Automated Model Selection via Gaussian Processes

Rui Shu RSHU15@STANFORD.EDU
Stephen Koo SCKOO@CS.STANFORD.EDU
AA 228, Stanford University

Abstract

In this paper, we tackle model selection by formulating the problem of hyperparameter optimization in the multi-model setting as a two-tiered system. The first tier is a multi-armed Gaussian Bandit problem that selects the model. The second tier is a Gaussian process-based Bayesian optimization technique that selects the optimal hyperparameters. We compare our model selection system to random search and demonstrated superior results. Our system extends on previous work in model selection, expanding model selection to include not only hyperparameter selection but also model family selection. We describe our system in detail and discuss future work for developing a more intelligent automated model selection system.

1. Introduction

The desire to learn the a model in a fully automated way has been of considerable interest in the machine learning community (Kanter & Veeramachaneni, 2015). Much attention has been places on the automation of feature engineering; there has been a recent surge in the popularity of deep neural network architecture because of a DNN's ability to avoid feature engineering. However, these models often require large amounts of data to be trained properly, are computationally costly, are difficult to interpret and intuit, and require careful tuning of hyperparameters in order for the DNN to train successfully.

Additionally, it is often a rule of thumb to start with a simple model before moving to more complex models on a case-by-case basis. But given the wide array of models that one can choose from, it is worth wondering whether it is possible to fully automate the predictive modeling process, providing automated selection of not only the optimal hyperparameters but also the optimal model for a given modeling task in the least amount of computational time.

2. Hyperparameter Selection

Many machine learning models requires the selection of hyperparameters *a priori*. These hyperparameters are often critical to the performance of the model (Snoek et al., 2015) and reflect assumptions the model makes about the data distribution prior to training the model on the existing data. The most common approach to hyerparameter tuning involves cross-validation: using a given set of hyperparameters, a model is trained on training data and evaluated on a test set (**?**). The optimal set of hyperparameters is thus the hyerparameters that maximizes the model's performance on the test set. It is thus possible to consider the existence of a latent function f that maps a given set of hyperparameters $\mathbf{x} \in \mathbb{R}^k$ to a performance metric. For a classification problem, this metric may, for example, be the accuracy of the model,

$$f(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1} \left\{ y_i = \hat{y}_i \mid \mathbf{x} \right\}, \tag{1}$$

where y_i corresponds to the true label of the i^{th} sample in the test set and $\hat{y}_i$ corresponds to label predicted by the trained model given hyerparameters $\mathbf{x}$. It is often the case that $f(\mathbf{x})$ is observed with noise (resulting from any randomness associated with the training of the model), thus making it appropriate to reason about the expected value of f instead.

2.1. Gaussian Processes

Since the goal to to find $\arg\max_{\mathbf{x}} \mathbb{E}\left[f(\mathbf{x})\right]$, the task of hyperparameter selection has often been formulated as a function optimization procedure performed using Gaussian process regression (Snoek et al., 2012). Gaussian process regression is a non-parametric Bayesian modeling tool that imposes a prior distribution over f and updates its belief about the distribution upon the observation of data. Formally, a Gaussian process is parameterized by a mean function $m(\mathbf{x})$ and a kernel $k(\mathbf{x}, \mathbf{x}')$, which we define as,

$$\begin{aligned} m(\mathbf{x}) &= \mathbb{E}\left[f(\mathbf{x})\right] & (2) \\ k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}\left[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))\right], & (3) \end{aligned}$$

which can be rewritten as $f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$. The exact choice of the functions m and k are priors required by the Gaussian process and the proper selection of such priors is describes in the next section. Crucially, a Gaussian process simply describes a collection of random variables $\{f(\mathbf{x}_i)\}_{1:n}$ drawn from a multivariate Gaussian distribution (Rasmussen & Wiliams, 2006), and can thus be rewritten as,

$$\begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m_1 \\ \vdots \\ m_n \end{bmatrix}, \begin{bmatrix} k_{11} & \dots & k_{1n} \\ \vdots & \ddots & \vdots \\ k_{n1} & \dots & k_{nn} \end{bmatrix} \right), \tag{4}$$

where the dependency on $\mathbf{x}_{1:n}$ ($f_i = f(\mathbf{x}_i)$, $m_i = m(\mathbf{x}_i)$ and $k_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$) is suppressed for notational simplicity. For further simplicity, we write,

$$f_{1:n} \sim \mathcal{N}\left(m_{1:n}, k_{(1:n)\times(1:n)}\right). \tag{5}$$

When seen as a multivariable Gaussian distributtion, it is easy to see how updating the Gaussian process involves computing a conditional Gaussian distribution. Supposing that the first $n-1$ points $f_{1:n-1}$ were observed, we compute the conditional distribution as,

$$\begin{bmatrix} f_{1:n-1} \\ f_n \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m_{1:n-1} \\ m_n \end{bmatrix}, \begin{bmatrix} k_\alpha & k_\beta \\ k_\beta^\top & k_\gamma \end{bmatrix} \right), \tag{6}$$

$$f_n \mid f_{1:n-1} \sim \mathcal{N}(m_{\mathcal{D}}(x_n), k_{\mathcal{D}}(x_n, x_n)), \tag{7}$$

$$m_{\mathcal{D}} = m_n + k_\beta^\top k_\alpha^{-1}(f_{1:n-1} - m_{1:n-1}), \tag{8}$$

$$k_{\mathcal{D}} = k_\gamma - k_\beta^\top k_\alpha^{-1} k_\beta, \tag{9}$$

where $k_\alpha = k_{(1:n-1)\times(1:n-1)}$, $k_\beta = k_{(1:n-1)\times n}$, and where $k_\gamma = k_{nn}$ (Rasmussen, 2006). Here, $m_{\mathcal{D}}$ and $k_{\mathcal{D}}$ denote the posterior mean and covariance functions upon observation of the data $\mathcal{D} = f_{1:n-1}$. Since it is often the case that realizations of the random variables $f_{1:n-1}$ are observed with some σ-Gaussian noise, it is common to use $k_\alpha = k_{(1:n-1)\times(1:n-1)} + \sigma^2 I_{n-1}$. Crucially, performing Gaussian process regression returns not only a posterior mean function, but also a posterior covariance function that properly captures the uncertainty of the interpolation. It is this ability of a Bayesian modeling tool to reason about the uncertainty of the system makes Gaussian processes for the online reinforcement learning task of function optimization (Wang et al., 2005).

2.1.1. Selection of Mean

Despite being non-parameteric, a Gaussian process is not completely free-form and does require the imposition of certain priors, namely the mean and kernel functions. Since it is often the case that the regression task is performed within the unit hypercube, a zero-mean Gaussian process where $m(\mathbf{x}) = 0$ will often suffice and its impact on the posterior mean diminishes quickly with the introduction of more data. It is sometimes useful, however, to impose a quadractive prior if expert knowledge suggests that the optimal hyperparameters is unlikely to appear on the boundary constraints of the optimization task (Snoek et al., 2015).

2.1.2. Selection of Kernel

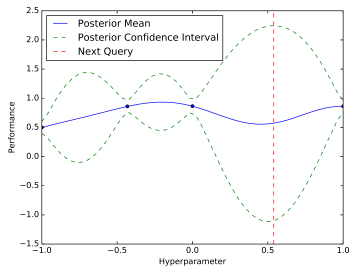
Perhaps of greater importance to the behavior of the Gaussian process is the kernel function. The choice of kernel functions can greatly impact the posterior mean and covariance of the Gaussian process. A plethora of kernel functions exist, ranging from periodic kernels to the Matérn and squared exponential kernel. Extensive research has been done on the performance of composite kernels generated from the linear combination of a set of basis kernels (Duvenaud, 2014). Within the context of test-set performance function, however, if one is reasonably confident that the performance function varies smoothly over the hyperparameter space, it is often enough to impose the squared exponential kernel,

$$k(\mathbf{x}, \mathbf{x}') = \theta_0^2 \exp\left(-\frac{1}{2}\gamma(\mathbf{x}, \mathbf{x}')\right), \tag{10}$$

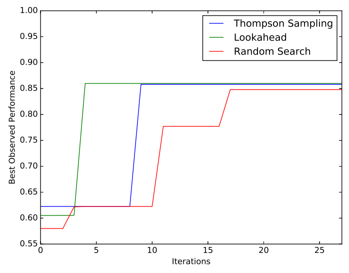
where $\gamma(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^k \frac{(x_i - x_i')^2}{\theta_i^2}$. It is still necessary to learn the parameters $\theta_{0:k}$. Of especial importance are the length scale parameters $\theta_{1:k}$, which reflect the extend to which observed points can influence the interpolation in its neighborhood. A large length scale reflects a greater sphere of influence, and vice versa. To learn these parameters, we choose the parameters $\theta_{0:k}$ that maximize the log marginal likelihood,

$$\begin{aligned} &\log p(f_{1:n} | \mathbf{x}_{1:n}, \theta_{0:k}) \propto \\ &-(f_{1:n} - m_{1:n})^\top k_\alpha^{-1}(f_{1:n} - m_{1:n}) - \log\det k_\alpha, \end{aligned} \tag{11}$$

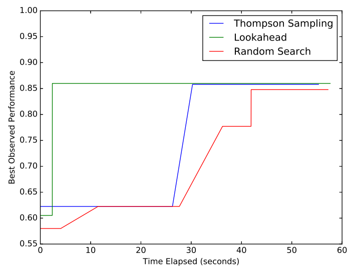
which can be performed via any gradient ascent algorithm (Rasmussen, 2006). The term involving $\log\det k_\alpha$ can be interpreted as a regularization parameter that encourages longer length scales (i.e. a low variance model) as long as such simplicity does not greatly hinder the model's ability to fit the data, which is captured by the quadratic form $(f - m)^\top k_\alpha^{-1}(f - m)$. Since k_α is also a function of the noise parameter σ, it is also possible to learn the value of σ via gradient ascent as well. However, introducing σ as a learnable paramater often produces many local optima that reflects a different interpretation of the data; a large σ is often accompanied by the learning of larger length scales, reflecting a low-variance high-bias interpolation, whereas a small σ is accompanied by small length scales, reflecting a high-variance low-bias interpolation (Rasmussen & Wiliams, 2006). Within the context of hyperparameter selection, the noise parameter is chosen beforehand and noted in the Experimental section.



(a) Gaussian process regression.



(b) Model search comparison by iteration.



(c) Model search comparison by time elapsed.

Figure 1. In 1(a), show an example Example Gaussian process fit of an unknown function, marked with the next point to query based on the computed maximum of the acquisition function. In 1(c) and **??**, we compare the model selection algorithms we present in this paper by plotting the trajectories of the best performance found with respect to the number of iterations (i.e. number of model-parameter combinations tried) and time elapsed. Note that **??** simply gives a different scaling of the same results, accounting for the differences between the training times required for different kinds of models.

2.2. Sequential Hyperparameter Selection

The use of a Gaussian process provides the Bayesian approach to modeling the uncertainty of the system, namely the confidence of our interpolated performance function. In an online function optimization framework, however, a sequential series of hyperparamters must be chosen, with the hope that each choice in the series allows us to make a more informed decision about the identity of the hyperparameters. To do so, it is important for the online system to properly manage exploration and exploitation. It is possible to formulate this task as a Belief-State Markov Decision Process (MDPs) where the belief-state is our belief about the latent performance function as capture by the Guassian process, the action-space is the set of all possible hyperparameters to choose from, and the observation-space is the set of all possible performance function evaluations at a particular hyperparameter. However, the continuous nature of the action and observation space makes most online methods intractable without discretization. It is thus common within the hyperparameter optimization literature to adopt the use of an acquisition function—a deterministic function of the posterior mean and covariance from the Gaussian process that allows for a simple albeit ad-hoc trade-off between exploration and exploitation (Snoek et al., 2012).

As is the case with kernel selection, many acquisition functions exist (Garnett, 2015). In our experiments, we use the upper-confidence bound (UCB) for hyperparameter selection, defined as,

$$m_D(\mathbf{x}) + 2\sqrt{k_D(\mathbf{x}, \mathbf{x})}. \tag{12}$$

The key benefit of such an acquisition function is that it is easily differentiable with respect to $\mathbf{x}$, enabling the use of gradient ascent to identify where the acquisition function is maximized without the need of evaluating the Gaussian process on a pre-defined grid of hyperparameters (Snoek et al., 2012).

3. Model Selection

While hyperparameter optimization has been extensively studied, there has been few research done on the hyperparameter optimization in a multi-model setting. Research in model selection has largely been focus on variable selection and hyperparameter tuning constrained within a single family of models, such as Random Forest (Kanter & Veeramachaneni, 2015). As the number of such approaches increases, it is natural to consider the possibility of extending the task of Bayesian optimization to include not only hyperparameter selection, but also model family selection. Here, we describe our formulation of the problem as an multi-armed Gaussian bandit.

3.1. Multi-Armed Gaussian Bandit

A multi-armed Gaussian bandit problem addresses the question of optimal decision making where one sequentially selects from one of N Gaussian distributions with unknown means and variances. In the bandit problem, the reward of each action is the observed realization from the chosen distribution. The multi-armed Gaussian bandit formulation of the model selection problem can be stated as follows.

Each sequential decision involves selecting both a model type (i.e. logistic regression, radius nearest-neighbors, etc.) and a set hyperparameters to initialize the model. The difficulty of performing model selection stems from the fact that each model type has an associated Gaussian process conditioned on the observations made thus far. However, note that we have predetermined our hyperparameter selection policy to be that of maximizing the UCB acquisition function. By doing so, we reduce our decision of which model down to that of selecting one of N models at particular hyperparameter initializations. By looking at each model at a particular hyperparameter, we reduce the Gaussian process to a single Gaussian distribution for each model, thus avoiding the need to make a decision based on N Gaussian processes.

3.2. Sequential Model Selection

As with sequential hyperparameter selection, there are many methods with which sequential model selection can be performed. The reduction of the problem to that of a multi-armed Gaussian bandit further opens up room for the use of various online methods available for Partially Observable Markov Decision Processes (POMDPs) (Kochenderfer, 2015). Of particular interest to us is to build a simple system that seeks a dynamic trade-off between exploration and exploitation dependent on the amount of time a user is willing to dedicate to automated model selection. Intuitively, the optimal policy ought to take into account a time limit imposed on the automated selection process; as the time limit gets smaller, the selection algorithm should become increasingly more conservative about querying computationally costly models. To reflect this, we define our reward function as follows,

$$R(a_t) = \begin{cases} o_t^{(f)} & o_t^{(t)} > 0 \\ 0 & o_t^{(t)} \leq 0 \end{cases}, \tag{13}$$

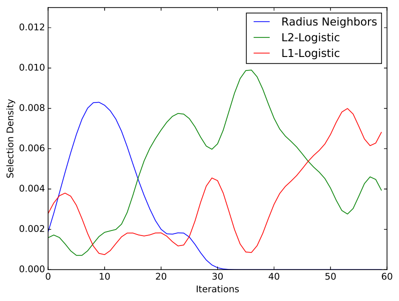
where a_t is the model chosen at time t, yielding a 2-tuple observation $o_t = (o_t^{(f)}, o_t^{(t)})$ where $o_t^{(f)}$ is the observed performance of the model at the chosen hyperparameters $\mathbf{x}$, and $o_t^{(t)}$ is the amount of time left on the clock after the t steps.

3.2.1. Approximate Value Function Lookahead

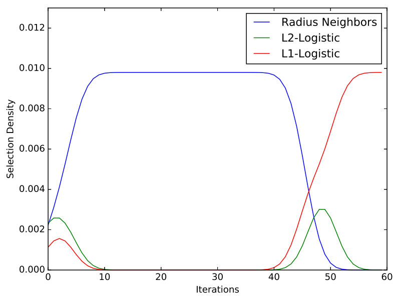
In the multi-armed Gaussian Bandit problem, the continuous obseravtion space still presents a challenge for optimal policy construction (Wang et al., 2005). One simple approach for tackling such a POMDP is to a one-step lookahead policy. Given a belief about the N Gaussian distributions, we choose the action that maximizes the following expression,

$$R(b_t, a_t) + \gamma \frac{1}{n} \sum_{i=1}^{n} U(b_{t+1}^{(i)}), \tag{14}$$

where $b_{t+1}^{(i)}$ is the updated belief after simulating a virtual observation o_t from the current belief-action pair. U is the approximate value function which is approximated via a rollout evluation strategy described in (Kochenderfer, 2015).



(a) Thompson Sampling model selection.



(b) Lookahead model selection.

Figure 2. In these figures we chart the frequency with which each model is tried by the respective algorithms. For visualization purposes, we use Gaussian kernel density estimation to estimate the density of these selection events with respect to the cumulative number of tries.

3.2.2. Thompson Sampling

Given current belief b_t about the multi-armed Gaussian bandit, Thompson sampling samples a observation of each of the N Gaussian distribution based on our current belief and simply performs the action that maximizes the reward. This simple approach is effective despite its simplicity and has often outperforms competing techniques. However, it is known that Thompson sampling is still a myopic selection strategy and is prone to overexploration (Wang et al., 2005).

4. Experiments

4.1. Experimental Details

To evaluate the performance of our online multi-model selection algorithm, we evaluated three models, L1-logistic regression L2-logistic regression, and Radius Neighbors classifier on a synthetic binary classification dataset containing 10000 samples. The samples were generated from a mixture of four Gaussian distributions and labeled according to the XOR labeling scheme. As such, the data set is not linearly separable. Each model contained a single hyperparameter, namely the penalty hyperparameter for the logistic regressions and the radius for the Radius Neighbors classifier.

We ran three different three different model selection algorithms. We used pure random search as our baseline. Despite its simplicity, random search is often used for hyperparameter selection and serves as a minimal baseline for our multi-model selection task (**?**). The random search algorithm randomly selects a model and a single set of hyperparameter at each iteration. We then used two flavors of our proposed multi-model selection algorithm. The first implements sequential model selection using lookahead with approximate value function, while the second uses Thompson sampling. Both versions iteratively selects a single model type (according to their respective methods) as well as a single hyperparameter using the UCB acquisition function.

All three selection strategies were tested with a fixed time limit of 60 seconds for model training and testing. The imposition of the time limit for this example problem simulates the situation in reality when model selection and hyperparameter selection are costly operations that make take many hours to run. Under these circumstances, the balance of exploitation and exploration ought to change depending on the amount of time allowed for the optimization task as well as the differential computation cost of each family of models.

4.2. Results

Our results demonstrate that both model selection approaches (Thompson sampling and Approximate Value Function Lookahead) are superior to random search. As can be see in Figures 1(c) and **??**, our selection techniques found the optimal model and hyperparameters in a significantly smaller number of queries than random search, thus reducing the number of expensive model performance computation queries. The ability of our selection system to quickly identify the optimal hyperparameter in a multi-model setting will be of significantly greater value when scaled to tackle truly complex models with costly cross-validation performance evaluations.

It is also interesting to compare the differences in the behavior of the model selection system when using Thompson sampling as compared to Approximate Value Function Lookahead. Both consistently exploit the best-performing model at the beginning. Since the data set is not linearly separable, the non-parametric Radius Neighbors classifier significantly outperforms logistic regression when its hyperparameter is carefully chosen. However, because Radius Neighbors classifier is computationally more costly than logistic regression (the Radius Neighbors has a run-time complexity of $O(n^2)$ while logistic regression has a run-time complexity of $O(p)$ where n is the number of samples and p is the number of predictors in the dataset), both model selection systems favor the simpler models as we approach the time limit. However, the two systems exhibit very different behaviors in the middle. Figures **??** and **??** show that Thompson Sampling is more likely to explore all three model types than Approximate Value Function Lookahead. This difference in behavior is consistent with Thompson sampling's known tendency to over-explore.

5. Discussion

We demonstrated two simple but successful ways of extending automated machine learning to that of the multi-model level by leveraging Bayesian-theoretic approaches toward reinforcement learning. The two-tier approach we propose for model and hyperparameter selection is highly flexible and can be readily modified to meet the needs of the user. Here, we propose future work that will improve upon our proposed system.

5.1. Bayesian Selection Techniques

We demonstrated the use of the UCB acquisition function, Thompson sampling, and Approximate Value Function Lookahead for the purposes of online hyperparameter and model selection. We directly compared Thompson sampling with Approximate Value Function Lookahead and demonstrated Thompson sampling's tendency to overexplore. While Approximate Value Function Lookahead reduces the risk of overexploration, it is nevertheless a crude technique for approximating the optimal value function associated with a given belief state. Other techniques, such as sparse sampling and Monte Carlo Tree Search has been proposed to deal with POMDPs with continuous observation space (**?**). It is conceivable that some of these more popular techniques for handling POMDPs will yield policies that better approximate the true optimal policy.

5.2. Reward Function Determination

Thus far, we implemented a fairly simplistic reward function. One of our goals for an automated model selection system, however, is for the system to adapt to the user's time and resource constraints, as well as any additional preferences the user may have of model complexity. For instance, if the user wishes to severely penalize the selection of a complex model, the reward function should be adjusted accordingly. One possibility is additively include a penalty term such as the Bayesian Information Criterion (BIC). The exact formulation of the reward function for automated model selection is worthy of exploration and an analysis of the model selection behavior under different types of reward function will allow us to tailor a reward function according to a user's needs.

5.3. Inter-Model Covariance

Our existing system uses a single Gaussian process for each model type. As it stands, observations from one model does not influence the Gaussian processes associated with other models. However, it is easy to intuit that the performance of one model on a given dataset can sometimes inform us about the expected performance of a closely-related model. For example, the performance of L1-logistic regression should inform us about the performance of L2-logistic regression. In other words, it is not surprising for the performances of two models to have non-zero covariances across datasets. Since a Gaussian process is formally simply a collection of random variables and is agnostic toward the the random variables coming from different sources (Rasmussen & Wiliams, 2006), it is not out of the question to incorporate out-of-family model performances into the Gaussian process. The main challenge of incoporating inter-model information stems from the choice of the covariance terms. Since the external model information comes from separate hyperparameter space, it is not possible to use a distance-based kernel function to approximate the inter-model covariance terms. However, these covariances can be learned via maximum likelihood estimation by training all models on a variety of datasets. With these additional research avenues, it is possible for our automated selection system to perform even better.

References

Bergstra, James and Bengio, Yoshua. Random search for hyper-parameter optimization. In *JMLR*. 2012.

Duvenaud, David. Automatic model construction with gaussian processes. University of Cambridge, 2014.

Garnett, Roman. Bayesian optimization. In *CSE 515T Lecture Notes*. 2015. URL `www.cse.wustl.edu/~garnett/cse515t/files/lecture_notes/12.pdf`.

Guez, Arthur, Silver, David, and Dayan, Peter. Efficient bayes-adaptive reinforcement learning using sample-based search. In *NIPS*. 2013.

Kanter, James and Veeramachaneni, Kalyan. Deep feature synthesis: Towards automating data science endeavors. In *IEEE*. 2015.

Kochenderfer, Mykel. State uncertainty. In *Decision Making under Uncertainty*, pp. 150. The MIT Press, 2015.

Rasmussen, C and Wiliams, C. Gaussian processes for machine learning. In *Gaussian Processes for Machine Learning*. MIT Press, 2006.

Rasmussen, Carl. Gaussian processes in machine learning. 2006. URL `http://www.cs.ubc.ca/~hutter/earg/papers05/rasmussen_gps_in_ml.pdf`.

Snoek, Jasper, Larochelle, Hugo, and Adams, Ryan P. Practical bayesian optimization of machine learning algorithms. In *NIPS*. 2012.

Snoek, Jasper, Rippel, Oren, Swersky, Kevin, Kiros, Ryan, Satish, Nadathur, Sundaram, Narayanan, Patwary, Mostofa, Prabhat, and Adams, Ryan. Scalable bayesian optimization using deep neural networks. In *JMLR*. 2015.

Wang, Tao, Lizotte, Daniel, Bowling, Michael, and Shuurmans, Dale. Bayesian sparse sampling for on-line reward optimization. In *ICML*. 2005.