Bayesian Optimization with Support Vector Regression

Background

Many machine learning algorithms require careful tuning for model hyperparameters, regularization terms, and optimization parameters. This tuning, however, is not a simple task that can be optimized easily; it is often called a "black art" that requires expert experience, unwritten rules of thumb, or sometimes brute-force search. This led people to solve for automatic tuning problem and develop different frameworks. These optimization problems contain functions that are very expensive since they involve running the primary machine learning algorithm to completion. In this case, people should spend computational time making better choices about where to seek the best parameters.

State-of-the-art Hyperparameter Tuning

In most cases of hyperparameter tuning, grid search methods are utilized. Grid Search methods can be categorized into two main branches: Manual Grid Search(MGS) and Random Grid Search(RGS). Manual Grid Search(MGS) builds a grid of all possible sets of hyperparameters. Then, we evaluate the performance for every possible combinations. However, we can easily see that MGS takes a very long computational time and it can easily fall into the Curse of Dimensionality. As a solution to these problems of MGS, Random Grid Search(RGS) method is also popular. RGS is similar to MGS in that it also builds a grid of all possible sets of hyperparameters. Then, we randomly choose points to evaluate. By utilizing RGS, we can parallelize each of our randomly chosen point to evaluate, but there is no guarantee of reaching a high-performance region. Therefore, we are in need of conditionalizing the RGS process, and Bayesian Optimization, which uses previously evaluated hyperparameter to guide the next search, can algorithmically steers the search to the region with better performance.

Bayesian Optimization

Bayesian Optimization is a learning algorithm that its generalization performance is modeled as a sample from a Gaussian process (GP). Since posterior distribution induced by the GP is tractable, it enables optimal choices about what parameters to try next with efficient use of information obtained from previous experiments. For continuous functions, by assuming the unknown function was sampled from a Gaussian process (GP), Bayesian optimization works and maintains a posterior distribution for this function as observations are made. Picking the hyperparameters for the next experiment, one can use the expected improvement (EI) for optimization. The different aspect of Bayesian optimization from other procedures is that, constructing a probabilistic model for f(x), it exploits this model to make decisions about where in X to next evaluate the function, while integrating out uncertainty. This basically use all of the information available from previous evaluations of f(x) and not simply rely on local gradient.

Algorithm

To perform Bayesian optimization, first, one must select a prior over functions that express assumptions about the function being optimized. Choosing GP prior makes it flexible and tractable. Second, one must choose an acquisition function to construct a utility function from the model posterior to determine the next point to evaluate.

Algorithm 1 Bayesian Optimization

1: **for** $t = 1, 2, \ldots$ **do**
2: Find $\mathbf{x}_t$ by optimizing the acquisition function over the GP: $\mathbf{x}_t = \text{argmax}_{\mathbf{x}}\, u(\mathbf{x}|\mathcal{D}_{1:t-1})$.
3: Sample the objective function: $y_t = f(\mathbf{x}_t) + \varepsilon_t$.
4: Augment the data $\mathcal{D}_{1:t} = \{\mathcal{D}_{1:t-1}, (\mathbf{x}_t, y_t)\}$ and update the GP.
5: **end for**

Figure 1.1 Bayesian Optimization Algorithm

Illustrated in Figure 1.1, the algorithm can be put as three steps:

1. Given observed values $f(x)$, update the posterior expectation of f using the GP model.
2. Find x_{new} that maximises the EI: $xnew = argmaxEI(x)$.
3. Compute the value of f for the point x_{new}.

This procedure is either repeated for a pre-specified number of iterations, or until convergence.

Datasets

Table 1.1 Dataset Description

	# of samples	# of features	Class labels	Distribution
Red-wine-quality	1599	11	0~10 (Integer)	Normal
Iris	150	4	Iris Setosa Iris Versicolour Iris Virginica	Uniform

Illustrated from Table 1.1, we used two datasets, red-wine-quality dataset being large-scale in terms of sample size and feature size, and iris dataset being relatively simpler than the other. Red-wine-quality dataset has 1599 samples with 11 features (fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol), and its class, which is quality of wine, is integer value ranging from 0 to 10, 10 being the best wine. It is normally distributed, meaning most samples belong to class 4,5,6. Iris dataset has 150 samples with 4 features (sepal length, sepal width, petal length, petal width), and its class consists of three kinds of Iris. We assigned integer value 1,2,3 for each kind of Iris, and used both datasets to solve a regression problem with minimizing mean squared error.

Support Vector Regression

Support Vector Machine (SVM) was developed to solve the classification problem, but recently is being used to solve the regression problems as well, maintaining all the main features that characterize the algorithm (maximal margin). Support Vector Regression (SVR) uses the same principles as the SVM for classification with a few minor differences. Since output is now a real number making very difficult to predict the information at hand. By introducing an alternative loss function that includes a distance measure, SVM can be applied to regression problems.

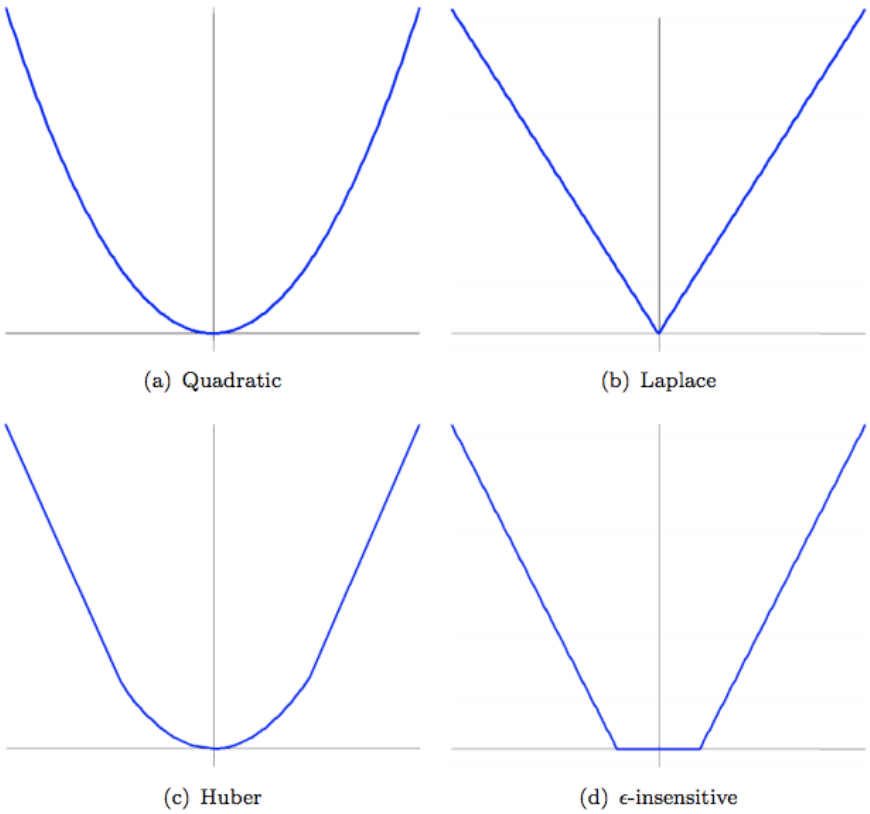

Figure 2.1 Loss Functions

Figure 2.1 represents different loss functions that can be used in SVM. SVR uses (d), $\epsilon - intensive$, that, unlike the quadratic and Huber loss, enables a sparse set of support vectors to be obtained. In the case of regression, one can set a margin of tolerance, epsilon. The main idea is, individualizing the hyperplane which maximizes the margin, tolerating some part of the error within epsilon-tube, to minimize error.

$$y = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) \cdot < \varphi(x_i), \varphi(x\ ) > + b \qquad (1.1)$$

$$y = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) \cdot K(x_i, x) + b \qquad (1.2)$$

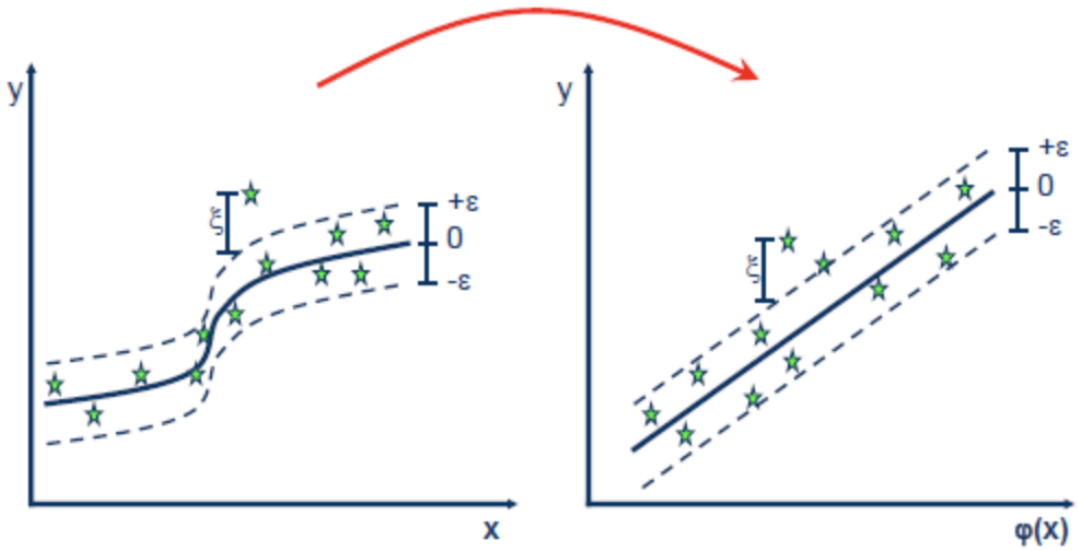

Figure 3.1 Transformation into High Dimensional Feature Space

Illustrated in Figure 3.1, for high dimensional feature datasets, a non-linear mapping can be used to map the data into a high dimensional feature space where linear regression is performed. The kernel approach is employed to address the curse of dimensionality.

Implementation and Result

For this project, we used scikit learn's SVR model and applied the bayesian optimization on this model. It has three parameters (C, γ, ϵ) that we can apply for bayesian optimization. C is a penalty parameter of the error term, $\gamma$is kernel coefficient for 'rbf', 'poly' and 'sigmoid', and ϵspecifies the epsilon-tube within which no penalty is associated in the training loss function with points predicted within a distance epsilon from the actual value. The default values for C, γ, and $\epsilon$are respectively 1.0, 1/number of features, and 0.1. For SVR, we used default kernel, 'rbf', for all experiments. For bound of hyperparameters (C, γ, ϵ), we set [-4, 1]. We sampled those parameters on the log-uniform scale. We used mean square error for our loss function, which needs to be minimized instead of maximized.

For our experiment, we used guessing vs. bayesian optimization. For guessing, we arbitrarily guessed hyperparameters as $C = 2.0, \gamma = 2.0, \epsilon = 1.0$. Then we compared MSE from SVR using those arbitrary hyperparameters to MSE from SVR using best hyperparameters obtained from bayesian optimization algorithm. For bayesian optimization performance comparison, we used 10, 50 and 200

iterations to compare the performance with different number of iterations of bayesian optimization, and used two different GP kernels, rbf and matern with $\nu = 2.5$ since it can be applied for twice differentiable loss function.

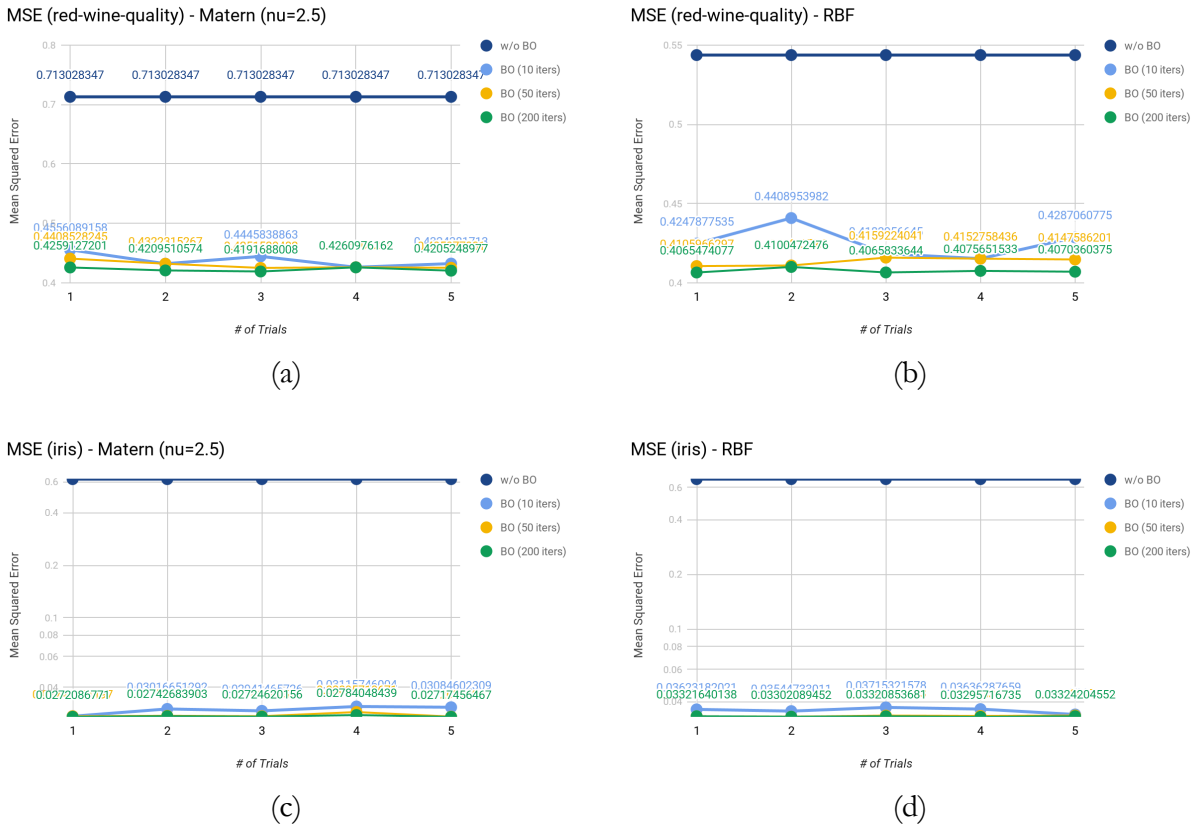

Figure 4.1 Mean Squared Error for 5 Trials with Different Iterations and Kernels

From Figure 4.1, for Red-Wine-Quality dataset, as the iteration gets larger, both the matern kernel, 4.1(a), and the rbf kernel, 4.1(b), gave lower MSE and perform better. Overall, for both cases of Matern and RBF kernels, the evaluation with optimized Bayesian-optimized hyperparameters gave much better performance than original unoptimized arbitrary hyperparameters. For Iris dataset, which is much smaller scale, from both figures 4.1(c) and 4.1(d) shows the effectiveness of Bayesian Optimization. For smaller dataset, number of iteration did not affect the performance as much as it did in a bigger data set, and they all produced similar performance results.

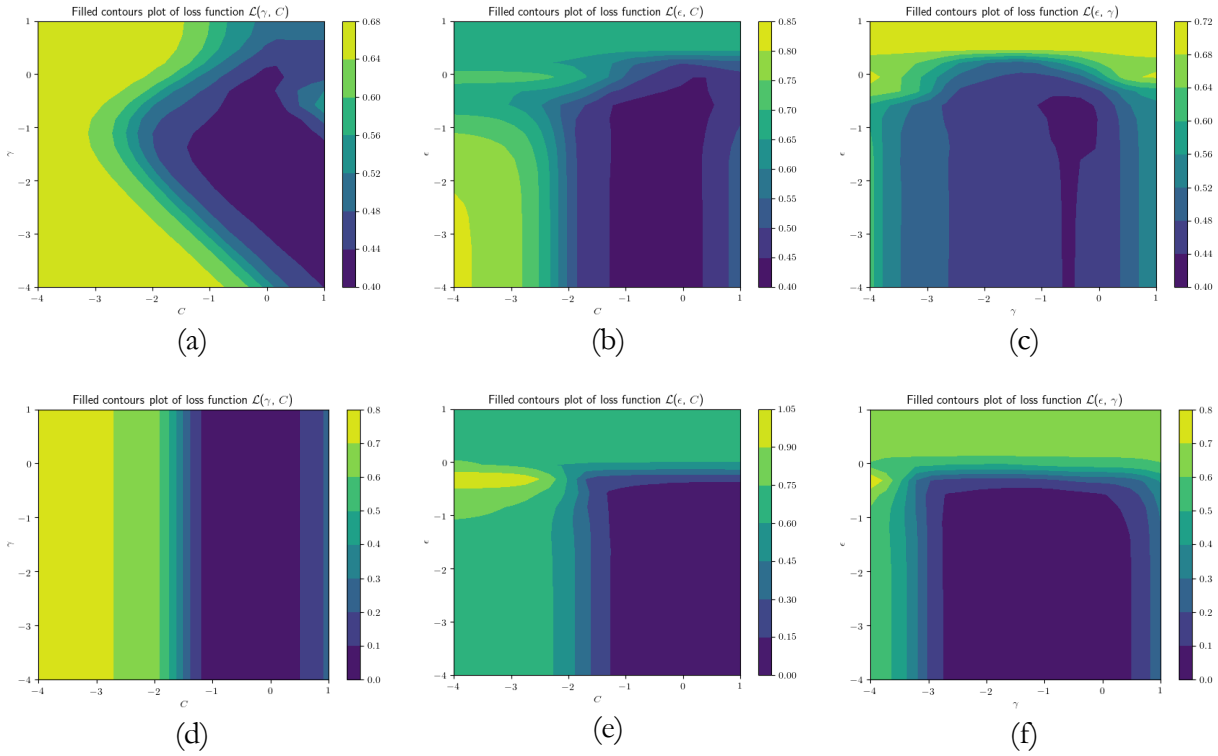

Figure 5.1 Loss Surface as a Function of C, γ, ϵ

As visualized in Figure 5.1, We can actually compute the loss surface as a function of C, γ and ϵ. This way, we can get an accurate estimate of where the true optimum of the loss surface is. For 5.1(a), 5.1(b) and 5.1(c), they are loss surfaces for red-wine-quality dataset as a function of (C ,γ), (C , ϵ) and (γ, ϵ) respectively. For 5.1(d), 5.1(e) and 5.1(f), they are loss surfaces for iris dataset as a function of (C ,γ), (C , ϵ) and (γ, ϵ) respectively. The bar represents the MSE, and the dark navy region represents the lowest MSE, where bayesian optimization algorithm should settle in.

Conclusion

Hyperparamter tuning methods fundamentally integrates grid search algorithm for possible combinations. However, by integrating Bayesian Optimization, we can improve the Random Grid Search by guiding each random search towards the region of high performance. By comparing the loss between models with optimized and unoptimized hyperparameters, we can easily tell that not only the model's performance has improved but also validate that our Bayesian Optimization algorithm truly guided our random search towards a low-error region. On the other hand, further study is needed on the selection of GP kernel and its parameter during our belief update, for we

have manually chosen our GP kernel and parameters for this project. Furthermore, initial search bounds for hyperparameters were arbitrarily set to [-4,1], which is a sufficient bound when converted to a log-scale, when searching for kernel hyperparameters.

References

[1] Huijskens, Thomas, “Bayesian Optimization with Scikit-learn”. https://thuijskens.github.io/2016/12/29/bayesian-optimisation/

[2] Jiang, Mingfeng, et al. "Study on parameter optimization for support vector regression in solving the inverse ECG problem." Computational and mathematical methods in medicine 2013 (2013).

[3] Snoek, Jasper, Hugo Larochelle, and Ryan P. Adams. "Practical bayesian optimization of machine learning algorithms." Advances in neural information processing systems. 2012.