Bayesian Optimization for Parameter Selection of Random Forests Based Text Classifier

Anonymous Author(s)
Affiliation
Address
email

Abstract

While random forest algorithm has been found to be prominent for various classification tasks, like many other machine learning algorithms it requires a number of parameters to be tuned to ensure better performance. Even though the strong influence of different parameters on random forest is evident, an attempt to systematically optimize these parameters is rare. Common techniques for parameter tuning such as cross validations are not often sufficient in this case, as the number of choices are increased. In this context, we propose a Bayesian optimization method to tune the parameters of random forest. We implemented a text classification system using the random forest package of Scikit-learn. To evaluate our approach, we compare the results on different parameter settings generated during optimization procedure. We also examine how various choices of acquisition functions could potentially affect the optimization. Our results suggest that by tuning the parameters for random forest, we could enhance the classification performance over default choices of parameters provided in Scikit-learn package.

1 Introduction

In recent years, due to its algorithmic simplicity and prominent classification performance for high dimensional data, random forest has become a promising method for different classification tasks such as text categorization. Random forest is an ensemble of a set of a single type of decision trees. The algorithm randomly selects a subset of features at each node to grow branches of a decision trees. Then, the voting mechanism operates on the top of base learners to ensure highly accurate predictions of the ensemble. This ensemble method helps to avoid overfitting, and is less sensitive to noisy data compared to other classification methods [1].

Even though the performance of random forest classifier is impressive, it has a number of crucial parameters that can significantly influence the behavior and performance that it offers. For instance, the size of the random forest, the maximum allowed tree depth, the number of features chosen at random, and the split criteria: all of them are reported to affect the performance of the classifier [2]. Despite such influential characteristics, very little attention is provided to carefully tune these parameters for classification tasks. While cross validations or some brute-force searches are often applied to adjust the hyperparameters, as the number of parameters becomes high they may not be viable options. This leads to great appeal for automatic approaches that can optimize the performance of random forest algorithm.

One of the good choices of automatic optimization of parameters is Bayesian optimization, which has been shown to outperform other state of the art global optimizations on a number of benchmark functions [3]. Bayesian optimization can be used on top of Gaussian process, by assuming that the unknown function was sampled from a Gaussian process and maintaining a posterior distribution for this function as observations are made. To pick the set of parameter values for the next experiment one can use different acquisition functions.

In this project, we are interested in applying Bayesian optimization on top of Gaussian process to tune the parameters of random forest. In particular, we would like to use the observations made from the results of running random forest algorithm experiments, and use them to pick a next values of the parameters. Our hypothesis is that by using such automatic parameter tuning, the performance of random forest can be improved for many different classification tasks. In addition, we are also interested to know whether any particular choice of acquisition function would lead to better performance than the others.

The primary task that we have chosen is text classification over a standard dataset. In text classification, we can literally have millions of dimensions, causing the different parameters of random forest to play more crucial role in affecting the performance of the classifier than many other types of classification. Thus, this task makes a suitable scenario for automatic tuning via Bayesian optimization.

Remainder of this report is organized as follows: in Section 2 we describe the random forest and its crucial parameters as well as Bayesian optimization method. The implementation of text classification and Bayesian optimization are provided in Section 3. We analyze our results empirically in Section 4. Finally, we discuss what lessons have been learned throughout the project and what are the possible future directions.

2 Background

2.1 Random Forests and its model parameters

The primary idea of random forest is to build a large collection of de-correlated trees, and then average the prediction over all of them [1]. Random forest improves the variance reduction of bagging by reducing the correlation between the trees, without increasing the variance too much. This is achieved in the tree-growing process through random selection of the input variables. To grow each tree, the algorithm draws bootstrap samples $Z*$ of size N from the training data and then recursively split nodes based on a random set of m different features drawn from p different features, until a minimum node size is reached. To classify a new object from its features, the algorithm pushes the input feature vector through each of the decision trees in the forest (starting at the root), until it reaches the corresponding leaves. Thus, each tree gives a class prediction, in other words it votes for that class. The forest chooses the classification having the most votes over all the trees in the forest. More elaborated description of the algorithm can be found at [1, 2]; here we focus our discussion on different model parameters of the algorithm and their influences, as they are the primary interests of this paper.

There are a number of influential model parameters of random forests. A nice description of the effect of some of these parameters can be found at [2]. Here we are summarizing some of the key parameters that we are interested to tune:

1) **Depth of the tree (D)**: The tree depth is a crucial parameter in avoiding under-fitting or over-fitting. By experimenting with varying tree depth D, the authors observe that as the tree depth increases, the overall prediction confidence also increases [2]. It has also been found that too shallow trees leads to under-fitting (class boundary become too course). On the contrary, a large value of tree depth tends to produce over-fitting, i.e., posterior tends to split off isolated clusters of noisy training data. In essence, the maximum tree depth parameter $\boldsymbol{D}$ controls the amount of over-fitting. Therefore, one needs to be very careful to select the most appropriate value of $\boldsymbol{D}$ as its optimal value.

2) **Number of samples for Bagging:** In bagging, randomness is injected by randomly sampling different subsets of training data. So, each tree sees a different training subset. The choice of how many samples should be in each subset controls the effect of randomness. If we avoid bagging and use all the training data, then we would

reproduce a max-margin behavior, while increasing randomness leads to smoother posteriors whose optimal boundary does not coincide with the maximum margin. In other words, bagging gives immunity to outliers. Overall, this behavior is controlled by how much randomness is injected through selecting the number of samples for bagging.

3) **Number of features for splitting node (m):** In random forests, only a subset of features (τ) of size m is used from the original set of features having size p to split the node. A smaller value of m enhances randomness making the trees very different from each other. The ratio of m/p controls the randomness.

4) **Forest size (T):** Previous research works have pointed out how the testing accuracy increases monotonically with the forest size T [2]. It has been found that single tree produces over-confidence, and ultimately leads to imperfect generalization. On the contrary, more trees give much smoother class posterior. While this would encourage us to use larger size of T, computation could take much longer time. In addition, note that results will stop getting significantly better beyond a critical number of trees. Hence, finding an optimum forest size that is big enough to produce smoother boundary, yet small enough for computation cost is essential.

Beside the abovementioned parameters, we also have a set of other choices that needs to be made, such as the split criteria (Information Gain (IG) verses Gini index), and the minimum number of samples to have in newly created leaves etc. In Section 3, we will describe how we apply Bayesian optimization to tune the abovementioned parameters.

2.2 Bayesian optimization with Gaussian Process

Bayesian optimization has been found to be increasingly popular in recent years [3]. It could be a very effective strategy for finding the extreme of objective functions that are expensive to evaluate. The technique is particularly useful when we do not have a closed-form expression for the objective functions, but we can make observations of the function at sampled values.

More formally, Bayesian optimization aims to find the minimum (or maximum) of a function $f(x)$, on some bounded set X. It constructs a probabilistic model for $f(x)$ and then exploits this model to make decisions about where in X we should sample next. To sample efficiently, Bayesian optimization uses acquisition function which essentially trade-offs between exploration and exploitation [4].

To perform Bayesian optimization one must select a prior over functions that will express assumptions about the functions being optimized. The Gaussian process (GP) serves as a convenient and powerful prior distribution of functions. A GP is an extension of the multivariate Gaussian distribution over functions, specified by its mean function m and covariant function, $K: f(s) \sim GP(m(x), k(x, x'))$. We assume that the function $f(x)$ is drawn from a Gaussian process prior and that our observations are of the form $\{x_n, y_n\}_{n=1}^{N}$, where $y_n \sim N(f(x_n, v))$ and v is the variance of the noise induced into the observations.

The abovementioned prior and data induce a posterior over functions called acquisition functions. Maximizing the acquisition function is used to find the next point to evaluate the function, i.e., we wish to sample f at $argmax_x u(\boldsymbol{x}|D)$, where $u(.)$ is the generic symbol for an acquisition function.

Probability of Improvement: One strategy to maximize the probability of improving over the current best $f(\boldsymbol{x}+)$:

$$PI(x) = P(f(x) \geq \mathrm{f}(x^*)) = \Phi(\frac{\mu(\mathrm{x}) - \mathrm{f}(x^+)}{\sigma(\mathrm{x})})$$

Where, Φ is the normal cumulative distribution function.

Expected Improvement: Alternatively, we can try to minimize the expected deviation from the true maximum $\mathrm{f}(x^*)$, when choosing a new point to sample. Mockus et al. proposed maximizing the expected improvement with respect to $\mathrm{f}(x^+)$ [5] as follows:

$$x = argmax_x\, \mathbb{E}(max\,\{0, f_{t+1}(x) - \boldsymbol{f}(\boldsymbol{x}^+)\}|\boldsymbol{D_t})$$

The expected improvement can be evaluated analytically:

$$EI(x) = \begin{cases} \left(\mu(x) - f(x^+)\right)\Phi(Z) + \sigma(x)\phi(Z) & \text{if } \sigma(x) > 0 \\ 0 & if\ \sigma(x) = 0 \end{cases}$$

$$Z = \frac{\mu(x) - f(x^+)}{\sigma(x)}$$

GP confidence bound criteria: Recently Srinivas *et al.* exploit confidence bound to construct acquisition functions that minimizes regret over the course of their optimization [6]. The acquisition function has the form:

$$\mathrm{GP-UCB}(x) = \mu(x) + \sqrt{v t_t \sigma(x)}$$

The acquisition functions described above have analytical expressions that are easier to evaluate. A further improved way could be to follow an approach described by Eric Brochu et al., where a portfolio of acquisition functions governed by an online multi-armed bandit strategy was used which was reported to outperform the best individual acquisition function [7].

3 Random Forest Parameter Selection Using Bayesian Optimization

In this section, we first describe the implementation of text classification algorithm using random forest, as well as the parameter space that we explored in the experiments. Then, we discuss the implementation of Bayesian optimization along with the selection of covariance functions and acquisition functions.

3.1 Text classification using random forest

3.1.1 Dataset

20Newsgroups [8] data set is a popular text corpus for experiments in text applications of machine learning techniques. It is a set of 18,828 Usenet messages from 20 different online discussion groups. The corpus is sorted by date and divided in advance into a training (60%) set and a chronologically following test set (40%) (This way we avoid randomness in train/test set selection).

3.1.2 Feature extraction

In order to perform classification on text documents, we first need to convert the text content into numerical feature vectors. A common way to do so is to utilize bags of words representation. We first tokenize the text and filter the stopwords. Then we build a dictionary from words and assign a fixed numeric index to each word occurring in any document of the training set. We count the number of occurrences of each word. We then compute Term Frequency times Inverse Document Frequency (*tf-idf*) and use it in the feature vector representation. Bag of words are typically high-dimensional sparse datasets, so we only store the non-zero parts of the feature vectors in memory.

3.1.3 Random forests classification

We use the Random forest algorithm using Scikit-learn: a machine learning toolkit in python [9]. This implementation is similar to the description provided in [10]. However, this implementation combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class. Particularly, this implementation provides us with a way to set the different parameters (such as tree depth, number of trees, number of features to be used for splitting nodes, criteria for splitting node (entropy vs. gini index) etc. This way we were able to run the algorithm with various set of parameters.

The four parameters that we experimented are listed in Table 1. Considering the computational cost (and fact that the performance of random forest becomes optimum after reaching certain size of forest), we keep the maximum forest size to be 100. Other parameters are set based on the dataset. One point to note that the Scikit-learn implementation does not provide the option to directly set the number of samples for bootstrapping, rather a Boolean parameter is provided which can be set to turn on or off bagging. Therefore, we did not use this parameter in this experiment and would like to explore this in the future.

Table 1: The set of parameters to be tuned for random forests

Parameters	**Range of values**	**Default value in Scikit**
Forest size (T)	Min:1, Max:100	10
Depth of the tree (D)	Min:10-Max:10000	None (nodes are expanded until all leaves are pure)
The minimum number of samples required to split an internal node	Min:5-Max: all samples	2
Number of features for finding best split node (m)	Min:2-Max:100	sqrt(number of features)

3.2 Bayesian Optimization

As stated before, the main objective of using Bayesian optimization here is to find the suitable value for each parameter of random forest algorithm. To do so, we followed an approach of Bayesian optimization described in [3]. There are at least three important practical choices that we need to consider: the covariance functions, selection of its hyperparameters and the acquisition functions. A default choice of covariance function is to use squared exponential kernel. However, similar to [3], we use automatic relevance determination (ARD) Matern 5/2 kernel.

$$K_{M52}(x,x') = \theta_0(1 + \sqrt{5r^2(x,x')} + \frac{5}{3}r^2(x,x'))\exp\{-\sqrt{5r^2(x,x')}\}$$

Then second question is that the above kernel function itself has few parameters that needs to be managed (such as covariance amplitude θ_0 and the observation noise v). As pointed out in [3], we could do it by marginalize over hyperparameters and compute the integrated acquisition function. To serve this purpose we can blend acquisition functions arising from samples from the posterior over GP hyperparameters and have a Markov Chain Monte Carlo (MCMC) estimate of the integrated expected improvement.

The final question is which acquisition functions to use. There are several different parameterized acquisition functions in the literature (some of them are mentioned in Section 2), and often it is difficult to decide which one is the most suitable given the optimization tasks. In this work, we evaluate the results based on multiple acquisition functions and compare between them.

4 Empirical Analyses

In this section, we empirically analyze the parameter optimization of random forest performed by Bayesian optimization. Our primary goals are two-folds. First, we would like to compare the optimization results based on different acquisition functions. Second, we want to examine whether Bayesian optimization leads to better classification performance of random forests, when comparing with the results produced by Scikit-learn's default parameter setting.

4.1 Experiments

We perform experiments using three different types of optimization strategies that were implemented in [3]: GP EI MCMC, GP EI OPT and random grid search. For each experiment, we run 40 iterations of the Bayesian optimization. At each iteration, a new set of parameters were generated by the acquisition functions, and the random forest algorithm was called based on these parameters.

As the classifier is trained and prediction tasks are performed, we compute various matrices such as average precision, recall and F-score. We choose the F1-score to be the best indicator of performance, since it takes both precision and recall into account. These results are then feed to the Bayesian optimization procedure. The objective of Bayesian optimization is then to find the parameters of random forests that maximize this F1 score.

4.2 Results

We collect the results obtained from different optimizations. While it is preferable to retrieve results on multiple runs and average them, due to time constraint we collect one set of results per optimization strategy (Each experiment needs several hours to complete). At each iteration, we evaluate the function value (F1 score) and keep track of the best value obtained so far. Figure 1 shows the performance of different optimization strategies. As we can see GP EI MCMC performs the best followed by GP EI. In both cases, within very few iterations, the maximum F1 score was achieved. Random Grid search produces better F1-score at the beginning but eventually other two methods found higher function values.

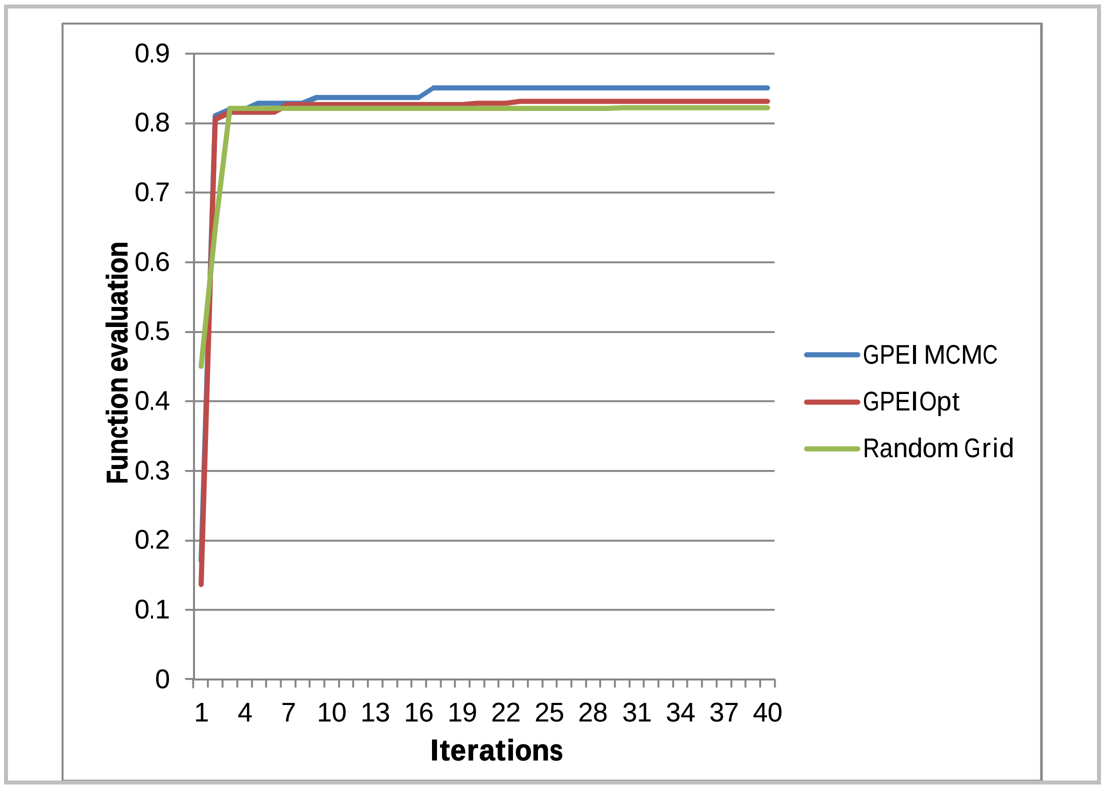

Figure 1: Bayesian optimization results for text classification. The graph plots iterations on X axis and F1 score obtained for that iteration on Y axis

We also run the experiments of Random forest classification using default parameter settings of Scikit-learn (as mentioned in Table 1). When we compare the results with the best value obtained using Bayesian optimization with Random forest having default setting, we notice significant improvement over F1 score (beating by over 4.1%). We regard this as encouraging results.

There are a number of limitations of the experiments reported here, that we would like to address in the future. Overall, our results are generated by a small set of experiments and therefore further experiments are required for each optimization strategy to conclude whether the results are significantly different. Also sufficient error analysis is required to perform, when comparing between different results. Finally, further experiments are required on large text dataset to examine how Bayesian optimization could potentially improve accuracy and recall in such scenarios.

5 Conclusion and Future Work

In this project, we explore the idea of using Bayesian optimization to tune the hyperparameters of random forests algorithm. Previously, only a little attention was provided to tune these parameters, and they were primarily tuned based on cross validations. Our results show that Bayesian optimization can be very effective to find the optimized parameter values that maximize classification performance. Moreover, we found that such optimal values were obtained within a few iterations, thus reducing the cost of evaluating functions, which often takes longer to compute for random forest algorithm. We believe that these results are encouraging enough for those who want to ensure the optimized performance of random forest algorithm for various classification tasks.

There are a number of avenues that we would like to explore in the future. First, we would like to explore other variants of Bayesian optimization such as portfolio of acquisition functions governed Bayesian online multi-armed bandit strategy, which outperforms individual acquisition functions [4], or applying binary trees partition on the input parameters [11] and compare the performance. Secondly, while we wanted to optimize random forest parameters through Bayesian optimization, this optimization method itself could have some choices as explained before, which also need to be optimized (such as choice of acquisition function, co-variance function, and along with the parameters). Finally, we would like to experiment on sufficiently large-scale dataset to see how having billions of features could possibly lead to different possible settings of parameters.

Acknowledgments

Special thanks to Jasper Snoek for providing some clarifications regarding his publicly available software (Spearmint) for Bayesian optimization. Also I would like to thank Bobak Shahriari for useful suggestions.

References

[1] Hastie T., Tibshirani R., and Friedman, J. (2009) The Elements of Statistical Learning: Data Mining, Inference and Prediction. Springer, second ed.

[2] Criminisi, A., Shotton, J., & Konukoglu, E. (2011) Decision Forests for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning, Technical report MSR-TR-2011-114.

[3] Snoek, J., Larochelle, H., and Adams, R. P. (2012) Practical Bayesian optimization of machine learning algorithms. *In Advances in Neural Information Processing Systems (NIPS).*

[4] Brochu, E., Cora, V. M., and de Freitas, N. (2009) A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. Technical Report UBC TR-2009-23. arXiv:1012.2599, Dept. of Computer Science, University of British Columbia.

[5] Mockus, J., Tiesis, V., and Zilinskas. A. (1978) The Application of Bayesian Methods for Seeking the Extremum, *Toward Global Optimization,* 2:117-128.

[6] Srinivas, N., Krause, A., Kakade, S. M., and Seeger, M. (2010) Gaussian process optimization in the bandit setting: No regret and experimental design. *In Proceedings of the International Conference on Machine Learning (ICML).*

[7] Homan, M., Brochu, E., and de Freitas, N. (2011) Portfolio allocation for Bayesian optimization. *In UAI*, 327-336.

[8] Lang. K. (1995) Newsweeder: Learning to filter net-news. *In Proceedings of the International*

Conference on Machine Learning (ICML), pages 331–339. Morgan Kaufmann.

[9] Pedregosa et al. (2011) Scikit-learn: Machine Learning in Python, *Journal of Machine Learning Research*, 12:2825-2830.

[10] Breiman, L. (2001). Random Forests. *Machine Learning* 45(1):5-32.

[11] Gramacy, R. B., Lee, H. K. H., & Macready, W. G. (2004) Parameter space exploration with Gaussian process trees. *In Proceedings of the International Conference on Machine Learning (ICML)*, pages 45-52.