. If things do "work," I expect you to think about *why* it was succesful. If not, I expect you to think about *why not!*

This project will be undertaken alone, although I encourage you to talk about your ideas in class, on the Slack, and/or in person with your classmates!

Option 1: Novel project with existing data

Some of you may be working on existing projects or be an expert in relevant data where the techniques of this class could be brought to bear; for example if you are a Ph.D. student wishing to analyze existing data, or if you have a hobbyist interest in and passion for some particular dataset.

If this is the case you may propose a project applying Bayesian optimization to your problem. There is a great deal of freedom in defining this project to suit your needs. However, a critical requirement to this option is that *you be an expert in the data you are proposing to work with.* In particular, the following are *insufficient:*

- Kaggle competitions, etc.
- reference datasets from machine learning, UCI datasets, etc.

If you wish to pursue this route, your project will comprise three parts:

- A one-page proposal outlining the problem you want to solve, its importance and motivation, your proposed solution, and the data will use for your investigation. This will be due **Friday, 5 November 2021.** We will review your proposal and give you feedback. At this stage your proposal might be *rejected* if we feel the problem is not appropriate.
- A two-page status report reporting on your current progress and detailing any issues/questions you may have, due **Friday, 19 November 2021.** We will provide feedback and guidance.
- A four-page final report detailing the final approach and the outcomes of the investigation, due **Tuesday, 14 December 2021.** (No extensions!)

Option 2: Supervised project

If you do not wish to pursue the first project option, you can instead complete a guided project incorporating many notions from this class following the outline below. There is still considerable room for creativity in this option.

The project will be an investigation into applying Bayesian optimization to automatic and efficient hyperparameter tuning of machine learning models.

We will use two sources of data for this investigation. The first is a classical synthetic benchmark function used in optimization, the *six-hump camel* function. See here for a description, formula, and some implementations:

```
https://www.sfu.ca/~ssurjano/camel6.html
```

This is a function of a two-dimensional input and can be useful for visualizing the behavior of the methods you will implement. The goal is to minimize the function over the specified box-bounded domain.

The second is real data from a hyperparameter tuning task on two different models: SVM and online LDA. These datasets were used in the paper "Practical Bayesian Optimization of Machine Learning Algorithms" by Snoek, et al. The paper is available at

```
http://tiny.cc/bopt
```

You should read it!

Given some data, the performance of each methods was evaluated on a three-dimensional grid of hyperparameter values, defining an objective function $f(\theta)$, where θ is a three-dimensional hyperparameter vector. Our goal is to maximize the performance of the model as a function of the hyperparameters. As the cost of training and evaluating the performance of these models is so great, we will use the precomputed grids instead. The data is available here:

```
https://github.com/mwhoffman/benchfunk/tree/master/benchfunk/functions/data
```

For each problem, the goal is to minimize the value in the fourth column given the values of the hyperparameters in the first three. You can ignore the final column.

Note that all of these are *minimization* problems.

The project will comprise a series of smaller components that you must complete. You will then compile your findings into a report. I expect every bulleted item to be addressed in the report. The report will be due **Tuesday, 14 December 2021.** (No extensions!)

If you have any question about what may or may not be appropriate for any of these parts, please feel free to ask on Slack!

Data visualization

First let's do some visualization of the data to get some insight into the functions we are dealing with.

- Make a heatmap of the value of the six-hump camel function over the domain $\mathcal{X} = [-3, 3] \times [-2, 2]$ using a dense grid of values, with 1000 values per dimension, forming a 1000×1000 image.
- Describe the behavior of the function. Does it appear *stationary?* (That is, does the behavior of the function appear to be relatively constant throughout the domain?)
- Can you find a transformation of the data that makes it more stationary?

- Make a kernel density estimate of the distribution of the values for the LDA and SVM benchmarks. Interpret the distributions.

- Again, can you find a transformation that makes the performance better behaved?

Model fitting

Before we can succeed with optimization, we must first determine whether we can build a useful model of the objective function. If we can't find an informative model that can make reasonable predictions, it would be foolish to try to use the model to make decisions.

We begin with the six-hump camel function where we can make useful diagnostic plots. For this first series of steps, do not apply any transformation to the outputs of the function; use its output directly.

- Select a set of 32 training points for the six-hump camel function in the domain $\mathcal{X} = [-3, 3] \times [-2, 2]$ using a *Sobol sequence*. This is a so-called low-discrepancy sequence that produces "quasirandom" points that fill the space relatively evenly; see the Wikipedia article for more information, and note that this should be built into most scientific software. Measure the function at these locations to form a dataset $\mathcal{D}$.

- Fit a Gaussian process model to the data using a constant mean and a Matérn $\nu = 5/2$ covariance. As the output of a deterministic computer program, there is no "noise" in our observations, so we should fix the standard deviation of the noise to a small value such as 0.001. This is possible in most GP software packages.

  Maximize the marginal likelihood of the data as a function of the hyperparameters: the constant mean value and the length scale and output scale of the covariance function.

- What values did you learn for the hyperparameters? Do they agree with your expectations given your visualization?

- Make a heatmap of the Gaussian process posterior mean as you did of the function. Compare the predicted values with the true values. Do you see systematic errors?

- Make a heatmap of the Gaussian process posterior standard deviation (not variance!) as you did of the function. Do the values make sense? Does the scale make sense? Does the standard deviation drop to near zero at your data points?

  Note that if these heatmaps do not agree with your intuition, it may indicate a bug or mistake somewhere!

- Make a kernel density estimate of the z-scores of the residuals between the posterior mean of trained Gaussian process and the true values. If the Gaussian process model is well calibrated this should be approximately standard normal. Is that the case?

- Repeat the above using a log transformation to the output of the six-hump camel function. Does the marginal likelihood improve? Does the model appear better calibrated?

Now let's try some different models. Given a GP mean function and covariance, we can derive a score for how well those choices fit a given dataset, such as the dataset of 32 observations $\mathcal{D}$ above. The most natrual score would be the model evidence after integrating over the hyperparameters, but unfortunately this is intractable for Gaussian process models. A widespread and convenient approximation is called the *Bayesian information criterion* (BIC). Look up the Wikipedia article and

read it! To compute the BIC we first find the values of the hyperparameters maximizing the (log) marginal likelihood:

$$\hat{\theta} = \arg\max_{\theta} \log p(\mathbf{y} \mid \mathbf{x}, \theta).$$

Now the BIC is a combination of two terms rewarding model fit and penalizing model complexity:

$$\text{BIC} = |\theta| \log |\mathcal{D}| - 2 \log p(\mathbf{y} \mid \mathbf{x}, \hat{\theta}),$$

where $|\theta|$ is the total number of hyperparameters (above, 3) and $|\mathcal{D}|$ is the number of observations (above, 32). Given a set of models, we prefer the one with the lowest BIC, and this score can actually be interpreted as an approximation to the (negative) log model evidence.

- Compute the BIC score for the data and model from the last part.
- Considering BIC as a function of the choice of mean and covariance functions (μ, K), as well as the dataset described above, attempt a search over models to find the best possible explanation of the data. What is the best model you found and its BIC score? The notion of the "compositional kernel grammar" may be useful for automating the search, or you could do it by hand. You may also consider transformations of the data to improve the BIC further.
- Perform a similar search for the SVM and LDA datasets, using 32 randomly sampled observations from each dataset. What is the best GP model you found for each?

Bayesian optimization

Let's fix a choice of Gaussian process model for each of the datasets above by selecting the best model (possibly coupled with a transformation) you found, and for now let's fix the hyperparameters of that model to be those that maximize the marginal likelihood of the dataset. We will now implement a basic Bayesian optimization procedure and observe its performance.

- Implement the *expected improvement* acquisition function (see chapter 8), which provides a score to each possible observation location in light of the observed data. The formula is a function of the posterior predictive mean and pointwise variance of a point, which should be readily available.

  Be careful as we are doing *minimization* here. One easy solution is to negate and maximize.
- For the six-hump camel function, make new heatmaps for the posterior mean and standard deviation from the 32 datapoints we used before using the model you selected. Make another heatmap for the EI value, and place a mark where it is maximized. Does the identified point seem like a good next observation location?
- For the six-hump camel, SVM, and LDA functions, implement the following experiment:
  - select 5 randomly located initial observations, $\mathcal{D}$
  - repeat the following 30 times:
    * Find the point x that maximizes the expected improvement acquisition function given the current data. For the six-hump camel function, you can simply measure EI on a dense (I'm talking maybe one million points here) grid or on a dense Sobol set and find the maximum on that dense set. For the other datasets, you can measure EI on the unlabeled points and maximize.
    * Add the observation $\big(x, f(x)\big)$ to your dataset.

  - return the final dataset (containing 35 observations)

- We will evaluate performance as follows. Using the final returned dataset, identify the best point found. We will score optimization performance using the "gap" measure, which for maximization is:
$$\text{gap} = \frac{f(\text{best found}) - f(\text{best initial})}{f(\text{maximum}) - f(\text{best initial})}.$$
You can interpret gap is the portion of the "gap" between the best initial observation among the 5 initial data points and the optimum that we managed to traverse over the course of optimization; this gives us a normalized score between 0 and 1 for how well we did.

- Now perform 20 runs of the above Bayesian optimization experiment using different random initializations, and store the entire sequence of observations for each run.

  For a baseline, we will use random search. Using the same initializations, implement random search (i.e., a policy that samples an unlabeled point in the domain completely at random). Allow random search to have a total budget of 150 observations rather than 30 (5 times the budget), but store the entire sequence of data. This will allow us to compare with naïve "parallel random search" below.

- Make a plot of "learning curves" for each of the methods on each of the datasets. For now only using the first 30 observations for random search, plot the average gap achieved by Bayesian optimization and random search as a function of the number of observations for each of the datasets. (That is, after 1, 2, ... 30 observations, we visualize the performance of each method.) You'll want to make one plot per dataset with a curve for each method, allowing us to see how well each method compares on that dataset.

- What is the mean gap for EI and for random search using 30 observations? What about 60? 90? 120? 150? Perform a paired t-test comparing the performance of random search (using 30 observations) and EI. What is the p-value? How many observations does random search need before the p value raises above 0.05? We can interpret the result in terms of a "speedup" of Bayesian optimization over random search.

Bonus

Finally, for the remainder of the project, take this initial investigation into a new direction of your choosing. There is considerable flexibility here, and some examples are below. I obviously don't expect you to do all of these or even any of these if you have a different idea:

- Investigate different objectives or a different task.

- Implement more acquisition functions and compare their performance with EI and random search as above. There are numerous options out there!

- Investigate how the performance of Bayesian optimization changes if after every observation we relearn the model's hyperparameters by maximizing the marginal likelihood.

- Investigate batch Bayesian optimization. What if we select multiple points at once? There are numerous acquisition functions for implementing such a policy.

- Investigate how Bayesian optimization performs in the face of observation noise. You can simulate this noise by adding random noise with a given variance into each observation. Does noise slow us down? How much?

Please write a (at-least) one-page report on this final part, including your idea and your findings.