Hyperparameter Optimization with Bayesian Optimization

Many production and research systems that process large amounts of data iterate over some very large datasets and a single run can take up to a few hours. These processes are often controlled by a set of parameters that will dictate the effectiveness of the overall system for the needed task at hand. Some of the most data-intensive processes are model training and maximization processes that take training data and fit a model to it. A matrix of parameters is often used to store the trained model and these parameters are known as model parameters and are the goal of the training process. Often model training occurs for many iterations until the change of performance between model iterations is less than some small epsilon.

But before training occurs another set of parameters known as hyperparameters control the overall process through the many iterations. Examples of such hyperparameters include the learning rate in machine learning and deep learning processes, the number of topics in PSLA, and the k1, b, and k3 parameters in the Okapi BM25 ranking function. As seen in MP2.2, that despite the improvement over the years on various ranking functions, selecting the correct hyperparameters for a given data set and goal is very important for properly evaluating the performance of a model for a task. In MP2.2 two methods of tuning hyperparameters were most likely used by various students, random and grid search using many runs to determine which hyperparameter set is best. A grid search involves evenly dividing the hyperparameter vector domain evenly and running the algorithm each time and represents an exponential runtime. Given that training a model with a specific set of hyperparameters is time-consuming already, there have been techniques developed to search in a smarter manner.

To better frame the task at hand we can imagine these systems as black boxes that take a set of hyperparameters as input and outputs a metric to indicate the performance of the trained model. The goal is now to run this black box with sets of hyperparameters to maximize the output metric. The details of the model itself and how it runs are abstracted away from the task and the performance of the hyperparameter optimization schemes can be measured by the number of black-box runs it takes before reaching optimized hyperparameters. Random search and grid search explore the hyperparameter domain space with independent runs. This review will go over Bayesian optimization, a technique that uses the collection of the black box runs to create a statistical model of the domain space to predict the maxima.

Bayesian optimization is a strategy for black-box optimization that assumes that the function the black-box represents is a random variable and any known sample points of the function are called the prior. The prior is used to form a posterior distribution, typically using a method called Gaussian processes and Kriging. The posterior distribution can then be used to construct an acquisition function. The acquisition function takes into account the evaluated prior points in the gaussian process prior to suggesting the next best point to sample by predicting the areas with the most amount of expected improvement by sampling. The acquisition over time will trend towards the global maximum of the hyperparameter space.

There are some limitations to Bayesian optimization. The process of calculating a posterior distribution and acquisition function is costly with more black-box runs and the general consensus is to use it for systems with less than 20 hyperparameters with bounded ranges to keep the bayesian optimization from using more time than the black box runs. The technique is best used when the black box runs themselves are computationally time-intensive or resource-intensive such as when models are trained in the cloud, incurring a monetary cost. The black-box function should be continuous in nature and not contain discontinuities. If the black-box function has any special patterns like linearity with respect to its hyperparameters, it would be better to use an optimization scheme that uses those patterns instead of Bayesian optimization.

With respect to text information systems, Bayesian hyperparameter optimization is well suited. Since text retrieval systems rely on a collection of documents and the evaluation of a system is done using a collection of queries to determine the average precision, the evaluation of each set of hyperparameters used can take quite some time. The metrics calculated for text models such as average precision and likelihood shouldn't have huge jumps that would cause problems with continuity if used as the black-box output. These text models have complexity and are highly dependent on the types of documents being used such that there is no certainty to know if the text model's output metric has any features such as linearity to exploit instead of Bayesian optimization.

Although the math behind Bayesian optimization takes time to implement as code, there are many existing libraries that have fully implemented Bayesian optimizations. GPyOpt and bayeso are python implementations that can be installed using the standard python pip package manager. Bayesopt is a C++, C and python library implementing Bayesian optimization. GPflowOpt is a python implementation that leverages TensorFlow to improve the speed of the calculations and can be beneficial to systems already using TensorFlow. MOE is an implementation written by yelp and has libraries supporting C++ and python and is notable for being best deployed using docker to a virtualized environment and interfaced using REST commands. Aside from the surface interfaces, these implementations all offer support for Bayesian optimization with some offering support for more specialized systems.

The overall setup to get started with Bayesian optimization is as follows:

1. Define the hyperparameters of your model and the upper and lower bounds for them. Generally, integer hyperparameters can be optimized to a certain extent but must be rounded as required.
2. Set up the target model to be optimized for by wrapping it in a method that accepts a set of hyperparameters to use, trains the model, and outputs a single metric measuring its performance with those hyperparameters most likely with test queries or data. This is the black-box function to optimize.
3. Initialize the Bayesian optimizer to take the number of hyperparameters and the upper and lower bounds for them.
4. Initialize an empty history of black-box runs and their evaluated output
5. (Optional) Run the black-box function for some small number of sets of hyperparameters within the domain. These initial hyperparameter sets can help inform the Gaussian process posterior acquisition function and if not provided the

process will often start with the center and then the corners of the domain for the first few runs.
6. Pass in any prior evaluated hyperparameter sets and outputs to the Bayesian optimizer. The Bayesian optimizer should return the next point to evaluate
7. Evaluate the next set of hyperparameters with the black-box function and append the point and the output to the history. Keep track of the best output and the hyperparameter values used.
8. Repeat steps 6-7 for some N number of runs.
9. Return the values of the hyperparameters that yielded the best output.

References:
1. https://arxiv.org/pdf/1807.02811.pdf
2. https://sheffieldml.github.io/GPyOpt/firststeps/index.html
3. http://rmcantin.github.io/bayesopt/html/index.html
4. https://github.com/GPflow/GPflowOpt
5. https://bayeso.org/
6. https://github.com/yelp/MOE