Supporting Information (SI)

Polymer Sequence Design via Molecular Simulation-Based Active Learning

Praneeth S Ramesh and Tarak K Patra[*]

([*]Corresponding Author, E-mail: tpatra@iitm.ac.in)

Department of Chemical Engineering, Center for Atomistic Modeling and Materials Design and Center for Carbon Capture Utilization and Storage, Indian Institute of Technology Madras, Chennai, TN 600036, India

1) <u>Gaussian Process Regression</u>

A Gaussian Process (GP) is an extension of the multivariate Gaussian distribution to an infinite-dimension stochastic process for which any finite combination of dimensions will be a Gaussian distribution. Just as a Gaussian distribution is a distribution over a random variable, completely specified by its mean and covariance, a GP is a distribution over functions[1], completely specified by its mean function, *m* and covariance function, *k*:

$$f(x) \sim GP\ (m(x), k(x_i, x_j))$$

A GP is analogous to a function, but instead of returning a scalar *f(x)* for an arbitrary x, it returns the mean and variance of a normal distribution over the possible values of *f* at *x*. The Gaussian Process Regression (GPR) algorithm implements Gaussian processes for regression with the construct, $Y = f(X) + \varepsilon$, wherein the unknown function *f* is assumed to follow a $GP\ (m(x),\ k(x_i, x_j))$. The training set $D_{1:\,t}$ is constituted by the set of ordered pairs $\{X_{1:\,t}, Y_{1:\,t}\}$. wherein $X_{1:\,t} = [x_1,\ x_2, \ldots, x_t]^T$ denotes the copolymer sequences and $Y_{1:t} = [y_1,\ y_2, \ldots, y_t]^T$ denotes the corresponding radius of gyration respectively, where *t* stands for the total number of sequences for which radius of gyration values have been computed from previous iterations. $\varepsilon \sim N(0, \sigma_n^2)$, stands for independently identically distributed Gaussian noise with mean zero and variance σ_n^2. The objective of the GPR is to fit the training data $D_{1:\,t}$ and to generate reasonable and meaningful predictions for other copolymer sequence data points for which the radius of gyration values are not yet known. For convenience, we assume that the prior mean function is the zero-function *m(x) = 0*. In the context of Gaussian processes, the covariance functions, $k(x_i, x_j)$[2], which are termed as kernels, and they represent the "similarity" between pairs of data-points in the copolymer sequence space. Choosing the appropriate kernel requires a rigorous analysis combining strategies such as hierarchical Bayesian model selection or/and cross-validation. Out of the many kernels that are popularly used, the radial basis function kernel and Matern kernel[3] are chosen for this work. These kernels are stationary kernels, and are invariant to translations in the X-space as they depend only on the distance between two data points and not their absolute values.

1) The **radial basis function (RBF) kernel** can be represented as

$$k\left(x_i\ ,\ x_j\right) = exp\left(-\frac{(\|x-x'\|_2)^{\ 2}}{2\ l^2}\right),$$

where l is the characteristic length scale of the kernel and $\|x-x'\|_2$ is the Euclidean distance measure or the L2 norm, which is the length of the line segment joining x and x'.

2) Matern Kernel generalizes the popularly used radial basis function kernel by incorporating a parameter ν which controls the smoothness and is thus more flexible as it includes an additional free parameter in the model. In this work ν = 3/2 is considered, resulting in the following functional form of the Matern 3/2 kernel:

$$k\left(x_i\ ,\ x_j\right) = \left(1+\frac{\sqrt{3}\,\|x-x'\|_2}{l}\right)exp\left(-\frac{\sqrt{3}\ \|x-x'\|_2}{l}\right)$$

l is the characteristic length scale and $\|x-x'\|_2$ is the Euclidean distance measure or the L2 norm.

Thus, the kernel matrix K is given by:

$$\boldsymbol{K} = \begin{bmatrix} k(x_1,x_1) & \cdots & k(x_1,x_t) \\ \vdots & \ddots & \vdots \\ k(x_t,x_1) & \cdots & k(x_t,x_t) \end{bmatrix}$$

With this framework, it would seem that, for either of the two kernels, RBF or Matern, covariance values represented by the diagonal elements are 1, since each data point is perfectly correlated with itself. However, such a scenario is possible only in a noise-free environment[3], and the very modeling of the regression problem $Y = f(X) + \ \varepsilon$ with the inclusion of Gaussian Noise $\varepsilon \sim N(0,\sigma_n^2)$ is to facilitate smoothing over noisy data. Accordingly, in practical implementation, an additional term, σ_n^2 is added to all the diagonal elements, resulting in the following form of Kernel matrix. Thus, the training set $Y_{1:t}$ which is a collection of functions $[f(x_1), f(x_2), \ldots, f(x_t)]$ follows multivariate normal distribution $N(0,K')$

$$\boldsymbol{K'} = \begin{bmatrix} k(x_1,x_1)+\sigma_n^2 & \cdots & k(x_1,x_t) \\ \vdots & \ddots & \vdots \\ k(x_t,x_1) & \cdots & k(x_t,x_t)+\ \sigma_n^2 \end{bmatrix}$$

$$\boldsymbol{K'} = K + \sigma_n^2 I$$

Moreover, the addition of σ_n^2 to the diagonal ensures that the calculated values form a positive definite matrix during the process of model fitting and thus prevents a potential numerical issue.

With the current information on the R_g values for the t sequences, GPR helps in predicting the R_g values for sequences that are not part of $D_{1:t}$. To predict the function values f_* at a test location x_{test}, the joint distribution of $Y_{1:t} = [y_1, \ y_2, \ldots, y_t]^T$ and f_* is constructed. By the properties of Gaussian processes, $Y_{1:t}$ and f_* are jointly Gaussian:

$$\begin{bmatrix} y_{1:t} \\ f_* \end{bmatrix} \sim N\left(0, \begin{bmatrix} \boldsymbol{K} + \sigma_n^2 I & \boldsymbol{K}_* \\ \boldsymbol{K}_*^T & \boldsymbol{k}(\boldsymbol{x}_{test}, \boldsymbol{x}_{test}) + \sigma_n^2 \end{bmatrix}\right)$$

where

$$\boldsymbol{K} = \begin{bmatrix} k(x_1, x_1) & \cdots & k(x_1, x_t) \\ \vdots & \ddots & \vdots \\ k(x_t, x_1) & \cdots & k(x_t, x_t) \end{bmatrix}$$

$\boldsymbol{I}$ is an identity matrix of size $n \times n$

$\boldsymbol{K}_* = [k(x_{test}, x_1), k(x_{test}, x_2), \ldots, k(x_{test}, x_t)]^T$

After performing certain linear algebraic simplifications on the joint distribution, the predictions for the posterior distribution of the radius of gyration at x_{test} can be obtained. The predicted posterior distribution follows a normal distribution with posterior mean $\hat{\mu}$ and posterior variance $\hat{\Sigma}$.

$$P(f_* | D_{1:t}, x_{test}) \sim N(\hat{\mu}, \hat{\Sigma})$$

wherein:

$$\hat{\mu} = {K_*}^T [K + \sigma_n^2 I]^{-1} y_{1:t}$$

$$\hat{\Sigma} = k(x_{test}, x_{test}) + \sigma_n^2 - {K_*}^T (K + \sigma_n^2 I)^{-1} K_*$$

The posterior mean and variance evaluated at any point represent the GPR model's prediction and uncertainty measure respectively.

Implementation:

The implementation of GPR is done using the *gaussian_process.GaussianProcessRegresso*r function of the sklearn library, using the Matern Kernel, and the iterative retraining and predictions are done using the *models.BayesianOptimizer* function of the modAL library.

Hyperparameters:

GPR is a non-parametric method in the sense that **the regression function f has no explicit parametric form** because GP represents a function (i.e. an infinite dimensional vector). As the number of data points increases, so does the number of model 'parameters'. Unlike a parametric model, where the number of parameters remains fixed with respect to the size of the data, the number of parameters grows with the number of data points in nonparametric models. Although GPR models are considered to be non-parametric, their hyperparameters (parameters that cannot be learnt during the training task, but would have to be specified *a priori*) significantly influence their predictive capabilities.

The hyperparameters for the GPR are

1. The characteristic length scale of the kernel, which is denoted as *l,* defines the length scale of the respective feature dimension, and
2. The Gaussian Noise σ_{noise}, which is added to the diagonal of the kernel matrix.

Hyperparameter Tuning:

Hyperparameter tuning comprises a set of strategies to navigate the space of hyperparameters and pinpoint the parameter combination that brings about the best performance of the machine learning model. Cross-validation also known as rotation estimation or out-of-sample testing is a resampling procedure used popularly by Machine Learning practitioners for hyperparameter tuning. However, the hyperparameter tuning in Gaussian Process Regression can be done differently, and in fact faster than the cross-validation approach, by performing gradient descent to maximize the negative log marginal likelihood with respect to the hyperparameters. As the log marginal likelihood might have multiple local optima, the gradient descent optimizer is run repeatedly for over 50 different initializations.

2) Support Vector Regression (Linear Kernel)

SVR is a regression version of support vector machines that transforms data points to higher-dimensional feature space and constructs a nonlinear regression function based on a kernel function. A regression function of the form $f(x) = w^T\phi(x) + b$ is modelled, wherein $\boldsymbol{x}$ is mapped onto a higher-dimensional space by a function $\phi(x)$, while w and b denote the weight vector and the bias parameter respectively.

The first step involves fitting the SVR model on the training data. If we consider the training set $D_{1:t}$ to be constituted by the set of ordered pairs $\{X_{1:t}, Y_{1:t}\}$. wherein $X_{1:t}$ denotes the copolymer sequences and $Y_{1:t}$ denotes the corresponding radius of gyration respectively, the ε-SVR algorithm tries to find a function *f(x)* that has at most ε deviation from the computed target property values $Y_{1:t}$, while simultaneously being as flat as possible. Based on the derivative of *f(x)* with respect to x, $\frac{1}{2}w^Tw$ or $\frac{1}{2}||w||^2$ should be as minimal as possible to ensure flatness. $||w||^2$ is the square of the magnitude of the normal vector to the surface that is being approximated, and is also the L2 norm of w. The magnitude of w provides a control over the flatness of the solution. This optimization problem is defined along with a constraint that the function *f(x)* should be restricted within ε units of the true values, y_i.[4]

$$\underset{w,\ b}{Minimize}\ \frac{1}{2}w^Tw$$

Subject to $w^T\phi(x_i) + b - \epsilon \leq y_i \leq w^T\phi(x_i) + b + \epsilon$ for $i = 1, 2, \dots, t$

However, as much as it is desired to find an optimal function *f* which approximates the target property values y_i within ϵ units of precision, in practice, it is difficult to strictly restrict the deviations between predicted and true values within epsilon for all the data points in the training set, and this might result in the optimization problem becoming infeasible.

A Soft-Margin approach accounts for the possibility that these deviations are potentially likely, and attempts to minimize them as much as possible. The ϵ-insensitive loss function ($|\xi|_\epsilon$) penalizes predictions that are farther than ϵ while ignoring errors lesser than ϵ.

$$(|\xi|_\epsilon)_i = \max\{0, |y_i - (w^T\phi(x_i) + b)| - \epsilon\}$$

$$(|\xi|_\epsilon)_i = 0 \qquad if\ |y_i - (w^T\phi(x_i) + b)| \leq \epsilon$$

$$(|\xi|_\epsilon)_i = |y_i - (w^T\phi(x_i) + b)| - \epsilon \qquad if\ |y_i - (w^T\phi(x_i) + b)| > \epsilon$$

For the sake of ease in linear algebraic simplifications, rather than working with the construct of $|\xi|_\epsilon$, two non-negative slack variables ξ and ξ^* are introduced for each data point. These deviations are added to the objective function prefixed by a hyperparameter C which determines the trade-off between control over flatness and tolerance beyond ϵ.

This results in the optimization problem being reframed as:

Minimize $\underset{w,b,\xi,\xi^*}{Minimize} \frac{1}{2} w^T w + C \sum_{i=1}^{n} (\xi_i + \xi_i^*)$

Subject to $\begin{cases} w^T \phi(x_i) + b - (\epsilon + \xi_i^*) \leq y_i \leq w^T \phi(x_i) + b + (\epsilon + \xi_i) \\ \xi_i, \xi_i^* \geq 0, \quad i = 1, 2, \dots, t \end{cases}$

In the expression, $f(x) = w^T \phi(x) + b$, $\phi(x)$ is generally represented in an implicit form using a kernel function $k(x, x')$ such that $k(x, x') = \phi(x)^T \phi(x')$. Assuming the bias function as zero for the sake of simplicity, and after solving the optimization problem with a Lagrange reformulation, the expression for the SVR functional form at any point x_{test} where we would wish to obtain the prediction value, is given by

$$f^{pred}(x_{test}) = \sum_{i=1}^{t} \alpha_i k(x_{test}, x_i)$$

where i = 1, 2, …, t

where α_i are the weights that are fit to the training data and $k(x_{test}, x_i)$ is be the kernel function that computes the similarity between a pair of points x_{test} and x_i. The summation is performed over the entire training set of size *t*.

Kernels:

Of the many kernels that could be used, linear kernel and radial basis function kernel are employed in this work.

1) The **linear kernel** refers to $k(x, x') = \langle x, x' \rangle$, i.e., the dot product of *x* and *x'*.

2) The **radial basis function kernel** has the functional form,

$$k\ (x, x') = exp^{\left(-\gamma \|x - x'\|_2{}^2\right)}$$

where $\|x - x'\|_2$ is the Euclidean distance measure or the L2 norm, which is the length of the line segment joining x and x' and γ codifies the inverse of the variance of the radial basis function.

As such, these methods do not estimate uncertainties, and thus a nonparametric bootstrap sampling with replacement is implemented to generate multiple subsets of the training dataset to form a distribution. The empirical distribution obtained with bootstrapping mimics the statistics associated with the original population. Thus, an ensemble of 10 models of the same ML method trained on different subsets of the training set, aids us in obtaining robust predictions with measures of uncertainties. The uncertainty in these two methods is the standard deviation of the prediction from the 10 models[5].

Hyperparameters:

- Epsilon (ϵ) determines the width of the ϵ-insensitive zone, or in other words the margin of tolerance for deviations between predicted and true values. A smaller value indicates a lower tolerance for deviations, while a larger value indicates greater tolerance for deviations.
- C is a regularization hyperparameter that can tune the relative importance of minimizing $||w||^2$ vis-à-vis minimizing the slack variables (ξ, ξ^*)
- Gamma (γ), in the case of Radial Basis Kernel codifies the inverse of the variance of the radial basis function, and can be seen as a measure of the spread of the kernel. When gamma is very small, the model is too constrained, and cannot capture the complexity or shape of the data, while very high gamma values result in model overfitting and non-generalizability to newer data points.

Tuning Hyperparameters:

Compared to popularly used forms of automated hyper-parameter tuning such as grid search which involves brute-force exploration of a pre-defined set of hyperparameter combinations or random search which involves random searching of the hyperparameter space iteratively, the Bayesian Optimization approach views the hyperparameter tuning as a classic complex optimization problem. The input variables are the hyperparameters and the objective score function is defined by the average cross-validation error which is intended to be optimized to as low a value as possible. The training dataset is split into a training subset and validation subset multiple times, by considering different subsets, and the errors on the validation subsets are averaged out so as to compute the average cross-validation error. A search space of the hyperparameters needs to be defined by specifying the lower bound and upper bound of the range from which each of the hyperparameters can be chosen from. The Bayesian Optimizer algorithm would navigate the search space following an iterative approach, and converge to optimal hyperparameters, much the same way as how the outer Bayesian Optimization framework is intended to converge toward identifying optimal sequences. In this work, a log scale from 10^{-6} to 1.0 is defined as the range for each of the numeric hyperparameters, Epsilon (ϵ), C and Gamma (γ) (in the case of RBF kernel). This in turn defines the two-dimensional (for the Linear SVR kernel sub-case) or the three-dimensional (for the RBF kernel sub-case) hyperparameter search space. Owing to differences in the randomly chosen initial point, the results of convergence might vary across multiple runs, and it is a wise practice to consider only the solutions that lie within the search space and not on the periphery/boundary of the search space.

<u>Implementation:</u>

The implementation of SVR was done using the svm.SVR function of sklearn library, and the implementation of bootstrapping using ensemble model was done using the ensemble.BaggingRegressor function of the sklearn library. The hyperparameter tuning was done using the BayesSearchCV function of the Scikit-Optimize Library

3) <u>Kernel Ridge Regression</u>

KRR combines the power of the kernel trick with the simplicity of standard least-squares regression. While the form of the model learnt by KRR and the notion of using kernel functions are similar to that of SVR, KRR does not ignore errors smaller than ε, and the squared error is used instead of the absolute error.[6]

If we consider the training set $D_{1:t}$ with ordered pairs $\{X_{1:t}, Y_{1:t}\}$, with $X_{1:t}$ being the copolymer sequences and $Y_{1:t}$ being the corresponding radii of gyration, the expression for the Kernel Ridge Regression functional form at any point x_{test} where we would wish to obtain the prediction value, is given by

$$f^{pred}(x_{test}) = \sum_{i=1}^{t} \alpha_i k(x_{test}, x_i)$$

where i = 1, 2, …, t

where α_i are the weights and $k(x_i, x_{test})$ to be the kernel function that computes the similarity between a pair of points x_i and x_{test}. The summation is performed over the entire training set of size *t*.

The optimized values for the weights α_i are obtained through the minimization of the following cost function by running a summation over the entire training set[7]. The following expression is very similar to the cost function that is minimized in order to estimate the parameters of a standard linear least-squares regression model.

$$\underset{\alpha_1, \alpha_2, .., \alpha_t}{minimize} \sum_{i=1}^{t} (f^{pred}(x_i) - y_i)^2 + \lambda\, \alpha^T \alpha$$

where the hyperparameter λ controls the strength of the regularization

The solution to this is given by

$$\alpha = (K + \lambda\, I)^{-1}\, Y$$

where α is the vector of all α_i values, $[\alpha_1,\ \alpha_2, \dots, \alpha_t]^T$, I is the $t \times t$ identity matrix, K is the kernel matrix constituted by elements $K_{ij} = k\ (x_i\ ,\ x_j)$, i and j spanning the entire training set, and Y is the set of computed target property values $Y_{1:t} = [y_1,\ y_2, \dots, y_t]^T$

For the KRR algorithm, it is possible to get a closed form[8] exact solution for α, which is why fitting a KRR can be done in closed-form and is typically faster compared to SVR for medium-sized datasets.

Kernels:

While there are numerous kernels such as Polynomial, Exponential chi2 and sigmoid kernels that could be potentially used, in this work radial basis function and laplacian kernel are used.

1) The **Radial Basis function** has the functional form: $k\ (x, x') = exp^{\left(-\gamma\left(\|x-x'\|_2\right)^2\right)}$, wherein $\|x - x'\|_2$ is the Euclidean distance measure or the L2 norm, which is the length of the line segment joining x and x' and γ is a measure of the spread of the kernel

2) The **Laplacian function** has the functional form $k\ (x, x') = exp^{\left(-\gamma\|x-x'\|_1\right)}$, wherein $\|x - x'\|_1$ stands for the Manhattan distance or L1 norm which is the distance between x and x' measured only along perpendicular axes and γ is a measure of the spread of the kernel

Hyperparameters:

- Lambda (λ) is used to codify the extent of regularization. Larger values specify stronger regularization, and result in lower variance, thereby avoiding overfitting the function to the training data.

- Gamma (γ) is the measure of the spread of the kernel, and has a similar implication as it has for SVR.

Tuning Hyperparameters:

Like in SVR, the hyperparameter tuning is done by a Bayesian approach. In this work, a log scale from 10^{-6} to 1.0 is defined as the range for each of the numeric hyperparameters Lambda (λ) and Gamma (γ) and this in turn defines the two-dimensional search space that would be searched by the Bayesian Optimizer. The Bayesian optimizer would navigate the search space following an iterative approach to zero down on an optimal set of hyperparameters.

Implementation:

The implementation of KRR was done using the kernel_ridge.KernelRidge function of sklearn library, and the implementation of bootstrapping using the ensemble model was done using the ensemble.BaggingRegressor function of the sklearn library. The hyperparameter tuning was done using the BayesSearchCV function of the Scikit-Optimize Library

Search Space Definition:

During the implementation of the iterative active learning framework, running the surrogate model to make predictions over the entire space of copolymer sequences poses a practical challenge. For the first problem statement in which there is no constraint on the number of As and Bs, the total number of possible copolymer sequences is $\frac{1}{2} \times 2^{100} = 2^{99}$ (halving is done in order to eliminate double-counting of sequences which are exact reverse of each other). At each iteration, a small (relatively small) search space is defined by enumerating the possible sequences that are sequentially proximate to the ones in the training set accumulated until that particular iteration. For every copolymer sequence in the training set, the exhaustive list of copolymer sequences that differ in the monomeric entity at exactly one position (i.e. one position of A replaced by B, or one position of B replaced by A) is considered to populate the search space. This is notionally an equivalent of performing a single mutation operation in the parlance of Genetic Algorithm[9]. The best 32 sequences from this search space are recommended for computation via MD simulation, as selected by a query strategy. This approach of single-mutation enumerated search space is implemented while experimenting with different surrogate models, kernels and query strategies, for all the 30 optimization frameworks.

In addition to this approach, two alternative strategies are attempted as an illustration for the first problem statement, for a specific surrogate model. With these alternative approaches, the premise is to push the boundaries of the optimizer beyond working with a restricted sequentially proximate single-mutation enumerated search space, towards search spaces that are sequentially different to a considerable extent. While numerous such alternative strategies could potentially be defined, the following two have been taken up as illustrative examples in this work.

Alt. Strategy 1:

At each iteration, the search space (S) is defined by a combination of two individual search spaces:

S1 - The exhaustive set of sequences obtained by mutating (replacement of A by B or B by A) at one position in the chain

S2 - The exhaustive set of sequences obtained by mutating at four positions in the chain

$$S = S_1 \cup S_2$$

By implementing the query strategy on each of these search spaces, 16 candidate sequences from S1 and 16 candidate sequences S2 are selected to add up to a sum of total of 32 candidate sequences in every iteration.

Alt. Strategy 2:

At each iteration, the search space (S) is defined by a combination of three individual search spaces:

S1 - The exhaustive set of sequences obtained by mutating (replacement of A by B or B by A) at one position in the chain

S2 - The exhaustive set of sequences obtained by mutating at four positions in the chain

S3 - The exhaustive set of sequences obtained by mutating at ten positions in the chain

$$S = S_1 \cup S_2 \cup S_3$$

In each of these three search spaces, the query strategy is implemented to recommend the best sequences for the computationally expensive MD simulation. 12 candidate sequences from S1 and 10 candidate sequences from each of S2 and S3 are selected to add up to a sum total of 32 candidate sequences in every iteration.

For the second problem statement in which A: B is constrained to be in a 50:50 ratio, the total number of possible sequences is $\frac{1}{2} \times \frac{100!}{50!\,50!}$. The application of mutations would disturb the A: B ratio, and hence, the search space at each iterative step is defined by generating an exhaustive list of copolymer sequences that are obtained by swapping one pair of A and B for every sequence in the training set accumulated until that particular iteration. While certain alternative strategies that have been attempted for the first problem statement, no such similar attempts of defining search spaces by enumerating sequences obtained by swapping 4 A-B pairs or swapping 10 A-B pairs have been explored, on account of rising computational costs for swapping vis-à-vis mutations.

In all of these approaches, the basic premise is that as newer candidate sequences are added into the training set, greater number of sequences would be defined as part of the search space. Thus, a framework is developed to define a candidate search space, the size of which becomes progressively larger at each iterative step.

Query Strategies:

Once the predictions and uncertainties of the R_g values are obtained for the sequences enumerated in the search space at every iterative step, there are different strategies that could be used to shortlist the sequences for which the computationally expensive molecular simulations are to be done. The technique/approach of recommending the best sequences is known under various names such as query strategy / acquisition function / utility function / selector[15].

Purely Exploitative and Purely Explorative Strategies:

Fundamentally, the two ends of the spectrum are Exploitation – recommending candidates which have high predictions of the target property and Exploration – recommending candidates that have high uncertainties. With the exploitation approach, the motive is to expand the training set with candidates that are predicted to have high values of the target property with a reasonably high degree of confidence (low estimates of uncertainty). The motive of the exploration approach is to extend the domain of applicability of the model and to improve the model by using the predicted uncertainties to study regions of sequence space wherein the model predictions have a high estimates of uncertainty or in other words a low degree of confidence[10]. The former approach aims at sampling optimal candidates in the next iteration while the latter approach aims at improving the understanding of the universal sequence- R_g correlation and does not have an explicit motive of optimization. The purely exploitative approach has been reported to be a "greedy" approach[11], and is associated with local optima traps resulting in suboptimal solutions. On the other hand, the purely explorative approach has been reported to have poor convergence towards optimality. Certain query strategies that have their roots in information theory, have successfully demonstrated the ability to escape local optima traps and to improve the convergence towards global optima, and thus efficiently balance the trade-off between exploitation and exploration.

Expected Improvement & Probability of Improvement:

The Query Strategy **Maximization of Expected Improvement (Max EI)**[12] recommends candidate sequences that have the scope to result in the maximum improvement in y (the Radius of Gyration) over the current best value. With t computed values of Radius of gyration, let μ^* be the maximum value that has been observed so far.

$$\mu^* = \underset{i\,=\,1,..,\;t}{max}\; y_i$$

Suppose, x_{test} is an arbitrary point that is considered as a potential candidate for the computation of radius of gyration value. After the computation, the best observed value would be max (y_{test}, μ^*). The difference between these values is the improvement on account of additional computation, and is given by $\max(y_{test}, \mu^*) - \mu^*$ = $\max(y_{test} - \mu^*, 0)$.

The motive is to choose an x_{test} such that the improvement is as large as possible. Rather than attempting to compute the actual improvement, the posterior probability distribution of f(x) can be used to define the expectation of the improvement for each x_{test}, The expectation of the improvement, also termed as expected improvement EI(x) is the expectation of this improvement conditioned on the dataset $D_{1:t}$.

EI(x_{test}) = E (I ($x_{test}|D_{1:t}$)) = E [$\max(f(x_{test}) - \mu^*, 0)|\, D_{1:t}$]

This expression can be computed explicitly in terms of the normal probability density function (pdf) $\phi(.)$ and normal cumulative distribution function cdf $\Phi(.)$ as

$EI(x_{test}) = (\mu(x_{test}) - \mu^*) \, \Phi\left(\frac{\mu(x_{test})-\mu^*}{\sigma(x_{test})}\right) + \sigma(x_{test}) \, \phi\left(\frac{\mu(x_{test})-\mu^*}{\sigma(x_{test})}\right)$

where $\mu(x_{test})$ and $\sigma(x_{test})$ are the predicted estimates of the mean and variance at point x_{test}. This can be written in a concise form as

$EI(x_{test}) = (\sigma(x_{test})) \, [\Phi(z) + z \, \phi(z)]$

where $z = \frac{\mu(x_{test})-\mu^*}{\sigma(x_{test})}$

In the limit of minimal uncertainty $\sigma(x_{test}) \to 0$, candidates will be selected with a $\mu(x_{test})$ greater than the best measured so far (μ^*), by an exploitative approach trusting the predictions of the surrogate model. Similarly, in the other limit $\sigma(x_{test}) \to \infty$, the candidates with the largest uncertainty would be a selected by following an explorative approach. For values between the two extremes, the trade-off between the two would be balanced by the Expected Improvement query strategy, exploring the sequence space thereby avoiding local minima and exploiting the space thereby narrowing on the optimal candidate sequences.

Even with the formulation of the Expected Improvement function, addition of margins[13] is employed in order to allow for finer control in balancing exploration and exploitation. A margin specifies a minimum amount of improvement over the current best point, and is integrated into the equation by replacing the term $z = \frac{\mu(x_{test})-\mu^*}{\sigma(x_{test})}$with $z = \frac{\mu(x_{test})-(\mu^*+\epsilon)}{\sigma(x_{test})}$ where $\epsilon \geq 0$ represents the degree of exploration. The higher the epsilon, more the exploration. This is because increasing ϵ results in querying locations with a larger sigma σ, moving towards exploration rather than exploitation[13].

The query strategy **<u>Maximization of Probability of Improvement (Max. PI)</u>** [12] recommends candidates for R_g computation, by considering the ones which have the highest probability of improvement over the current maximum value μ^*. As in Expected improvement, the addition of margins is incorporated by considering $\epsilon \geq 0$ which represents the degree of exploration.

$$PI(x_{test}) = P\big(I(x_{test})\big) = P(y_{test} \geq \mu^* + \epsilon)$$

Statistically, PI represents the upper-tail probability of the surrogate model.

$$P\big(f(x_{test})\big) \geq \mu^* + \epsilon$$

$$\Rightarrow PI(x_{test}) = \Phi\,(z)$$

where $z = \frac{\mu(x_{test})-(\mu^*+\epsilon)}{\sigma(x_{test})}$ and $\Phi(z)$ stands for the cumulative distribution function.

In this work, a value of 0.01 was taken as ϵ for both query strategies. In either of the query strategy, the top 32 candidate sequences that are found to have the highest EI values / highest PI values are recommended for computation of R_g through Molecular Simulation.

Random Selection:

Amongst the enumerated sequences in the search space at each iteration, 32 candidate sequences are randomly chosen, in an arbitrary manner with absolutely no preferences or biases in the selection.

Reference:

(1) Seko, A.; Maekawa, T.; Tsuda, K.; Tanaka, I. Machine Learning with Systematic Density-Functional Theory Calculations: Application to Melting Temperatures of Single- and Binary-Component Solids. *Phys. Rev. B* **2014**, *89* (5), 054303. https://doi.org/10.1103/PhysRevB.89.054303.

(2) Duvenaud, D. Advice on Covariance Functions. 14.

(3) Brochu, E.; Cora, V. M.; de Freitas, N. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. *ArXiv10122599 Cs* **2010**.

(4) Awad, M.; Khanna, R. Open Access Chapter First Online: 27 April 2015. 31.

(5) Yuan, R.; Liu, Z.; Balachandran, P. V.; Xue, D.; Zhou, Y.; Ding, X.; Sun, J.; Xue, D.; Lookman, T. Accelerated Discovery of Large Electrostrains in BaTiO $_3$ -Based Piezoelectrics Using Active Learning. *Adv. Mater.* **2018**, *30* (7), 1702884. https://doi.org/10.1002/adma.201702884.

(6) Stulp, F.; Sigaud, O. Many Regression Algorithms, One Unified Model: A Review. *Neural Netw.* **2015**, *69*, 60–79. https://doi.org/10.1016/j.neunet.2015.05.005.

(7) Seko, A.; Maekawa, T.; Tsuda, K.; Tanaka, I. Machine Learning with Systematic Density-Functional Theory Calculations: Application to Melting Temperatures of Single- and Binary-Component Solids. *Phys. Rev. B* **2014**, *89* (5), 054303. https://doi.org/10.1103/PhysRevB.89.054303.

(8) Vu, K.; Snyder, J. C.; Li, L.; Rupp, M.; Chen, B. F.; Khelif, T.; Müller, K.-R.; Burke, K. Understanding Kernel Ridge Regression: Common Behaviors from Simple Functions to Density Functionals. *Int. J. Quantum Chem.* **2015**, *115* (16), 1115–1128. https://doi.org/10.1002/qua.24939.

(9) Meenakshisundaram, V.; Hung, J.-H.; Patra, T. K.; Simmons, D. S. Designing Sequence-Specific Copolymer Compatibilizers Using a Molecular-Dynamics-Simulation-Based Genetic Algorithm. *Macromolecules* **2017**, *50* (3), 1155–1166. https://doi.org/10.1021/acs.macromol.6b01747.

(10) Mannodi-Kanakkithodi, A.; Chandrasekaran, A.; Kim, C.; Huan, T. D.; Pilania, G.; Botu, V.; Ramprasad, R. Scoping the Polymer Genome: A Roadmap for Rational Polymer Dielectrics Design and Beyond. *Mater. Today* **2018**, *21* (7), 785–796. https://doi.org/10.1016/j.mattod.2017.11.021.

(11) Balachandran, P. V.; Xue, D.; Theiler, J.; Hogden, J.; Lookman, T. Adaptive Strategies for Materials Design Using Uncertainties. *Sci. Rep.* **2016**, *6* (1), 19660. https://doi.org/10.1038/srep19660.

(12) Theiler, J.; Zimmer, B. G. Selecting the Selector: Comparison of Update Rules for Discrete Global Optimization. *Stat. Anal. Data Min. ASA Data Sci. J.* **2017**, *10* (4), 211–229. https://doi.org/10.1002/sam.11343.

(13) Jasrasaria, D.; Pyzer-Knapp, E. O. Dynamic Control of Explore/Exploit Trade-Off In Bayesian Optimization. *ArXiv180701279 Cs Stat* **2018**.